Communications in Information and Systems
Volume 23, Number 4, 447–475, 2023

PaletteNeRF: palette-based color editing for NeRFs*

Qiling Wu, Jianchao Tan, and Kun Xu†

Neural Radiance Field (NeRF) is a powerful tool to faithfully generate novel views for scenes with only sparse captured images. Despite its strong capability for representing 3D scenes and their appearance, its editing ability is very limited. In this paper, we propose a simple but effective extension of vanilla NeRF, named *PaletteNeRF*, to enable efficient color editing on NeRF-represented scenes. Motivated by recent palette-based image decomposition works, we approximate each pixel color as a sum of palette colors modulated by additive weights. Instead of predicting pixel colors as in vanilla NeRFs, our method predicts additive weights. The underlying NeRF backbone could also be replaced with more recent NeRF models such as KiloNeRF to achieve real-time editing. Experimental results demonstrate that our method achieves efficient, view-consistent, and artifact-free color editing on a wide range of NeRF-represented scenes.

1. Introduction

Neural Radiance Field (NeRF) [39] is a powerful tool for image-based modeling and rendering. With only a sparse set of captured images, it can faithfully generate rendering results from novel views. The core of NeRF is a neural volumetric representation of the scene using a multi-layer perceptron network. Due to its high effectiveness, it has attracted a wide range of attention from the community, with a bunch of follow-up works and applications since its introduction in 2020. However, due to the black-box nature of neural representations, all information, including geometries, materials, and light transports, are tightly baked into NeRFs, which are hard to interpret and edit.

Recently, some methods have been proposed to enable color editing of NeRFs [34, 70]. Given a few user-provided color scribbles, EditingNeRF [34]

*This work is supported by the National Natural Science Foundation of China (Project Number: 61932003).

†Corresponding author.

Figure 1: Color editing results of scene *chair* represented by Neural Radiance Field (NeRF). The origin and edited palettes are shown at the bottom-left corner. Users can modify the extracted palette to achieve intuitive, view-consistent, artifact-free editing of NeRFs.

propagates the user edits to the whole data to achieve color editing and shape modification. However, this work is too demanding for datasets, which require many instances from the same category for training, limiting its practical usage. CLIP-NeRF [70] uses embeddings of CLIP [53] to edit NeRFs. They finetune layers that influence color while freezing layers that influence density, to match the embedding of NeRF's output to that of the text editing prompt. However, it sometimes modifies undesired areas, leaving some artifacts in the results. PosterNeRF [68] gives an efficient way to extract palette from radiance fields, then utilizes posterization method [12] to achieve real-time color editing. Despite the real-time performance, the editing results have artifacts like color banding and leaking as side effects of posterization.

In this paper, we propose PaletteNeRF, an intuitive, view-consistent, and artifact-free palette-based color editing method for NeRFs. Inspired by palette-based image editing works, we approximate pixel colors as the sum of palette colors modulated by additive weights. Instead of predicting pixel colors as in vanilla NeRFs, we predict the high dimensional additive weights in our PaletteNeRF. The NeRF-represented scenes can be recolored by adjusting the palette colors without retraining or modifying the PaletteNeRF, as shown in Figure 1. The underlying NeRF backbone could also be replaced by more recent NeRF models, such as KiloNeRF [55], to achieve real-time editing.

2. Related work

2.1. Neural radiance field

Neural Radiance Field (NeRF) [39] utilizes radiance field to model a 3D scene implicitly. Specifically, they use MLPs to infer volumetric density and radiance for certain points and view directions of a scene and follow the paradigm of volumetric rendering to compute the image pixel colors. Given a sparse set of captured images, NeRF can generate high-quality results for novel views. However, this framework needs to train a separate MLP for every scene. All information about the scene, including geometries, materials, and light transports, is baked into the neural representation. Hence, the vanilla NeRF does not allow for changes in scene geometries, colors, and lighting.

Various follow-ups of NeRFs have been proposed to address this limitation, i.e., to deal with deformable scenes [52, 69, 48, 49], dynamic lighting [36], and scene composition [44, 79, 77, 75, 22, 46].

To improve NeRF's rendering speed, NSVF [33] and KiloNeRF [55] use empty space skipping and early ray termination. In addition, KiloNeRF divides the scene into small grids and uses a small and efficient MLP in each grid to achieve further speedup. FastNeRF [20] and PlenOctrees [76] utilize function factorization and cache the MLP results to speed up rendering. Some other methods aim at efficient training, e.g., Instant-NGP [40] use hierarchical hash encoding to replace the costly MLP.

As for editing, EditingNeRF [34] takes a set of objects with similar shapes and colors, such as cars from the Carla dataset [16], then associate each instance with a shape code and a color code, which are fed into Conditional Radiance Fields (CRFs). The users provide a few color scribbles to indicate color changes and shape addition/removal of an object, which are propagated to specific regions of that object. However, it requires a set of shapes in the same class with different geometries and colors. CLIP-NeRF [70] takes texts or exemplar images as edit prompts, which are fed into the multi-modal language model, i.e. CLIP, to obtain an embedding to serve as an offset for shape and color codes. The CRF processes the modified codes to output edited results. PosterNeRF [68] extracts the palette efficiently from radiance fields. They sample RGB color points and use volumetric visibility to remove outliers, after which, a palette is extracted from the remaining points. Each pixel is approximated as a linear blending of at most 2 palette colors. Color editing is performed by adjusting the palette color. However, undesired artifacts can be easily perceived in the editing

results. In contrast, our work only requires a single scene to enable color editing. Our method provides a relatively simple user interface and achieves intuitive, artifact-free, and view-consistent color editing. Readers could refer to more details in the surveys on NeRFs [13, 67].

2.2. Palette based editing

A palette is a concise representation of the color distributions of an image or video, which can be used to efficiently edit the image or video. Several works have studied the human perception of the palette. These works construct various data fitting models according to human preference and then regress a perceptual palette from input images [45, 31, 10, 18]. Other works [11, 41, 81, 1] adopt clustering methods (e.g., k-means) over pixel colors, and use the cluster centers as palette colors. Some other types of works [65, 63, 71, 14, 21, 27] utilize geometric methods to generate palettes for images. Specifically, Tan et al. [65, 63] compute an RGB space convex hull that includes all colors of an image, followed by an iterative simplification of the convex hull until the vertex number is reduced to a predefined number or the reconstruction error reaches a predefined threshold. The vertices of the simplified convex hull are considered palette colors. Recently, Du et al. [17] further extend this geometry-based method to time-lapse videos, generating time-varying palettes.

With the help of a palette, an image can be decomposed into multiple compositing layers, each of which contains spatially varied per-pixel opacities or per-pixel mixing weights. Several methods [62, 65] generate ordered translucent layers by assuming an alpha blending color composition mode. Other methods [2, 30, 63, 71, 61] generate order-independent layers by assuming additive color composition mode. Another concurrent work, also named PaletteNeRF [28], targets 3D scene palette-based editing, by using intrinsic decomposition [6] to separate the radiance of each 3D point into shading and reflectance, which is then decomposed into palette and mixing weights. The palette can be adjusted to achieve color editing.

