Editing Conditional Radiance Fields

Steven Liu[1] Xiuming Zhang[1] Zhoutong Zhang[1] Richard Zhang[2] Jun-Yan Zhu[2,3] Bryan Russell[2]
[1]MIT [2]Adobe Research [3]CMU
http://editnerf.csail.mit.edu/

Abstract

A neural radiance field (NeRF) is a scene model supporting high-quality view synthesis, optimized per scene. In this paper, we explore enabling user editing of a category-level NeRF – also known as a conditional radiance field – trained on a shape category. Specifically, we introduce a method for propagating coarse 2D user scribbles to the 3D space, to modify the color or shape of a local region. First, we propose a conditional radiance field that incorporates new modular network components, including a shape branch that is shared across object instances. Observing multiple instances of the same category, our model learns underlying part semantics without any supervision, thereby allowing the propagation of coarse 2D user scribbles to the entire 3D region (e.g., chair seat). Next, we propose a hybrid network update strategy that targets specific network components, which balances efficiency and accuracy. During user interaction, we formulate an optimization problem that both satisfies the user's constraints and preserves the original object structure. We demonstrate our editing approach on rendered views of three shape datasets and show that it outperforms prior neural editing approaches. Finally, we edit the appearance and shape of a single-view real photograph and show that the edit propagates to extrapolated novel views.

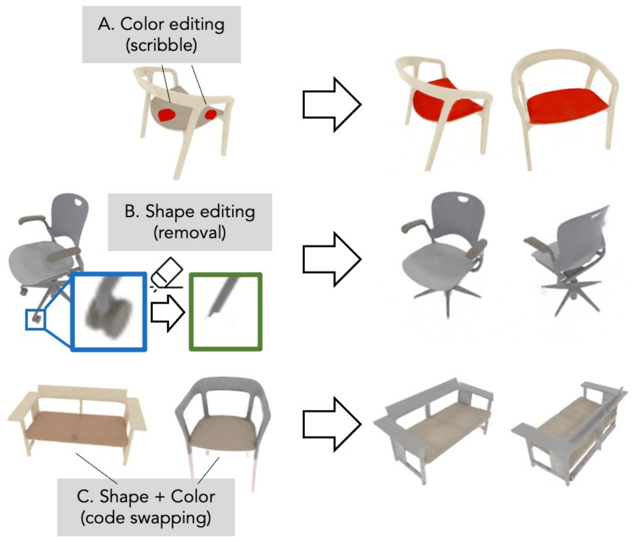
A. Color editing (scribble)
B. Shape editing (removal)
C. Shape + Color (code swapping)

Figure 1: **Editing a conditional radiance field.** Given a conditional radiance field trained over a class of objects, we demonstrate three editing applications: (A) color editing, (B) shape editing, and (C) color/shape transfer. A user provides coarse scribbles over a local region of interest or selects a target object instance. Local edits propagate to the desired region in 3D and are consistent across different rendered views.

1. Introduction

3D content creation often involves manipulating high-quality 3D assets for visual effects or augmented reality applications, and part of a 3D artist's workflow consists of making local adjustments to a 3D scene's appearance and shape [26, 28]. Explicit representations give artists control of the different elements of a 3D scene. For example, the artist may use mesh processing tools to make local adjustments to the scene geometry or change the surface appearance by manipulating a texture atlas [62]. In an artist's workflow, such explicit representations are often created by hand or procedurally generated.

While explicit representations are powerful, there remain significant technical challenges in automatically acquiring a high-quality explicit representation of a real-world scene due to view-dependent appearance, complex scene topology, and varying surface opacity. Recently, implicit continuous volumetric representations have shown high-fidelity capture and rendering of a variety of 3D scenes and overcome many of the aforementioned technical challenges [50, 66, 45, 59, 63]. Such representations encode the captured scene in the weights of a neural network. The neural network learns to render view-dependent colors from point samples along cast rays, with the final rendering obtained via alpha compositing [58]. This representation enables many photorealistic view synthesis applications [41, 47]. However, we lack critical knowledge in how to enable artists' control and editing in this representation.

Editing an implicit continuous volumetric representation is challenging. First, how can we effectively propagate sparse 2D user edits to fill the entire corresponding 3D region in this representation? Second, the neural network for an implicit representation has millions of parameters. It is unclear which parameters control the different aspects of the rendered shape and how to change the parameters according to the sparse local user input. While prior work for 3D editing primarily focuses on editing an explicit representation [62], they do not apply to neural representations.

In this paper, we study how to enable users to edit and control an implicit continuous volumetric representation of a 3D object. As shown in Figure 1, we consider three types of user edits: (i) changing the appearance of a local part to a new target color (e.g., changing the chair seat's color from beige to red), (ii) modifying the local shape (e.g., removing a chair's wheel or swapping in new arms from a different chair), and (iii) transferring the color or shape from a target object instance. The user performs 2D local edits by scribbling over the desired location of where the edit should take place and selecting a target color or local shape.

We address the challenges in editing an implicit continuous representation by investigating how to effectively update a conditional radiance field to align with a target local user edit. We make the following contributions. First, we learn a conditional radiance field over an entire object class to model a rich prior of plausible-looking objects. Unexpectedly, this prior often allows the propagation of sparse user scribble edits to fill a selected region. We demonstrate complex edits without the need to impose explicit spatial or boundary constraints. Moreover, the edits appear consistently when the object is rendered from different viewpoints. Second, to more accurately reconstruct shape instances, we introduce a shape branch in the conditional radiance field that is shared across object instances, which implicitly biases the network to encode a shared representation whenever possible. Third, we investigate which parts of the conditional radiance field's network affect different editing tasks. We show that shape and color edits can effectively take place in the later layers of the network. This finding motivates us to only update these layers and enables us to produce effective user edits with significant computational speed-up. Finally, we introduce color and shape editing losses to satisfy the user-specified targets, while preserving the original object structure.

We demonstrate results on rendered views of three shape datasets with varying levels of appearance, shape, and training view complexity. We show the effectiveness of our approach for object view synthesis as well as color and shape editing, compared to prior neural editing methods. Moreover, we show that we can edit the appearance and shape of a real single-view photograph and that the edit propagates to extrapolated novel views. We highly encourage viewing our video to see our editing demo in action. Code and more results are available at our GitHub repo and website.