3. Background

3.1. Neural Radiance Field (NeRF)

Given a sparse set of captured images of a scene, NeRF [39] can faithfully synthesize novel views for the scene. The core is a neural volumetric representation modeled with a multi-layer perceptron (MLP). The MLP f

predicts the RGB color and density observed at a 3D position $\mathbf{x}$ from a view direction $\mathbf{d}$:

$$(\mathbf{c}, \sigma) = f(\gamma(\mathbf{x}), \gamma(\mathbf{d})), \tag{1}$$

where $\gamma(\cdot)$ denotes a positional encoding function.

To render an image from a novel view, NeRF follows conventional ray marching techniques for volumetric rendering. For each image pixel, we first cast a ray $\mathbf{r}$ from the viewpoint through the pixel to the scene. After that, several 3D points are sampled along the ray, and each sampled point is fed into the MLP to obtain its color and density. Finally, the colors of all sampled points are blended to obtain the rendered pixel color.

A sparse set of captured images together with their camera parameters is sufficient for training a NeRF. A separate NeRF is required to be trained for each separate scene. During training, at each iteration, a batch of rays is randomly sampled from all (or a subset of) pixels. L2 differences between rendered pixel colors and ground truth pixel colors at given views are used as the loss function to be minimized.

3.2. Palette-based image decomposition

Given an RGB image $\boldsymbol{I}$, palette-based image decomposition techniques [65, 63, 71, 17] extract a small set of colors named *palette* from an image, such that the color $\mathbf{c}$ of each pixel $\mathbf{p}$ could be approximately represented as a linear combination of palette colors:

$$\mathbf{c} \approx \sum_{1 \le i \le K} w_i^{\mathbf{p}} \cdot \mathbf{v}_i, \tag{2}$$

where K denotes palette size (i.e., number of palette colors) which usually varies from 4 to 10, depending on the color complexity of the input image. $\mathbf{v}_i$ denotes the i-th palette color, and $w_i^{\mathbf{p}}$ denotes the additive weight of pixel $\mathbf{p}$ with respect to palette color $\mathbf{v}_i$. The pixel-wise additive weights are required to satisfy two properties. First, they are non-negative: $w_i^{\mathbf{p}} \ge 0$ $(1 \le i \le K)$; second, they sum to one: $\sum_{i=1}^{K} w_i^{\mathbf{p}} = 1$.

By denoting the palette as $\boldsymbol{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$ and the additive weight vector at each pixel $\mathbf{p}$ as $\mathbf{w}^{\mathbf{p}} = [w_1^{\mathbf{p}}, \ldots, w_K^{\mathbf{p}}]^T$, respectively, we could also rewrite the above formula in the vector-matrix form as:

$$\mathbf{c} \approx \boldsymbol{V} \cdot \mathbf{w}^{\mathbf{p}}. \tag{3}$$

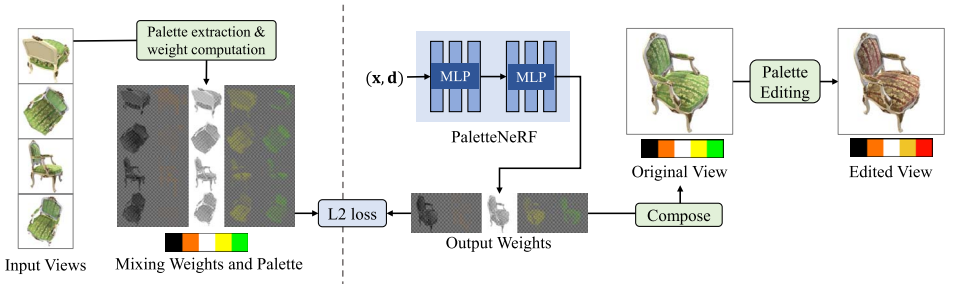

Figure 2: The pipeline of PaletteNeRF. We first preprocess input views to obtain the palette and mixing weights. The weights are used as NeRF's fitting target. After training NeRF, we obtain output weights for a novel view, and modify the palette color to edit that view.

By denoting the per-pixel additive weights ($\mathbf{w^p}$) as a weight image $\boldsymbol{W}$, i.e., having the same resolution as the input image $\boldsymbol{I}$ while the channel size is changed from 3 to K, we could approximate pixel colors of the input image by:

$$\boldsymbol{I} \approx \boldsymbol{V} \cdot \boldsymbol{W}. \tag{4}$$

There are various options of additive weights, including as-sparse-as-possible weights [65], RGBXY barycentric weights [63], and mean value coordinates (MVC) [71, 25, 19]. Since MVC weights can be efficiently computed and have been demonstrated to be smooth, sparse, and effective [71, 17], we use MVC weights in our paper.

Once the palette and the additive weight images are extracted, by simply adjusting the colors in the palette, the images can be instantly recolored through weighted linear interpolations from the palette colors (Eq. (2)). This offers a more convenient editing interface than stroke-based methods, since those methods additionally user interaction steps, i.e., drawing strokes on the image.

4. Our method

In this section, we introduce PaletteNeRF, a concise modification to NeRF that makes scenes editable. We will first introduce our pipeline (Sec. 4.1), then discuss the details of our network structure and training strategy (Sec. 4.2).

4.1. Overview and pipeline

Our goal is to provide an intuitive, efficient, and artifact-free color editing tool for NeRFs. Motivated by existing palette-based image decomposition works [65, 63, 71, 17], we find that if we approximate additive weights rather than pixel colors, the colors of the scene represented by NeRFs could be naturally edited.

Specifically, instead of using NeRFs to model pixel colors which are essentially 3D (RGB) signals, we model higher-dimensional signals, e.g., the pixel-wise additive weights. Thanks to the high capacities of MLPs, the K-dimensional pixel-wise additive weights can also be well approximated. After that, we could intuitively edit the colors of the NeRF-represented scene by adjusting the palette colors. We refer to our modified NeRF as *PaletteNeRF*.

Our overall pipeline is shown in Figure 2. The steps for utilizing PaletteNeRF include:

- **Data Preparation.** Given a set of sparse captured RGB images of a scene, we convert them to the same number of K-channel additive weight images and a shared palette $\boldsymbol{V}$ with K colors. This is done by stitching all images into a big image, and then directly applying the method in [71]. The pixel-wise additive weights are obtained using mean value coordinates.
- **Modified MLP.** We feed the K-channel additive weight images to PaletteNeRF. Different from the vanilla NeRF which predicts 3-channel RGB colors, our PaletteNeRF predicts a K-channel vector $\mathbf{w}$ indicating additive weights, i.e., its formulation is changed from $f(\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ to $f(\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{w}, \sigma)$.
- **Novel View Rendering.** To obtain the additive weight image $\boldsymbol{W}_{\mathbf{o}}$ from a novel view, we apply the same process as done in vanilla NeRF, i.e., through ray marching and blending of K-channel values. The RGB image from this novel view can be simply reconstructed through Eq. (2) as:

$$\boldsymbol{I}_{\mathbf{o}} = \boldsymbol{V} \cdot \boldsymbol{W}_{\mathbf{o}}. \tag{5}$$

- **Color Editing.** Then, we can freely adjust the palette colors $\boldsymbol{V}$ to achieve recoloring of the scene. During the color editing process, our PaletteNeRF, which predicts additive weights, is not modified at all. This indicates that our editing process is decoupled with the PaletteNeRF structure. Such decoupling has two advantages. First, the used NeRF backbone could be replaced with any newer, faster variants of

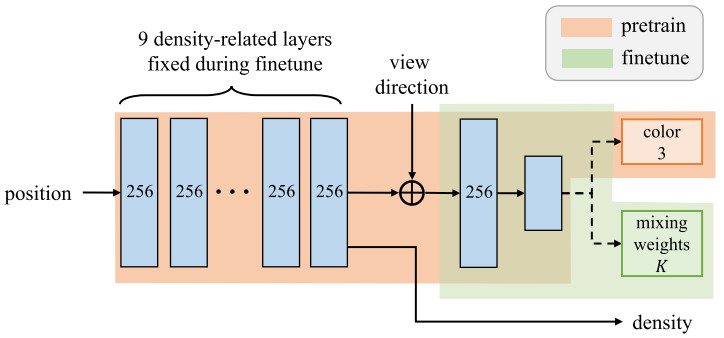

Figure 3: The network structure of our PaletteNeRF. The number inside each block denotes the dimension of the layer. The top branch of the network predicts 3D colors which are used in the pretrain stage, while the bottom branch predicts K-channel additive mixing weights which are used in the finetune stage.

NeRFs. In our experiments, we also demonstrate our method with KiloNeRF and Instant-NGP backbone [55, 40], which allows color editing in real-time. Second, the consistency between different views is also naturally preserved, leading to artifact-free color editing with high fidelity.

4.2. Network structure and training

As shown in Figure 3, our network structure is almost the same as the vanilla NeRF, except for the last layer. We use a $128 \times K$ fully connected (FC) layer, instead of a 128×3 FC layer, as the last layer to output a K-channel vector.

For training a PaletteNeRF, we need to change the loss function from computing L2 differences between rendered and GT RGB images to computing L2 differences between rendered (predicted) and the GT K-channel additive weight images. A straightforward training process would be directly using the same process as done for training a vanilla NeRF, only with the above change on the loss function. While such a straightforward training strategy could work, we find that its reconstruction quality is relatively low.

Based on the fact that our PaletteNeRF and the vanilla NeRF share a majority part of network structures, we propose a two-stage *pretrain* +

finetune training strategy. Specifically, we first pretrain a vanilla NeRF using RGB input images. After that, we fix the parameters of the density-related layers, i.e. the parameters in the first 9 layers are kept unchanged, and finetune the parameters of the last 2 color-related layers by minimizing the L2 differences between the rendered and the GT K-channel additive weight images. Experiments in Sec. 6.1 show that such a two-stage training strategy leads to better reconstruction accuracy.

5. Method extensions

Our method can be easily extended in several ways. In the following, first, we show that the backbone of PaletteNeRF can be replaced by KiloNeRF and Instant-NGP [55, 40] to achieve real-time recoloring (Sec. 5.1). Second, we extend the palette and mixing weights of PaletteNeRF to capture indirect lighting in synthetic scenes (Sec. 5.2).

Besides, we have shown that applications in palette-based editing can be directly incorporated into our framework, including color harmonization and color transfer.

5.1. Real-time color editing

As mentioned earlier, our editing process is decoupled with the underlying NeRF backbone. The vanilla NeRF backbone can be replaced by any newer, faster variants of NeRFs as long as they are still volumetric representations. To demonstrate this ability, we also implement our PaletteNeRF with KiloNeRF and Instant-NGP backbones [55, 40], which allows color editing in real-time. Similar to NeRF, we modify the output dimension of the network used in each work from 3 to K. The modified network produces frame rates of 14.3 and 21.0 FPS for KiloNeRF and Instant-NGP respectively, which are usable for real-time editing. The editing results of each backbone are shown in Figure 4. The results with KiloNeRF and Instant-NGP backbones are visually indistinguishable from those with vanilla NeRF backbones.

We also make a GUI to enable real-time novel view synthesis and color editing. Our GUI is based on the framework DearPyGUI [23]. Figure 5 shows an example of our GUI. A video demo is also provided in the supplementary video.

5.2. Second-order weights for synthetic scenes

NeRF scenes can be classified into captured scenes and synthetic scenes. Captured scenes refer to those scenes where only captured images from spe-

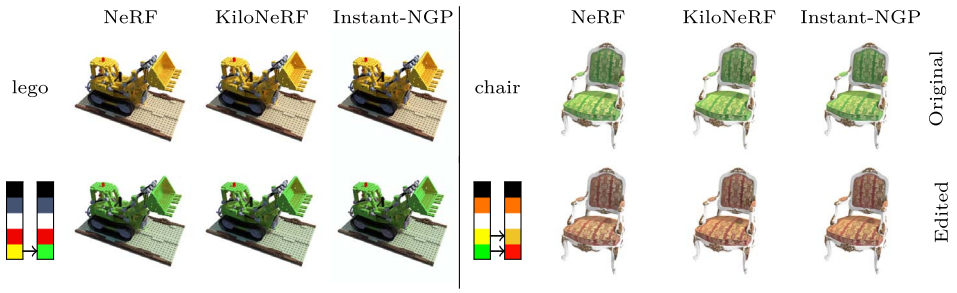

Figure 4: Editing results of using vanilla NeRF, KiloNeRF and Instant-NGP as backbones, respectively. All methods can fit the multi dimensional weights well, and achieve editing results with visually indistinguishable quality.

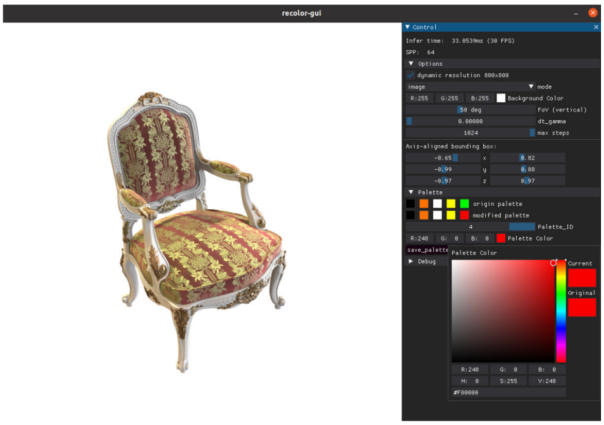

Figure 5: Recoloring scene *chair* with our GUI. User can select and modify palette to achieve real-time recoloring.

cific views are available. While the images of synthetic scenes are generated using renderers with given 3D geometries, materials, and lighting information.