2. Related Work

Our work is related to novel view synthesis and interactive appearance and shape editing, which we review here.

Novel view synthesis. Photorealistic view synthesis has a storied history in computer graphics and computer vision, which we briefly summarize here. The goal is to infer the scene structure and view-dependent appearance given a set of input views. Prior work reasons over an explicit [11, 15, 22, 72] or discrete volumetric [20, 33, 36, 40, 44, 55, 64, 65, 67, 68, 80, 82] representation of the underlying geometry. However, both have fundamental limitations – explicit representations often require fixing the structure's topology and have poor local optima, while discrete volumetric approaches scale poorly to higher resolutions.

Instead, several recent approaches implicitly encode a continuous volumetric representation of shape [14, 21, 38, 39, 42, 43, 48, 50, 54, 61] or both shape and view-dependent appearance [41, 45, 47, 59, 63, 66, 69, 76, 12, 73] in the weights of a neural network. These latter approaches overcome the aforementioned limitations and have resulted in impressive novel-view renderings of complex real-world scenes. Closest to our approach is Schwarz *et al.* [63, 12], where they build a generative radiance field over an object class and include latent vectors for the shape and appearance of an instance. Different from their method, we include a branch in our neural network that inductively biases the network to capture common features across the shape class. As we will demonstrate, this inductive bias more accurately captures the shape and appearance of the class. Moreover, we do not require an adversarial loss to train our network and instead optimize a photometric loss, which allows our approach to directly align to a single view of a novel instance. Finally, our work is the first to address the question of how to enable a user to make local edits in this new representation.

Interactive appearance and shape editing. There has been much work on interactive tools for selecting and cloning regions [2, 35, 56, 60] and editing single still images [3, 6, 7, 34]. Recent works have focused on integrating user interactions into deep networks either through optimization [1, 9, 81, 10] or a feed-forward network with user-guided inputs [79, 57, 52, 49]. Here, we are concerned with editing 3D scenes, which has received much attention in the computer graphics community. Example interfaces include 3D shape drawing and shape editing using inflation heuristics [27], stroke alignment to a depicted shape [13], and learned volumetric prediction from multi-view user strokes [16]. There has also been work to edit the appearance of a 3D scene, e.g., via transferring multi-channel edits to other views [24], scribble-based material transfer [4], editing 3D shapes in a voxel representation [37], and relighting a scene with a paint brush interface [53]. Finally, there has been work on editing light fields [25, 29, 30]. We encourage the interested reader to review this survey on artistic editing of appearance, lighting, and material [62]. These prior works operate over light fields or explicit/discrete volumetric geometry whereas we seek to incorporate user edits in learned

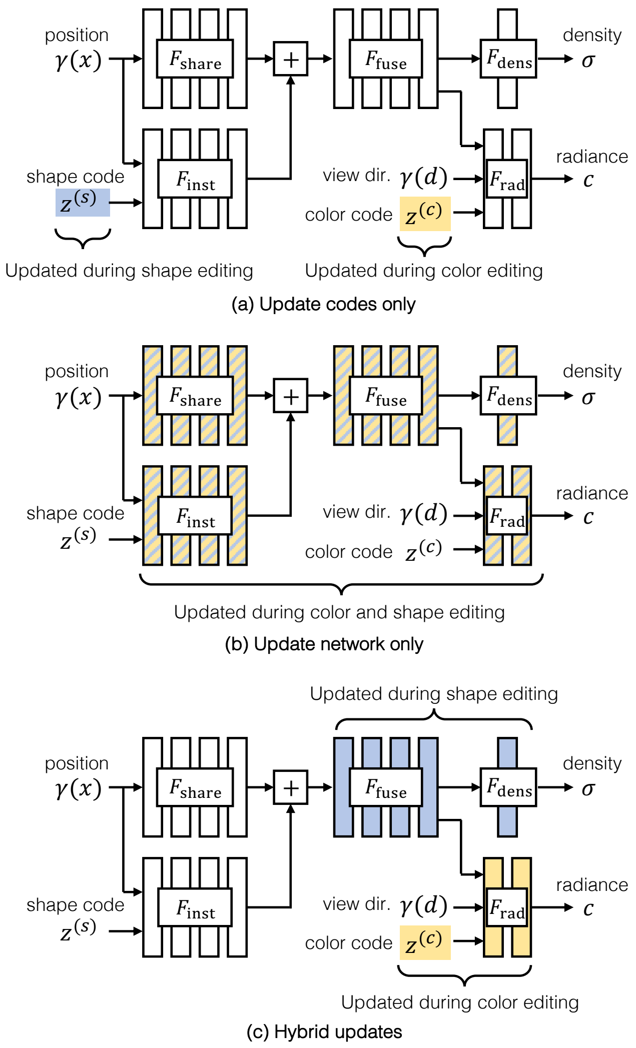

Figure 2: **Conditional radiance field network.** Our network maps a 3D location $\boldsymbol{x}$, viewing direction $\boldsymbol{d}$, and instance-specific shape code $\boldsymbol{z}^{(s)}$ and color code $\boldsymbol{z}^{(c)}$ to radiance $\boldsymbol{c}$ and scalar density σ. The network is composed of modular parts for better shape and color disentanglement. We train our network over a collection of 3D objects (Section 3.1). As highlighted, only a subset of the network components need to be updated during editing (Section 3.2).

implicit continuous volumetric representations.

A closely related concept is edit propagation [3, 19, 23, 74, 77], which propagates sparse user edits on a single image to an entire photo collection or video. In our work, we aim to propagate user edits to volumetric data for rendering under different viewpoints. Also relevant is recent work on applying local "rule-based" edits to a trained generative model for images [8]. We are inspired by the above approaches and adapt it to our new 3D neural editing setting.

3. Editing a Conditional Radiance Field

Our goal is to allow user edits of a continuous volumetric representation of a 3D scene. In this section, we first describe a new neural network architecture that more accurately captures the shape and appearance of an object class. We then describe how we update network weights to achieve color and shape editing effects.