For synthetic NeRF scenes, we can further extend our method to provide a more semantically meaningful palette and better controls on recoloring, i.e., supporting higher-order effects (indirect lighting) besides additive mixing. Let's imagine a synthetic 3D scene with static geometry and static lighting, but with some editable materials, whose diffuse colors $\mathbf{v}_i$ are allowed to adjust. Considering indirect illuminations up to 2 bounces, the rendered

color $\mathbf{c}$ of a pixel $\mathbf{p}$ can be computed as follows:

$$\mathbf{c} = w_0^{\mathbf{p}} + \sum_{1 \le i \le K} w_i^{\mathbf{p}} \cdot \mathbf{v}_i + \sum_{1 \le i \le j \le K} w_{i,j}^{\mathbf{p}} \cdot (\mathbf{v}_i \odot \mathbf{v}_j), \tag{6}$$

where $\odot$ denotes channel-wise multiplication, $w_0^{\mathbf{p}}$ denotes contributions from light paths not intersecting with any editable materials. $w_i^{\mathbf{p}}$ and $w_{i,j}^{\mathbf{p}}$ denote contributions from light paths intersecting one and two times of editable materials, respectively. All these weights are essentially light transports and can be directly computed using a path tracing renderer such as PBRT [50].

The equation can be written in the vectorized form:

$$\mathbf{c} = \boldsymbol{V} \cdot \mathbf{w}^{\mathbf{p}}, \tag{7}$$

where:

$$\begin{aligned} \boldsymbol{V} &= [\mathbf{1}, \mathbf{v}_1, \ldots, \mathbf{v}_K, \mathbf{v}_{1,1}, \ldots, \mathbf{v}_{K,K}], \\ \mathbf{w}^{\mathbf{p}} &= [w_0^{\mathbf{p}}, w_1^{\mathbf{p}}, \ldots, w_K^{\mathbf{p}}, w_{1,1}^{\mathbf{p}}, \ldots, w_{K,K}^{\mathbf{p}}]^T. \end{aligned} \tag{8}$$

The above modification can be directly supported by our PaletteNeRF, by only modifying the data preparation step at the beginning and the color editing step at the end, as described in Sec. 4.1. Notice that a palette size of K would produce a second-order weight vector $\boldsymbol{V}$ of length $(K+1)(K+2)/2$.

In the data preparation step, we no longer use existing palette-based image decomposition methods to extract palette and additive weights. Instead, we ask users to manually specify several materials he or she want to adjust and use the diffuse colors of specified materials as palette colors. Then, we compute the second-order weights in Eq. (8) directly using PBRT [50]. In the color editing step, we use Eq. (6) instead of Eq. (2) to compute the reconstructed colors.

We have tested two synthetic scenes: *Cornell box* and *breakfast*. The editing results are shown in Figure 6. Notice the visually plausible indirect illumination effects after recoloring, e.g., color bleeding effects in the ceiling and reflection of the sphere on the cylinder surface in scene *Cornell box*, and color bleeding effects including pink tint of the floor and green tint of the desk in scene *breakfast*.

5.3. Color harmonization and transfer

Our work bridges between NeRF rendering and palette-based editing. Therefore, many applications in either domain can be directly Incorporated into

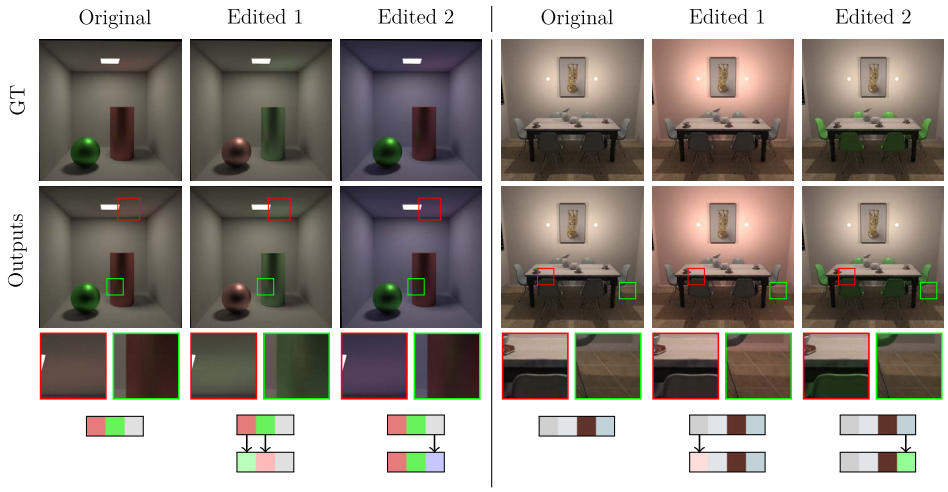

Figure 6: Rendered scenes (*Cornell box*, *breakfast*) and editing results. For each scene, the 4 rows are: ground truth, PaletteNeRF output, amplified areas, color of tunable materials. Colored rectangles amplify area which have interreflection effects. Palette colors are gamma-corrected.

our framework. In this section, we demonstrate two applications of palette-based editing in [64], including color harmonization and color transfer.

These two applications are based on several harmonic color templates in the Hue space. We use 7 harmonic templates designed for palette colors in [64]. Each harmonic template has at least 1 parameter α, which describes the angle of rotation in the hue of the axis. The harmonization is performed in HCL color space (Hue, Chroma, Luminance). We utilize the palette $\boldsymbol{V}$ extracted in the data preparation step of PaletteNeRF. For template $T_m(\alpha)$, we find the closest axis to each color in $\boldsymbol{V}$, by measuring the distance of hue in the HCL color space. The distance is weighted and summed to compute a template fitting cost, which is then minimized to find the best α for each template. Once the template is chosen, each palette color in $\boldsymbol{V}$ is projected onto the nearest axis of the template, forming the harmonized palette $\boldsymbol{V}'$. Then, as described in the main paper, $\boldsymbol{V}'$ is used to recolor, i.e. harmonize the whole scene. Figure 7 shows harmonized results of the scene *orchids*.

Based on harmonic templates, we can perform the color transfer application. Given an input image I and a reference image R, we extract their palette P_I, P_R, and compute optimal templates $T_I(\alpha_I), T_R(\alpha_R)$. Tan et al. [64] proposed two template-based color transfer methods – template

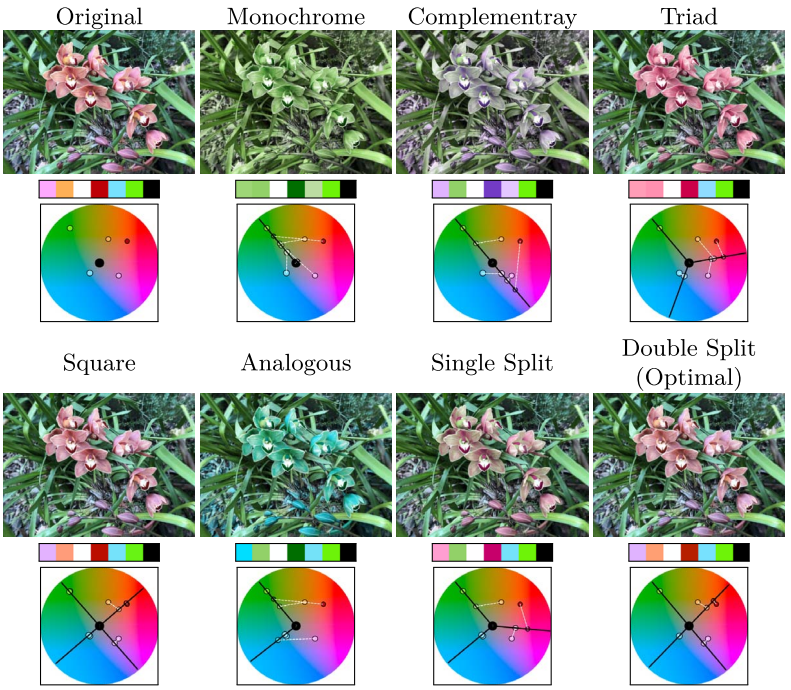

Figure 7: Comparison of the same scene *orchids* harmonized by 7 different templates.

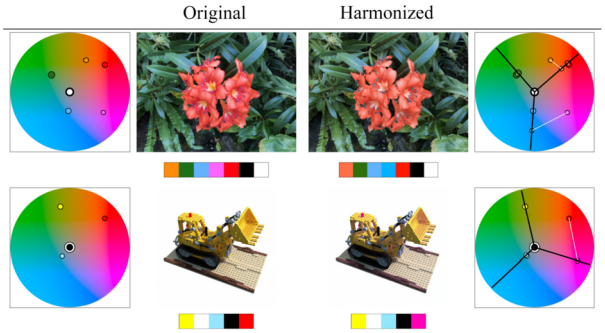

Figure 8: Examples of color harmonization applied to NeRF scenes.

alignment transfer and template fitting transfer. For the first one, we compute the main color axis of $T_I(\alpha_I), T_R(\alpha_R)$. The main axis represents the dominant color distribution of an image. Then we rotate $T_I(\alpha_I)$ to align the

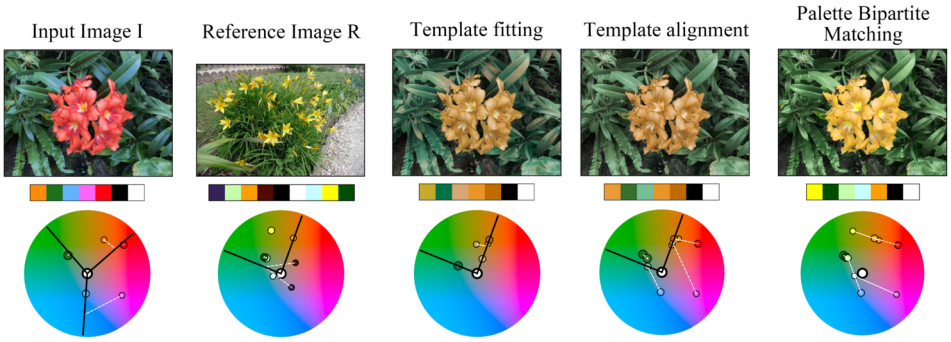

Figure 9: Results of two different template-based color transfer methods and palette bipartite matching based color transfer.

main axis with $T_R(\alpha_R)$. Finally, we perform harmonization on image I using $T_R(\alpha_R)$. The second method aims at better preserving the original colors. In this case, we harmonize palette P_I by fitting it to template $T_R(\alpha_R)$ without any rotation. Figure 9 demonstrates each method. Our palette-based NeRF editing is general to various down-streaming palette editing, we also show the palette bipartite matching-based color transfer application, which minimizes the Euclidean distance in HCL space between two palettes while performing their bipartite matching. The result is also shown in Figure 9.

6. Experiments

We conduct our experiments on a PC with an NVIDIA RTX 3080Ti GPU, a Ryzen 5900X CPU, and 64GB of RAM. For each scene, we train Palette-NeRF for 200k iterations, which takes around 5 hours.

6.1. Evaluation

6.1.1. Training strategy As mentioned in Sec. 4.2, we have proposed a two-stage (pretrain + finetune) training strategy. To validate the effectiveness of the two-stage strategy, we compare it against the traditional one. For a fair comparison, we set the number of iterations to be the same for both training strategies. Specifically, the traditional strategy uses 200k iterations, while our two-stage strategy sets iteration steps as 180k (pre-train) + 20k (finetune) for captured scenes, and 140k (pre-train) + 60k (finetune) for synthetic scenes. We have tested 8 scenes, including 6 captured scenes

Table 1: Quantitative comparison between the traditional strategy and our two-stage (pretrain + finetune) training strategy. "Additive Weights" columns report average scores of 6 captured scenes (*lego, chair, flower, orchids, vasedeck, valley*). "Second Order Weights" columns report average scores of 2 synthetic scenes (*Cornell box, breakfast*)

training method	Additive Weights			Second Order Weights		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
traditional	25.21	0.832	0.129	8.58	0.084	0.363
ours	**26.71**	**0.833**	**0.095**	**40.11**	**0.976**	**0.024**

Table 2: Ablation studies on the dimension of the last layer of PaletteNeRF (shown in the first 3 rows) evaluated on scenes *vasedeck* and *lego*. The last row shows scores of the vanilla NeRF with 128 dimension for comparison

dimension	vasedeck			lego		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
128	24.26	0.693	0.172	30.30	0.941	0.031
256	24.73	**0.722**	0.154	30.53	0.944	**0.030**
512	**24.73**	0.720	**0.156**	**30.66**	0.945	0.029
NeRF(128)	24.71	0.715	0.163	30.60	**0.946**	**0.030**

and 2 synthetic scenes, and report the average reconstruction scores for both strategies in Table 1.

From the results, we could find that our two-stage training strategy improves the reconstruction quality by a large gap, especially on the synthetic scenes with second-order weights. This is possibly due to the relatively large number of channels making it harder to be directly trained, while a pre-train step helps provide a good initialization.

6.1.2. Dimension of the last layer We also evaluate different choices of the input dimension of the last layer. Table 2 shows the reconstruction scores on two scenes when the dimension of the last layer is set to 128, 256, and 512, respectively. Increasing the dimension of the last layer could slightly increase reconstruction quality, at the cost of longer rendering time. However, to make a good trade-off between quality and speed, we set it as 128 in our experiments.

6.2. Comparisons

6.2.1. Comparisons with other palette-based models Figure 10 compares our edited results with a concurrent work – PosterNeRF [68] on

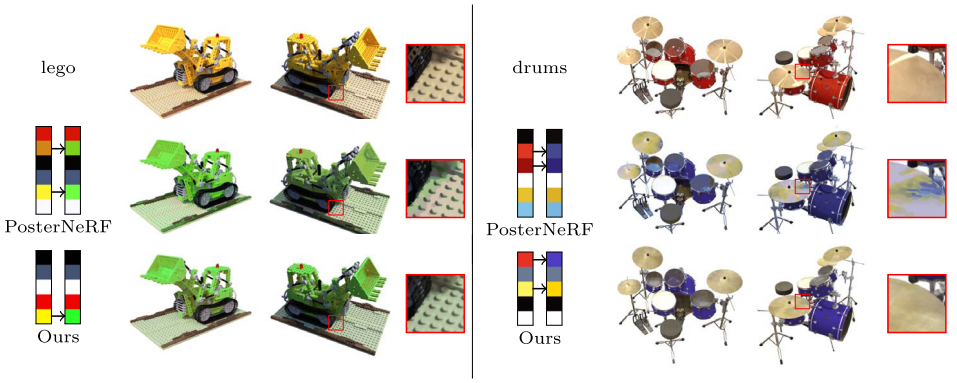

Figure 10: Comparison with PosterNeRF [68]. For each example, from top to bottom, we show the GT original view from dataset, recolored results generated by PosterNeRF and by our PaletteNeRF, respectively. Palettes on the left illustrate the editing operations used by each method.

two scenes *lego* and *drums*. PosterNeRF also employs a palette-based editing interface. However, it could easily generate visible artifacts, i.e., producing inconsistent recoloring results. Please see the close-up images in Figure 10: PosterNeRF produces unnatural recoloring on the ground near the excavator in scene *lego* and generates unsmooth recoloring on the cymbal in scene *drums*. In contrast, our recoloring results are much better without artifacts.