To achieve this goal, we build upon the recent neural radiance field (NeRF) representation [45]. While the NeRF representation can render novel views of a particular scene, we seek to enable editing over an entire shape class, e.g., "chairs". For this, we learn a conditional radiance field model that extends the NeRF representation with latent vectors over shape and appearance. The representation is trained over a set of shapes belonging to a class, and each shape instance is represented by latent shape and appearance vectors. The disentanglement of shape and appearance allows us to modify certain parts of the network during editing.

Let $\boldsymbol{x} = (x, y, z)$ be a 3D location, $\boldsymbol{d} = (\phi, \theta)$ be a viewing direction, and $\boldsymbol{z}^{(s)}$ and $\boldsymbol{z}^{(c)}$ be the latent shape and color vectors, respectively. Let $(\boldsymbol{c}, \sigma) = \mathcal{F}\big(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{z}^{(s)}, \boldsymbol{z}^{(c)}\big)$ be the neural network for a conditional radiance field that returns a radiance $\boldsymbol{c} = (r, g, b)$ and a scalar density σ. The network $\mathcal{F}$ is parametrized as a multi-layer perceptron (MLP) such that the density output σ is independent of the viewing direction, while the radiance $\boldsymbol{c}$ depends on both position and viewing direction.

To obtain the color at a pixel location for a desired camera location, first, N_c 3D points $\{t_i\}_{i=1}^{N_c}$ are sampled along a cast ray $\boldsymbol{r}$ originating from the pixel location (ordered from near to far). Next, the radiance and density values are computed at each sampled point with network $\mathcal{F}$. Finally, the color is computed by the "over" compositing operation [58]. Let $\alpha_i = 1 - \exp\left(-\sigma_i \delta_i\right)$ be the alpha compositing value of sampled point t_i and $\delta_i = t_{i+1} - t_i$ be the distance between the adjacent sampled points. The compositing operation, which outputs pixel color $\hat{C}$, is the weighted sum:

$$\hat{C}\left(\boldsymbol{r}, \boldsymbol{z}^{(s)}, \boldsymbol{z}^{(c)}\right) = \sum_{i=1}^{N_c - 1} c_i \alpha_i \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right). \quad (1)$$

Next, we describe details of our network architecture and our training and editing procedures.

3.1. Network with Shared Branch

NeRF [45] finds the inductive biases provided by positional encodings and stage-wise network design critical. Similarly, we find the architectural design choices important and aim for a modular model, providing an inductive bias for shape and color disentanglement. These design choices allow for selected submodules to be finetuned during user editing (discussed further in the next section), enabling more efficient downstream editing. We illustrate our network architecture $\mathcal{F}$ in Figure 2.

First, we learn a category-specific geometric representation with a *shared shape network* $\mathcal{F}_{\text{share}}$ that only operates on the input positional encoding $\gamma(\boldsymbol{x})$ [45, 70] to capture shared features across instances in the shape category. To modify the representation for a specific shape, an *instance-specific shape network* $\mathcal{F}_{\text{inst}}$ is conditioned on both the shape code $\boldsymbol{z}^{(s)}$ and input positional encoding. The representations are added and modified by a *fusion shape network* $\mathcal{F}_{\text{fuse}}$. To obtain the density prediction σ, the output of $\mathcal{F}_{\text{fuse}}$ is passed to a linear layer, the *output density network* $\mathcal{F}_{\text{dens}}$. To obtain the radiance prediction $\boldsymbol{c}$, the output of $\mathcal{F}_{\text{fuse}}$ is concatenated with the color code $\boldsymbol{z}^{(c)}$ and encoded viewing direction $\gamma(\boldsymbol{d})$ and passed through a two-layer MLP, the *output radiance network* $\mathcal{F}_{\text{rad}}$. We have tried separating $\mathcal{F}_{\text{rad}}$ into a shared radiance branch and an instance-specific branch, but have found that performance does not improve. We follow Mildenhall *et al*. [45] for training and jointly optimize the latent codes via backpropagation through the network. We provide additional training details in the supplement.

3.2. Editing via Modular Network Updates

We are interested in editing an instance encoded by our conditional radiance field. Given a rendering by the network $\mathcal{F}$ with shape $\boldsymbol{z}_k^{(s)}$ and color $\boldsymbol{z}_k^{(c)}$ codes, we desire to modify the instance given a set of user-edited rays by optimizing a loss over the network parameters and shared codes.

Our first goal is to conduct the edit accurately – the edited radiance field should render views of the instance that reflect the user's desired change. Our second goal is to conduct the edit efficiently. Editing a radiance field is time-consuming, as modifying weights requires dozens of forward and backward calls. Instead, the user should receive interactive feedback on their edits. To achieve these two goals, we consider the following strategies for selecting which parameters to update during editing.

Update the shape and color codes. One approach to this problem is to only update the latent codes of the instance, as illustrated in Figure 2(a). While optimizing such few parameters leads to a relatively efficient edit, as we will show, this method results in a low-quality edit.

Update the entire network. Another approach is to update all weights of the network, shown in Figure 2(b). As we will show, this method is slow and can lead to unwanted changes in unedited regions of the instance.

Hybrid updates. Our proposed solution, shown in Figure 2(c), achieves both accuracy and efficiency by updating specific layers of the network. To reduce computation, we finetune the later layers of the network only. These choices speed up the optimization by only computing gradients over the later layers instead of over the entire network. When editing colors, we update only $\mathcal{F}_{\text{rad}}$ and $\boldsymbol{z}^{(c)}$ in the network, which reduces optimization time by $3.7\times$ over optimizing the whole network (from 972 to 260 seconds). When editing shape, we update only $\mathcal{F}_{\text{fuse}}$ and $\mathcal{F}_{\text{dens}}$, which reduces optimization time by $3.2\times$ (from 1,081 to 342 seconds).

In Section 4.3, we further quantify the tradeoff between edit accuracy and efficiency. To further reduce computation, we take two additional steps during editing.

Subsampling user constraints. During training, we sample a small subset of user-specified rays. We find that this choice allows optimization to converge faster, as the problem size becomes smaller. For editing color, we randomly sample 64 rays and for editing shape, we randomly sample a subset of 8,192 rays. With this method, we obtain $24\times$ speedups for color edits and $2.9\times$ speedups for shape edits. We empirically find that a lower sampling for color (and higher for shapes) works well for the evaluated datasets as there is more variation in the shape than the colors. When subsampling user scribble rays, we find that there is a tradeoff between optimization speed and edit quality; please refer to the supplement for additional discussion.

Feature caching. NeRF rendering is slow, especially when rendering high-resolution views. To optimize view rendering during color edits, we cache the outputs of the network that are unchanged during the edit. Because we only optimize $\mathcal{F}_{\text{rad}}$ during color edits, the input to $\mathcal{F}_{\text{rad}}$ is unchanged during editing. Therefore, we cache the input features for each of the views displayed to the user to avoid unnecessary computation. This optimization reduces the rendering time for a 256×256 image by $7.8\times$ (from 6.2 to under 0.8 seconds).

We also apply feature caching during optimization for shape and color edits. Similarly, we cache the outputs of the network that are unchanged during the optimization process to avoid unnecessary computation. Because the set of training rays is small during optimization, this caching is computationally feasible. We accelerate color edits by $3.2\times$ and shape edits by $1.9\times$.

3.3. Color Editing Loss

In this section, we describe how to perform color edits with our conditional radiance field representation. To edit the color of a shape instance's part, the user selects a desired color and scribbles a foreground mask over a rendered view indicating where the color should be applied. The user may also scribble a background mask where the color should remain unchanged. These masks do not need to be detailed; instead, a few coarse scribbles for each mask suffice. We find that the final edited model is not sensitive to the user scribbles; different scribbles of the same foreground/background regions lead to visually identical results. Given the desired target color and foreground/background masks, we seek to update the neural network $\mathcal{F}$ and the latent color vector $\boldsymbol{z}^{(c)}$ for the object instance to respect the user constraints.

Let $\boldsymbol{c}_f$ be the desired color for a ray $\boldsymbol{r}$ at a pixel location within the foreground mask provided by the user scribble and let $y_f = \{(\boldsymbol{r}, \boldsymbol{c}_f)\}$ be the set of ray color pairs provided by the forground user scribble. For a ray $\boldsymbol{r}$ at a pixel location in the background mask, let $\boldsymbol{c}_b$ be the original rendered color at the ray location. Let $y_b = \{(\boldsymbol{r}, \boldsymbol{c}_b)\}$ be the set of rays and colors provided by the background user scribble.

Given the user edit inputs (y_f, y_b), we define our reconstruction loss as the sum of squared-Euclidean distances between the output colors from the compositing operation $\hat{C}$ to the target foreground and background colors:

$$\begin{aligned}\mathcal{L}_{\text{rec}} = &\sum_{(\boldsymbol{r}, \boldsymbol{c}_f) \in y_f} \left|\left|\hat{C}\left(\boldsymbol{r}, \boldsymbol{z}^{(s)}, \boldsymbol{z}^{(c)}\right) - \boldsymbol{c}_f\right|\right|^2 \\ &+ \sum_{(\boldsymbol{r}, \boldsymbol{c}_b) \in y_b} \left|\left|\hat{C}\left(\boldsymbol{r}, \boldsymbol{z}^{(s)}, \boldsymbol{z}^{(c)}\right) - \boldsymbol{c}_b\right|\right|^2. \end{aligned} \quad (2)$$

Furthermore, we define a regularization term $\mathcal{L}_{\text{reg}}$ to discourage large deviations from the original model by penalizing the squared difference between original and updated model weights.

We define our *color editing loss* as the sum of our reconstruction loss and our regularization loss

$$\mathcal{L}_{\text{color}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{reg}} \cdot \mathcal{L}_{\text{reg}}. \quad (3)$$

We optimize this loss over the latent color vector $\boldsymbol{z}^{(c)}$ and $\mathcal{F}_{\text{rad}}$ with $\lambda_{\text{reg}} = 10$.

3.4. Shape Editing Loss

For editing shapes, we describe two operations – shape part removal and shape part addition, which we outline next.

Shape part removal. To remove a shape part, the user scribbles over the desired removal region in a rendered view via the user interface. We take the scribbled regions of the view to be the foreground mask, and the non-scribbled regions of the view as the background mask. To construct the editing example, we whiten out the regions corresponding to the foreground mask.

Given the editing example, we optimize a density-based loss that encourages the inferred densities to be sparse. Let $\sigma_{\boldsymbol{r}}$ be a vector of inferred density values for sampled points along a ray $\boldsymbol{r}$ at a pixel location and let y_f be the foreground set of rays for the entire user scribble.

We define the density loss $\mathcal{L}_{\text{dens}}$ as the sum of entropies of the predicted density vectors $\sigma_{\boldsymbol{r}}$ at foreground ray locations $\boldsymbol{r}$,

$$\mathcal{L}_{\text{dens}} = -\sum_{\boldsymbol{r} \in y_f} \sigma_{\boldsymbol{r}}^{\intercal} \log\left(\sigma_{\boldsymbol{r}}\right), \quad (4)$$

where we normalize all density vectors to be unit length. Penalizing the entropy along each ray encourages the inferred densities to be sparse, causing the model to predict zero density on the removed regions.

We define our *shape removal loss* as the sum of our reconstruction, density, and our regularization losses

$$\mathcal{L}_{\text{remove}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{dens}} \cdot \mathcal{L}_{\text{dens}} + \lambda_{\text{reg}} \cdot \mathcal{L}_{\text{reg}}. \quad (5)$$

We optimize this loss over $\mathcal{F}_{\text{dens}}$ and $\mathcal{F}_{\text{fuse}}$ with $\lambda_{\text{dens}} = 0.01$ and $\lambda_{\text{reg}} = 10$.

The above method of obtaining the editing example assumes that the desired object part to remove does not occlude any other object part. We describe an additional slower method for obtaining the editing example which deals with occlusions in the supplement.

Shape part addition. To add a local part to a shape instance, we fit our network to a composite image comprising a region from a new object pasted into the original. To achieve this, the user first selects a original rendered view to edit. Our interface displays different instances under the same viewpoint and the user selects a new instance from which to copy. Then, the user copies a local region in the new instance by scribbling on the selected view. Finally, the user scribbles in the original view to select the desired paste location. For a ray in the paste location in the modified view, we render its color by using the shape code from the new instance and the color code from the original instance. We denote the modified regions of the composite view as the foreground region, and the unmodified regions as the background region.