The reason why PosterNeRF generates artifacts lies in the way it uses palettes and computes mixing weights. PosterNeRF will choose at most 2 palette colors, resulting in at most 2 non-zero values of weight $\mathbf{w^P}$ in Eq. (3). Thus, some colors cannot be reproduced precisely. Furthermore, the weights are also quantized with large step sizes, which leads to color banding artifacts. In contrast, our method approximates each pixel with a linear blending of an arbitrary number of palettes, and the weight of linear blending is accurate, thus producing smooth results.

In Figure 11, we compare with two state of art palette-based recoloring methods, including a concurrent work (Kuang et al. [28]) and a video recoloring method (Du et al. [17]). In scene *kitchen*, our results are comparable to Kuang et al., while Du et al. unexpectedly modified leaves to become reddish. We show two views of scene *bonsai* to demonstrate 3D consistency. Du et al. perform well in one view, but make the whole image become purple in another. In contrast, our model produces view-consistent results.

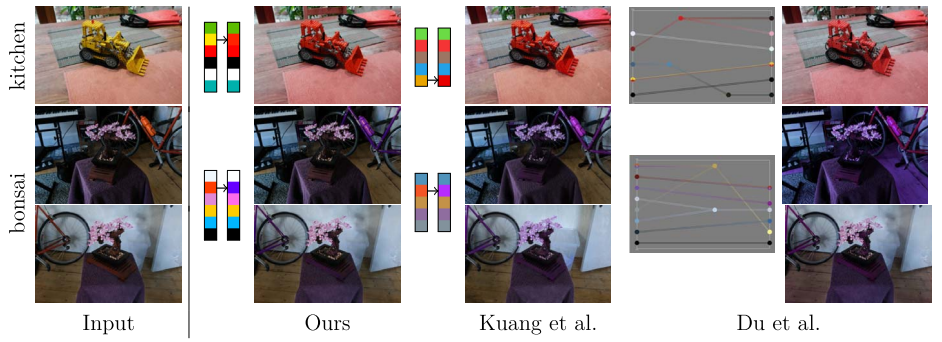

Figure 11: Comparison with Kuang et al. [28] and Du et al. [17] on dataset MipNeRF360 [5]. For each recolored images, we show the corresponding edited palette on the left side. We change the color of lego in scene *kitchen*, and bicycle in scene *bonsai*.

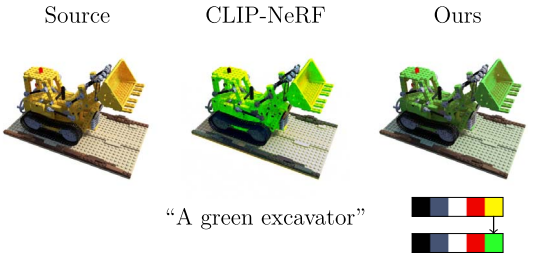

Figure 12: Comparison with CLIP-NeRF [70], which takes "A green excavator" as text edit prompt to edit the scene. We change one of the palette colors from yellow to green.

6.2.2. Comparison with CLIP-NeRF In Figure 12, we further compare our color editing results with CLIP-NeRF [70] on scene *lego*. We can see that CLIP-NeRF tends to blur the details of the excavator, add noise to the ground, and modify undesired regions such as red lights and grey-blue shafts. In contrast, our method has better image quality and provides better controllability for color editing.

We do not compare with EditingNeRF [34] since it is hard to make a fair comparison. Our method, like vanilla NeRF, could be trained through a single scene and used to recolor the scene. In contrast, EditingNeRF requires a scene dataset with many instances from the same category to enable editing, disabling its ability to edit a single NeRF scene (such as the standard ones like *lego* and *drums*).

Table 3: Quantitative comparison of PaletteNeRF to the vanilla NeRF on 8 scenes. We report PSNR/SSIM (higher is better) and LPIPS (lower is better). PaletteNeRF is able to achieve similar performance compared to the vanilla NeRF

	lego			chair		
method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	31.15	0.952	0.025	31.32	0.956	0.041
PaletteNeRF	30.14	0.941	0.030	30.72	0.950	0.045
	flower			orchids		
method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	28.15	0.887	0.053	21.33	0.750	0.109
PaletteNeRF	28.17	0.881	0.059	21.52	0.759	0.108
	vasedeck			valley		
method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	24.71	0.715	0.163	27.10	0.760	0.137
PaletteNeRF	24.66	0.721	0.152	26.64	0.745	0.159
	Cornell box			breakfast		
method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	44.15	0.993	0.007	39.95	0.971	0.03
PaletteNeRF	41.34	0.985	0.013	38.88	0.967	0.035

6.3. Results

In Figures 1 and 14, we show the recoloring results on several scenes using our PaletteNeRF. For each example, the recolored results on multiple novel views are provided. PaletteNeRF can produce consistent recoloring results across multiple views.

In Table 3, we also provide the quantitative reconstruction scores (including PSNR, SSIM, and LPIPS) of PaletteNeRF for all input scenes. The reconstruction score of the vanilla NeRF is also provided. Figure 13 gives visual comparisons between the reconstructed results of our PaletteNeRF and the vanilla NeRF. It could be found that the quality of PaletteNeRF is comparable to the vanilla NeRF from both aspects of quantitative measures and visual results. Without reducing visual qualities, our method enables the ability of color editing for NeRFs. Readers can view our supplementary video for more color editing results, including results of different backbones, and a GUI demo mentioned above.

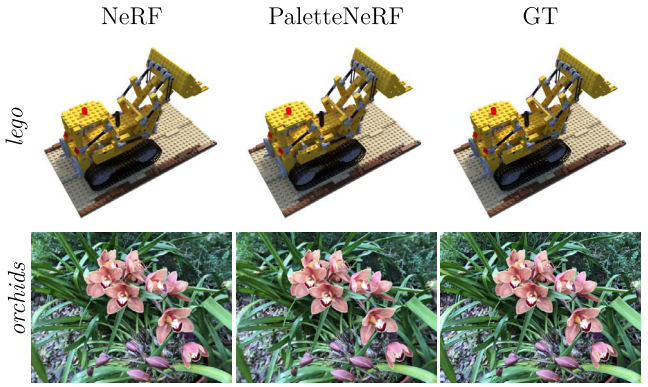

Figure 13: Qualitative comparison of PaletteNeRF to the vanilla NeRF. For each scene, from left to right, we show novel views synthesized by the vanilla NeRF and by our method, respectively, and the ground truth.