We define our *shape addition loss* as the sum of our reconstruction and our regularization losses

$$\mathcal{L}_{\text{add}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{reg}} \cdot \mathcal{L}_{\text{reg}} \quad (6)$$

and optimize over $\mathcal{F}_{\text{dens}}$ and $\mathcal{F}_{\text{fuse}}$ with $\lambda_{\text{reg}} = 10$.

We note that this shape addition method can be slow due to the large number of training iterations. In the supplement, we describe a faster but less effective method which encourages inferred densities to match the copied densities.

Please refer to our video to see our editing demo in action.

4. Experiments

In this section, we show the qualitative and quantitative results of our approach, perform model ablations, and compare our method to several baselines.

Datasets. We demonstrate our method on three publicly available datasets of varying complexity: chairs from the PhotoShape dataset [51] (large appearance variation), chairs from the Aubry chairs dataset [5, 18] (large shape variation), and cars from the GRAF CARLA dataset [63, 17] (single view per instance). For the PhotoShape dataset, we use 100 instances with 40 training views per instance. For the Aubry chairs dataset, we use 500 instances with 36 training views per instance. For the CARLA dataset, we use 1,000 instances and have access to only a single training view per instance. For this dataset, to encourage color consistency across views, we regularize the view direction dependence of radiance, which we further study in the supplement. Furthermore, due to having access to only one view per instance, we forgo quantitative evaluation on the CARLA dataset and instead provide a qualitative evaluation.

	PhotoShapes [51]		Aubry *et al.* [5]	
	PSNR ↑	LPIPS ↓	PSNR ↑	LPIPS ↓
1) Single NeRF [45]	17.81	0.435	14.26	0.390
2) + Learned Latent Codes	36.50	0.029	20.93	0.164
3) + Sep. Shape/Color Codes	36.88	0.028	21.54	0.153
4) + Share./Inst. Net (Ours)	**37.67**	**0.022**	**21.78**	**0.141**
5) NeRF Separate Instances	37.31	0.035	24.15	0.041

Table 1: **Ablation study.** We evaluate our model and several ablations on view reconstruction. Notice how separating the shape and color codes and using the shared/instance network improves the view synthesis quality. Our model even outperforms single-instance NeRF models (each trained on one object).

Implementation details. Our shared shape, instance-specific shape, and fusion shape networks $\mathcal{F}_{\text{share}}, \mathcal{F}_{\text{inst}}, \mathcal{F}_{\text{fuse}}$ are all 4 layers deep, 256 channels wide MLPs with ReLU activations and outputs 256 dimensional features. The shape and color codes are both 32-dimensional and jointly optimized with the conditional radiance field model using the Adam optimizer [32] and a learning rate of 10^{-4}. Additional implementation details are included in the supplement.

4.1. Conditional Radiance Field Training

Our method accurately models the shape and appearance differences across instances. To quantify this, we train our conditional radiance field on the PhotoShapes [51] and Aubry chairs [5] datasets and evaluate the rendering accuracy on held-out views over each instance. In Table 1, we measure the rendering quality with two metrics: PSNR and LPIPS [78]. In the supplement, we provide additional evaluation using the SSIM metric [71]. We find our model renders realistic views of each instance and, on the PhotoShapes dataset, matches the performance of training independent NeRF models for each instance.

We report an ablation study over the architectural choices of our method in Table 1. First, we train a standard NeRF [45] over each dataset (Row 1). Then, we add a 64-dimensional learned code for each instance to the standard NeRF and jointly train the code and the NeRF (Row 2). The learned codes are injected wherever positional or directional embeddings are injected in the original NeRF model. While this choice is able to model the shape and appearance differences across the instances, we find that adding separate shape and color codes for each instance (Row 3) and using a shared shape branch (Row 4) further improves performance. Finally, we report performance when training independent NeRF models on each instance separately (Row 5). In these experiments, we increase the width of the layers in the ablations to keep the number of parameters approximately equal across experiments. Notice how our conditional radiance network outperforms all ablations.

Moreover, we find that our method scales well to more training instances. When training with all 626 instances of the PhotoShape dataset, our method achieves reconstruction PSNR 35.79. We find that the shared shape branch helps our model scale to more instances. In contrast, a model trained without the shared shape branch achieves PSNR 33.91.

	PSNR ↑	LPIPS ↓
Model Rewriting [8]	18.42	0.325
Finetuning Single-Instance NeRF	29.53	0.068
Only Finetune Color Code	26.29	0.090
Finetuning All Weights	31.00	0.050
Our Method	**35.25**	**0.027**

Table 2: **Color editing quantitative results.** We evaluate color editing of a source object instance to match a target instance. Notice that our method outperforms the baselines on all criteria.

4.2. Color Edits

Our method both propagates edits to the desired regions of the instance and generalizes to unseen views of the instance. We show several example color edits in Figure 3. To evaluate our choice of optimization parameters, we conduct an ablation study to quantify our edit quality.

For a source PhotoShapes training instance, we first find a target instance which the model has **not been trained on** in the PhotoShapes chair dataset with an identical shape but a different color. Our goal is to edit the source training instance to match the target instance across all viewpoints. We conduct three edits and in Figure 3 show visual results on two. For each editing example, we manually scribble to roughly span the desired foreground/background regions to conduct the edit. Finally, we render 40 views from the ground truth instance and the edited model, and quantify the difference. The averaged results over the three edits are summarized in Table 2. In the supplement, we evaluate the radiance field based editing methods on 10 edits. Also in the supplement, we provide a method for editing a source *test* instance, which the model has not trained on.

We find that finetuning only the color code is unable to fit the desired edit. On the other hand, changing the entire network leads to large changes in the shape of the instance, as finetuning the earlier layers of the network can affect the downstream density output.

Next, we compare our method against two baseline methods: editing a single-instance NeRF and editing a GAN.

Single-instance NeRF baseline. We train a NeRF to model the source instance we would like to edit, and then apply our editing method to the single instance NeRF. The single instance NeRF shares the same architecture as our model.