7. Conclusion

In this paper, we present PaletteNeRF, which unifies the palette-based image decomposition methods and NeRFs to enable color editing of NeRF-represented scenes. Our method is intuitive, efficient, view-consistent, and artifact-free. The users could recolor the scene by adjusting palette colors and previewing the recolored results from any novel views. Moreover, the editing process is decoupled from the network structure, which means the backbone of PaletteNeRF and the data preparation step can be replaced with more advanced follow-ups.

Nevertheless, several limitations still exist in our method. For example, we only allow global color editing, hence, if a scene contains two red apples, it is not allowed to only change the color of one apple. Local color editing is an valuable topic for future work. Furthermore, geometry editing is not supported. It is worthwhile to investigate ways to edit geometries.

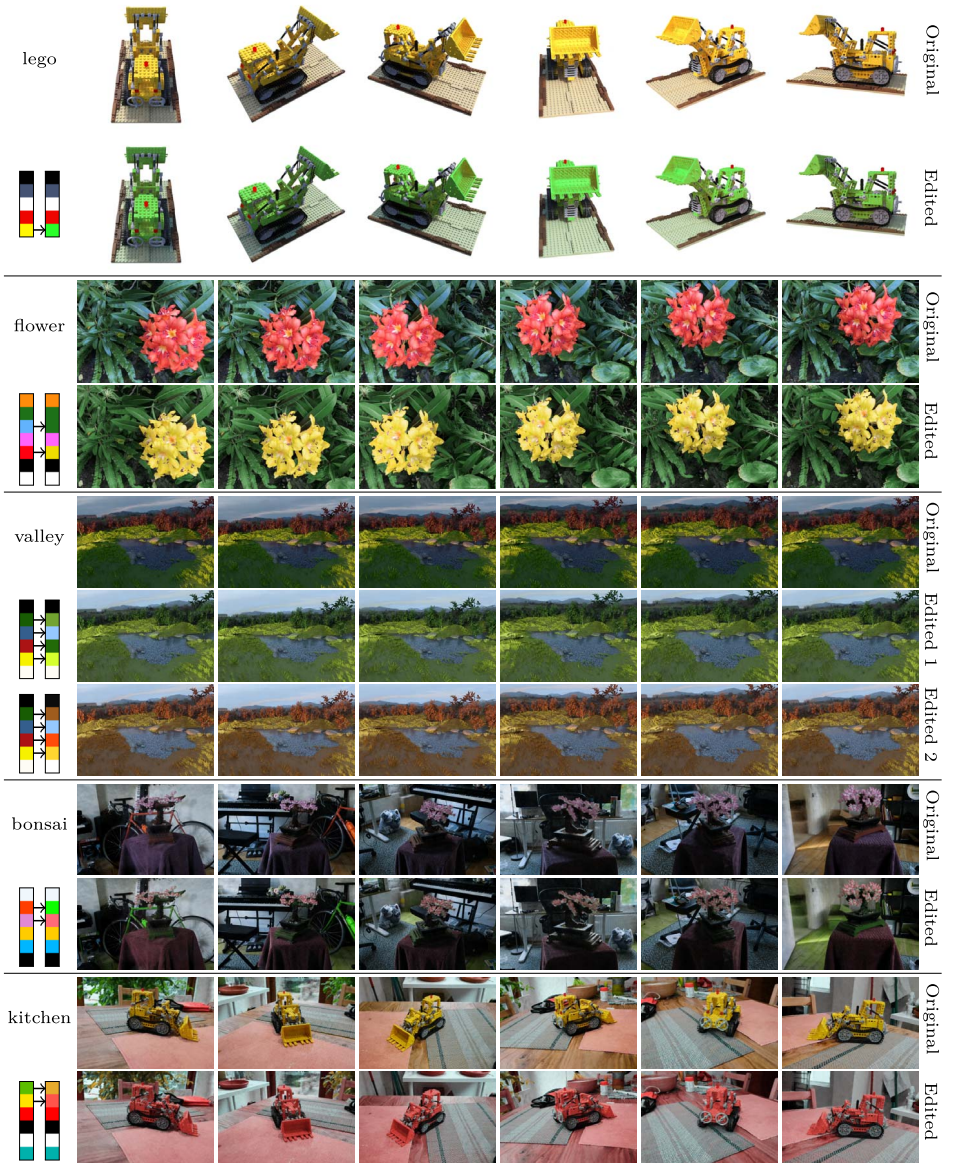

Figure 14: Color editing results of our method. In each 2 rows, the top one is NeRF's output, the bottom one is editing results of PaletteNeRF. Scenes are from [39, 72, 5]. We use vanilla NeRF as backbone for the first 3 scenes, and Instant-NGP for the last 2.

References

[1] Elad Aharoni-Mack, Yakov Shambik, and Dani Lischinski. Pigment-based recoloring of watercolor paintings. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, pages 1–11, 2017.

[2] Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. Unmixing-based soft color segmentation for image manipulation. *ACM Transactions on Graphics (TOG)*, 36(2):1–19, 2017.

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

[4] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[6] Harry Barrow, J. Tenenbaum, A. Hanson, and E. Riseman. Recovering intrinsic scene characteristics. *Comput. Vis. Syst.*, 2(3-26):2, 1978.

[7] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. Neural reflectance fields for appearance acquisition. *arXiv preprint arXiv:2008.03824*, 2020.

[8] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/.

[9] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik Lensch. NeRD: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021.

[10] Ying Cao, Antoni B. Chan, and Rynson WH Lau. Mining probabilistic color palettes for summarizing color use in artwork collections. In *SIGGRAPH Asia 2017 Symposium on Visualization*, pages 1–8, 2017.

[11] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph.*, 34(4):139–1, 2015.

[12] Cheng-Kang Ted Chao, Karan Singh, and Yotam Gingold. PosterChild: blend-aware artistic posterization. In *Computer Graphics Forum*, volume 40, pages 87–99. Wiley Online Library, 2021. Issue: 4.

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

[14] Baptiste Delos, Nicolas Mellado, David Vanderhaeghe, and Remi Cozot. RGB point cloud manipulation with triangular structures for artistic image recoloring. *arXiv preprint arXiv:1912.04583*, 2019.

[15] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: fewer views and faster training for free. *arXiv:2107.02791 [cs]*, July 2021.

[16] 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.

[17] Zheng-Jun Du, Kai-Xiang Lei, Kun Xu, Jianchao Tan, and Yotam Gingold. Video recoloring via spatial-temporal geometric palettes. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

[18] Zunlei Feng, Wolong Yuan, Chunli Fu, Jie Lei, and Mingli Song. Finding intrinsic color themes in images with human visual perception. *Neurocomputing*, 273:395–402, 2018.

[19] Michael S. Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005. MR2169052

[20] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-Fidelity Neural Rendering at 200FPS, pages 14346–14355, 2021.

[21] Mairéad Grogan and Aljosa Smolic. Image decomposition using geometric region colour unmixing. In *European Conference on Visual Media Production*, pages 1–10, 2020.