GAN editing baselines. We also compare our method to the 2D GAN-based editing method based on Model Rewriting [8]. We first train a StyleGAN2 model [31] on the images of the PhotoShapes dataset [51]. Then, we project unedited test views of the source instance into latent and noise vectors, using the StyleGAN2 projection method [31]. Next, we invert the source and target view into its latent and noise vectors. With these image/latent pairs, we follow the method of

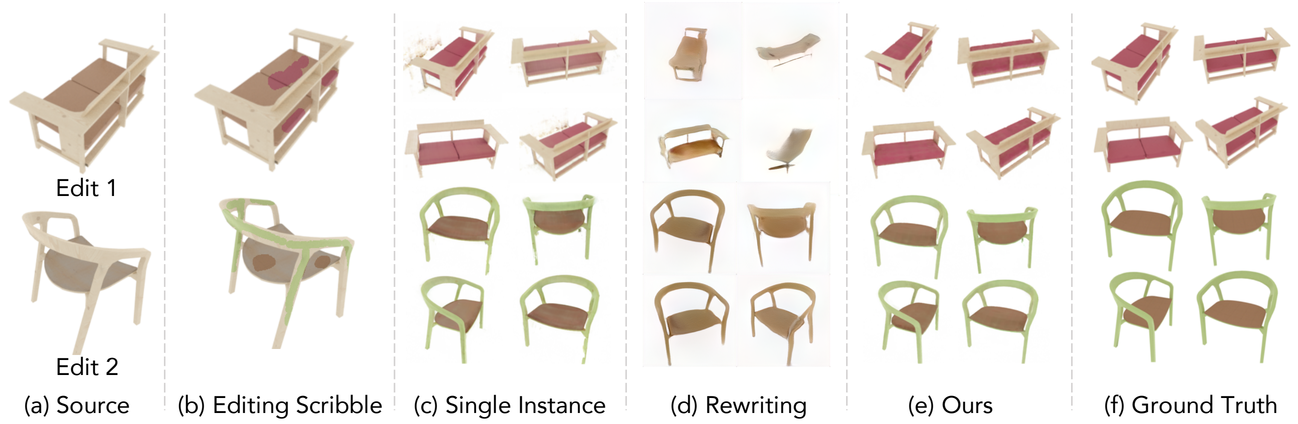

Figure 3: **Color editing qualitative results.** We visualize color editing results where the goal is to match a source instance's colors to a target. Our method accurately captures the colors of the target instance given scribbles over one view. Notice how (c) Editing a Single-Instance NeRF causes visual artifacts, and (d) Rewriting a GAN [8] fails to propagate the edit to unseen views and generates unrealistic outputs.

	PSNR ↑	LPIPS ↓	Time (s) ↓
Only Finetune Shape Code	22.07	0.119	36.9
Only Finetune $\mathcal{F}_{\text{dens}}$	21.84	0.118	**27.2**
Finetuning All Weights	20.31	0.117	66.4
Our Method	**24.57**	**0.081**	37.4

Table 3: **Shape editing quantitative results.** Notice how our hybrid network update approach achieves high visual edit quality while balancing computational cost.

Bau *et al*. [8] and optimize the network to paste the regions of the target view onto the source view. After the optimization is complete, we feed the test set latent and noise vectors into the edited model to obtain edited views of our instance. In the supplement, we provide an additional comparison against naive finetuning of the whole generator.

These results are visualized in Figure 3 and in Table 2. A single-instance NeRF is unable to find an change in the model that generalizes to other views, due to the lack of category-specific appearance prior. Finetuning the model can lead to artifacts in other views of the model and can lead to color inconsistencies across views. Furthermore, 2D GAN-based editing methods fail to correctly modify the color of the object or maintain shape consistency across views, due to the lack of 3D representation.

4.3. Shape Edits

Our method is also able to learn to edit the shape of an instance and propagate the edit to unseen views. We show several shape editing examples in Figure 4. Similar to our analysis of color edits, we evaluate our choice of weights to optimize. For a source Aubry chair dataset training instance, we find a target instance with a similar shape, which the model has **not been trained on**. We then conduct an edit to change the shape of the source instance to the target instance, and quantify the difference between the rendered and ground truth views. The averaged results across three edits are summarized in Table 3 and results of one edit are visualized in the top of Figure 4. In the supplement, we evaluate these editing methods on 10 edits and report error bars. We find that the approaches of only optimizing the shape code and only optimizing $\mathcal{F}_{\text{dens}}$ leave the chair mostly unchanged, while optimizing the whole network leads to removal of the object part, but causes unwanted artifacts in the rest of the object. Instead, our method correctly removes the arms and fills the hole of the chairs, and generalizes this edit to unseen views of each instance.

4.4. Shape/Color Code Swapping

Similar to GRAF [63], our model succeeds in disentangling shape and color. When we change the color code input to the conditional radiance field while keeping the shape code unchanged, the resulting rendered views remain consistent in shape. Our model architecture enforces this consistency, as the density output is independent of the color code.

When changing the shape code input of the conditional radiance field while fixing the color code, the rendered views remain consistent in color. This is surprising because in our architecture, the radiance of a point is a function of both the shape code and the color code. Instead, the model has learned to disentangle color from shape when predicting radiance. These properties let us freely swap around shape and color codes, allowing for the transfer of shape and appearance across instances; we visualize this in Figure 5.

4.5. Real Image Editing

We demonstrate how to infer and edit extrapolated novel views for a single real image given a trained conditional radiance field. We assume that the single image has attributes similar to the conditional radiance field's training data (e.g., object class, background). First, we estimate the image's viewpoint by manually selecting a training set image with similar object pose. Jointly optimizing the estimated pose during training can improve the synthesized views slightly,

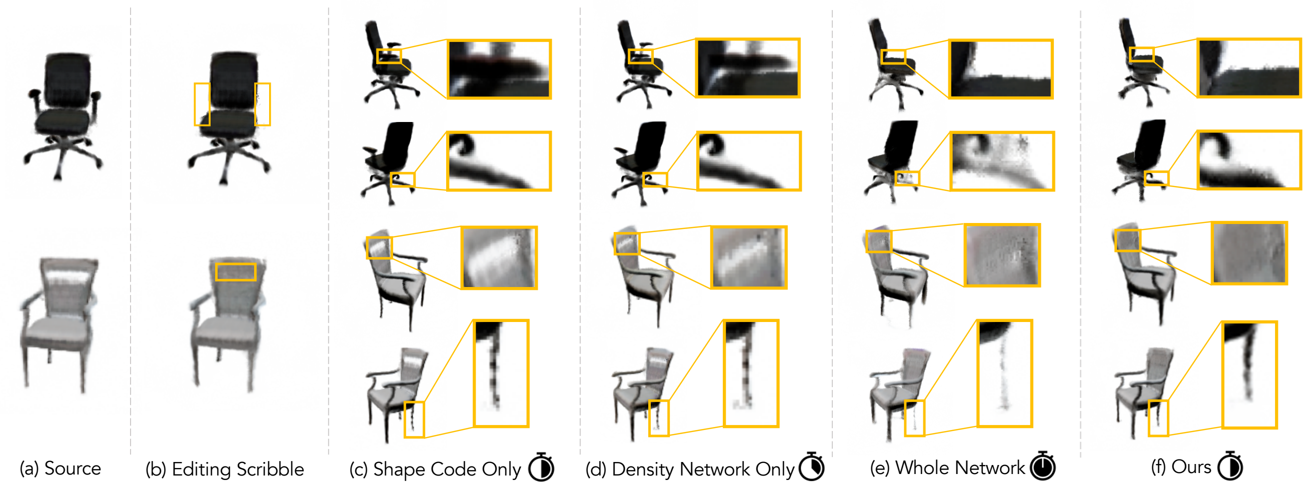

Figure 4: **Shape editing qualitative results.** Our method successfully removes the arms and fills in the hole of a chair. Notice how only optimizing the shape code or branch are unable to fit both edits. Optimizing the whole network is slow and causes unwanted changes in the instance.

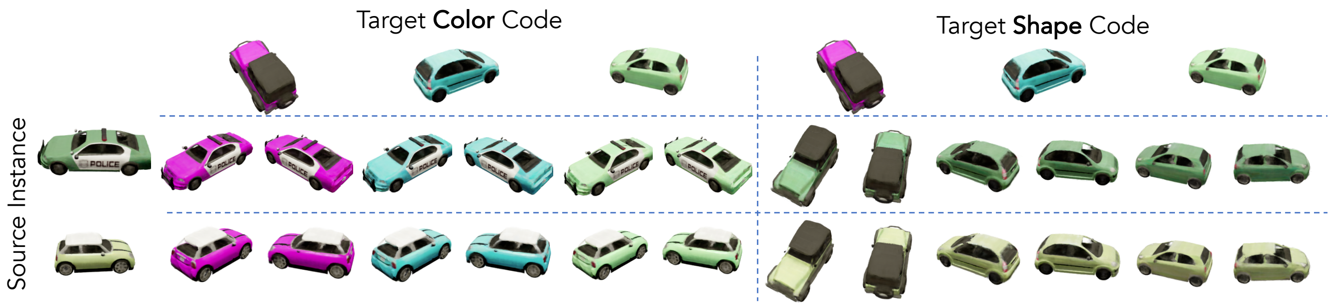

Figure 5: **Shape and color transfer results.** Our model transfers the shape and color from target instances to a given source instance. When a source's color code is swapped with a target's, the shape remains unchanged, and vice versa.

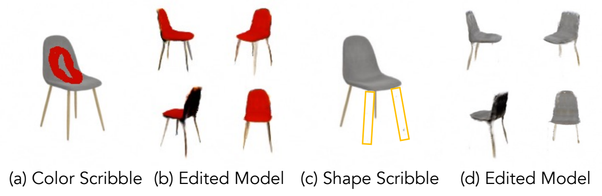

Figure 6: **Real image editing results.** Our method first finetunes a conditional radiance field to match a real still image input. Editing the resulting radiance field successfully changes the chair seat color to red and removes two of the chair's legs.

but can also cause visual artifacts. In practice, we find that a perfect pose estimation is not required. With the posed input image, we finetune the conditional radiance field by optimizing the standard NeRF photometric loss with respect to the image. When conducting this optimization, we first optimize the shape and color codes of the model, while keeping the MLP weights fixed, and then optimize all the parameters jointly. This optimization is more stable than the alternative of optimizing all parameters jointly from the start. Given the finetuned radiance field, we proceed with our editing methods to edit the shape and color of the instance. We show our results of editing a real photograph in Figure 6.

5. Discussion

We have introduced an approach for learning conditional radiance fields from a collection of 3D objects. Furthermore, we have shown how to perform intuitive editing operations using our learned disentangled representation. One limitation of our method is the interactivity of shape editing. Currently, it takes over a minute for a user to get feedback on their shape edit. The bulk of the editing operation computation is spent on rendering views, rather than editing itself. We are optimistic that NeRF rendering time improvements will help [46, 75]. Another limitation is our method fails to reconstruct novel object instances that are very different from other class instances. Despite these limitations, our approach opens up new avenues for exploring other advanced editing operations, such as relighting and changing an object's physical properties for animation.

Acknowledgments. Part of this work while SL was an intern at Adobe Research. We would like to thank William T. Freeman for helpful discussions.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[2] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *ACM SIGGRAPH*, 2004.

[3] Xiaobo An and Fabio Pellacini. Appprop: all-pairs appearance-space edit propagation. *ACM Transactions on Graphics (TOG)*, 27(3):40, 2008.

[4] Xiaobo An, Xin Tong, Jonathan D. Denning, and Fabio Pellacini. AppWarp: retargeting measured materials by appearance-space warping. *ACM Trans. Graph.*, 30(6):147, 2011.

[5] Mathieu Aubry, Daniel Maturana, Alexei A Efros, Bryan C Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3762–3769, 2014.

[6] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM SIGGRAPH*, 2007.

[7] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (TOG)*, 28(3):24, 2009.

[8] David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, and Antonio Torralba. Rewriting a deep generative model. In *European Conference on Computer Vision (ECCV)*, 2020.

[9] David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. Semantic photo manipulation with a generative image prior. *ACM SIGGRAPH*, 38(4), 2019.

[10] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[11] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *ACM SIGGRAPH*, page 425–432, New York, NY, USA, 2001. Association for Computing Machinery.

[12] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[13] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Trans. Graph.*, 32(6), Nov. 2013.

[14] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 11–20, New York, NY, USA, 1996. Association for Computing Machinery.

[16] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A. Efros, and Adrien Bousseau. 3d sketching using multi-view deep volumetric prediction. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1), July 2018.

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[18] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546, 2015.

[19] Yuki Endo, Satoshi Iizuka, Yoshihiro Kanamori, and Jun Mitani. Deepprop: Extracting deep features from a single image for edit propagation. *Computer Graphics Forum*, 35(2):189–201, 2016.

[20] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Styles Overbeck, Noah Snavely, and Richard Tucker. Deepview: High-quality view synthesis by learned gradient descent. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[21] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[22] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[23] Samuel W Hasinoff, Martyna Jóźwiak, Frédo Durand, and William T Freeman. Search-and-replace editing for personal photo collections. In *IEEE International Conference on Computational Photography (ICCP)*, 2010.

[24] James W. Hennessey, Wilmot Li, Bryan Russell, Eli Shechtman, and Niloy J. Mitra. Transferring image-based edits for multi-channel compositing. *ACM Transactions on Graphics*, 36(6), 2017.

[25] Daniel Reiter Horn and Billy Chen. Lightshop: Interactive light field manipulation and rendering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, page 121–128, 2007.

[26] Wendy Huther. *3DS Max Chair Modeling – Easy Beginner Tutorial*. https://www.youtube.com/watch?v=w_unzLDGj9U.

[27] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. *SIGGRAPH 99 Conference Proceedings, 109-126. ACM*, 99:409–416, 01 1999.

[28] iMeshh. *Blender 2.8 Pro Chair Modeling Guide! - iMeshh Furniture Tutorial*. https://www.youtube.com/watch?v=Q5XNaxa7jGg.

[29] Adrian Jarabo, Belen Masia, Adrien Bousseau, Fabio Pellacini, and Diego Gutierrez. How do people edit light fields? *ACM Trans. Graph.*, 33(4), July 2014.

[30] Adrian Jarabo, Belen Masia, and Diego Gutierrez. Efficient propagation of light field edits. In *Ibero-American Symposium in Computer Graphics (SIACG)*, pages 75–80, 2011.

[31] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision (IJCV)*, 38:199–218, 2000.

[34] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *ACM SIGGRAPH*, pages 689–694, 2004.

[35] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Transactions on Graphics (ToG)*, 23(3):303–308, 2004.

[36] Zhengqi Li, Wenqi Xian, Abe Davis, and Noah Snavely. Crowdsampling the plenoptic function. In *European Conference on Computer Vision (ECCV)*, 2020.

[37] Jerry Liu, Fisher Yu, and Thomas Funkhouser. Interactive 3d modeling with a generative adversarial network. In *2017 International Conference on 3D Vision (3DV)*, pages 126–134. IEEE, 2017.

[38] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2019.

[39] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. DIST: Rendering deep implicit signed distance function with differentiable sphere tracing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[40] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4):65:1–65:14, July 2019.

[41] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[42] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[43] Mateusz Michalkiewicz, Jhony K. Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[44] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019.

[45] 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 *European Conference on Computer Vision (ECCV)*, 2020.

[46] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards real-time rendering of neural radiance fields using depth oracle networks. *arXiv preprint arXiv:2103.03231*, 2021.

[47] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[48] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[49] Kyle Olszewski, Duygu Ceylan, Jun Xing, Jose Echevarria, Zhili Chen, Weikai Chen, and Hao Li. Intuitive, interactive beard and hair synthesis with generative models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[50] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[51] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. Photoshape: Photorealistic materials for large-scale shape collections. *ACM Trans. Graph.*, 37(6), Nov. 2018.

[52] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[53] Fabio Pellacini, Frank Battaglia, Keith Morley, and Adam Finkelstein. Lighting with paint. *ACM Transactions on Graphics*, 26(2), June 2007.

[54] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

[55] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 36(6), 2017.

[56] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH*, 2003.

[57] Tiziano Portenier, Qiyang Hu, Attila Szabó, Siavash Arjomand Bigdeli, Paolo Favaro, and Matthias Zwicker. Faceshop:

Deep sketch-based face image editing. *ACM Transactions on Graphics (TOG)*, 37(4):99:1–99:13, 2018.

[58] Thomas Porter and Tom Duff. Compositing digital images. *ACM SIGGRAPH*, 18(3):253–259, 1984.

[59] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

[60] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM SIGGRAPH*, 2004.

[61] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. PIFuHD: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[62] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material. In *Eurographics - State of the Art Reports*, 2014.

[63] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3d-aware image synthesis. In *Advances in Neural Information Processing Systems*, 2020.

[64] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision (IJCV)*, 35:151–173, 1999.

[65] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[66] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019.

[67] Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[68] Richard Szeliski and Polina Golland. Stereo matching with transparency and matting. *International Journal of Computer Vision*, 32:45–61, 1999.

[69] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[71] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing (TIP)*, 13(4):600–612, 2004.

[72] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *ACM SIGGRAPH*, USA, 2000.

[73] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. *arXiv preprint arXiv:2011.12950*, 2020.

[74] Kun Xu, Yong Li, Tao Ju, Shi-Min Hu, and Tian-Qiang Liu. Efficient affinity-based edit propagation using kd tree. *ACM Transactions on Graphics (TOG)*, 28(5):1–6, 2009.

[75] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024*, 2021.

[76] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[77] Kaan Yücer, Alec Jacobson, Alexander Hornung, and Olga Sorkine. Transfusive image manipulation. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.

[78] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[79] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. *ACM Transactions on Graphics (TOG)*, 9(4), 2017.

[80] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018.

[81] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision (ECCV)*, 2016.

[82] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Joshua B. Tenenbaum, and William T. Freeman. Visual object networks: Image generation with disentangled 3D representations. In *Advances in Neural Information Processing Systems*, 2018.