[22] Michelle Guo, Alireza Fathi, Jiajun Wu, and Thomas Funkhouser. Object-centric neural scene rendering. *arXiv preprint arXiv:2012.08503*, December 2020.

[23] Jonathan Hoffstadt and Preston Cothren. https://github.com/hoffstadt/dearpygui, 2021.

[24] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, January 1991.

[25] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Siggraph 2005 Papers*, pages 561–566. 2005.

[26] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 143–150, 1986.

[27] Suzi Kim and Sunghee Choi. Automatic color scheme extraction from movies. In *Proceedings of the 2020 International Conference on Multimedia Retrieval*, pages 154–163, 2020.

[28] Zhengfei Kuang, Fujun Luan, Sai Bi, Zhixin Shu, Gordon Wetzstein, and Kalyan Sunkavalli. PaletteNeRF: palette-based appearance editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20691–20700, 2023.

[29] Zhengfei Kuang, Kyle Olszewski, Menglei Chai, Zeng Huang, Panos Achlioptas, and Sergey Tulyakov. NeROIC: neural rendering of objects from online image collections. *arXiv:2201.02533 [cs]*, January 2022.

[30] Sharon Lin, Matthew Fisher, Angela Dai, and Pat Hanrahan. LayerBuilder: Layer decomposition for interactive image and video color editing. *arXiv preprint arXiv:1701.03754*, 2017.

[31] Sharon Lin and Pat Hanrahan. Modeling how people extract color themes from images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3101–3110, 2013.

[32] David B. Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565, 2021.

[33] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, January 2021. arXiv:2007.11571.

[34] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In

Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 5773–5783, 2021.

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

[36] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, January 2021. arXiv:2008.02268.

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

[38] 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)*, 38(4):1–14, 2019.

[39] 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*, pages 405–421. Springer, March 2020.

[40] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv:2201.05989 [cs]*, January 2022.

[41] Rang MH Nguyen, Brian Price, Scott Cohen, and Michael S. Brown. Group-theme recoloring for multi-image color consistency. In *Computer Graphics Forum*, volume 36, pages 83–92. Wiley Online Library, 2017. Issue: 7.

[42] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7588–7597, 2019.

[43] Thu H. Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. *Advances in Neural Information Processing Systems*, 31, 2018.

[44] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*, page 12, 2021.

[45] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Color compatibility from large datasets. In *ACM SIGGRAPH 2011 Papers*, pages 1–12. 2011.

[46] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2856–2865, 2021.

[47] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[48] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021.

[49] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. HyperNeRF: a higher-dimensional representation for topologically varying neural radiance fields. *arXiv:2106.13228 [cs]*, June 2021.

[50] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[51] Francesco Pittaluga, Sanjeev J. Koppal, Sing Bing Kang, and Sudipta N. Sinha. Revealing scenes by inverting structure from motion reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 145–154, 2019.

[52] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

[53] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela

Mishkin, and Jack Clark. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[54] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14161, 2021.

[55] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *ICCV 2021*, 2021.

[56] Christian Richardt, Jorge Lopez-Moreno, Adrien Bousseau, Maneesh Agrawala, and George Drettakis. Vectorising bitmaps into semi-transparent gradient layers. In *Computer Graphics Forum*, volume 33, pages 11–19. Wiley Online Library, 2014. Issue: 4.

[57] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019.

[58] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021.

[59] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: super-fast convergence for radiance fields reconstruction. *arXiv preprint arXiv:2111.11215*, 2021.

[60] Xin Sun, Kun Zhou, Yanyun Chen, Stephen Lin, Jiaoying Shi, and Baining Guo. Interactive relighting with dynamic BRDFs. *ACM Transactions on Graphics*, 26(3):27, July 2007.

[61] Jianchao Tan, Stephen DiVerdi, Jingwan Lu, and Yotam Gingold. Pigmento: Pigment-based image analysis and editing. *Transactions on Visualization and Computer Graphics (TVCG)*, 25(9), 2019.

[62] Jianchao Tan, Marek Dvorožňák, Daniel Sykora, and Yotam Gingold. Decomposing time-lapse paintings into layers. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.

[63] Jianchao Tan, Jose Echevarria, and Yotam Gingold. Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry. *ACM Transactions on Graphics*, 37(6):1–10, January 2019.

[64] Jianchao Tan, Jose I. Echevarria, and Yotam I. Gingold. Palette-based image decomposition, harmonization, and color transfer. *CoRR*, arXiv:1804.01225, 2018.

[65] Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. Decomposing images into layers via RGB-space geometry. *ACM Transactions on Graphics*, 36(1):1–14, February 2017.

[66] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, and Matthias Nießner. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020. Issue: 2.

[67] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. Advances in neural rendering. *arXiv:2111.05849 [cs]*, March 2022.

[68] Kenji Tojo and Nobuyuki Umetani. Recolorable posterization of volumetric radiance fields using visibility-weighted palette extraction. In *Computer Graphics Forum*, volume 41, pages 149–160. Wiley Online Library, 2022. Issue: 4.

[69] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhofer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12959–12970, 2021.

[70] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. CLIP-NeRF: text-and-image driven manipulation of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, March 2022. arXiv:2112.05139 [cs].

[71] Yili Wang, Yifan Liu, and Kun Xu. An improved geometric approach for palette-based image decomposition and recoloring. In *Computer Graphics Forum*, volume 38, pages 11–22. Wiley Online Library, 2019. Issue: 7.

[72] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF–: neural radiance fields without known camera parameters. *arXiv:2102.07064 [cs]*, April 2022.

[73] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: point-based neural radiance fields. *arXiv:2201.08845 [cs]*, January 2022.

[74] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep view synthesis from sparse photometric images. *ACM Transactions on Graphics (ToG)*, 38(4):1–13, 2019.

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

[76] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[77] Hong-Xing Yu, Leonidas J. Guibas, and Jiajun Wu. Unsupervised discovery of object radiance fields. In *International Conference on Learning Representations*, 2022.

[78] Jiakai Zhang, Xinhang Liu, Xinyi Ye, Fuqiang Zhao, Yanshun Zhang, Minye Wu, Yingliang Zhang, Lan Xu, and Jingyi Yu. Editable free-viewpoint video using a layered neural representation. In *ACM SIGGRAPH*, volume 40, pages 1–18, 2021.

[79] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.

[80] Qing Zhang, Yongwei Nie, Lei Zhu, Chunxia Xiao, and Wei-Shi Zheng. A blind color separation model for faithful palette-based image recoloring. *IEEE Transactions on Multimedia*, 2021.

[81] Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. Palette-based image recoloring using color decomposition optimization. *IEEE Transactions on Image Processing*, 26(4):1952–1964, 2017. MR3636243

[82] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. NeRFactor: neural fac-

torization of shape and reflectance under an unknown illumination. *arXiv preprint arXiv:2106.01970*, 2021.

[83] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3D representations. *Advances in Neural Information Processing Systems*, 31, 2018.

Qiling Wu
Key Laboratory of Pervasive Computing
Ministry of Education
Beijing National Research Center for Information Science and Technology
Department of Computer Science and Technology
Tsinghua University
Beijing, China
E-mail address: wql19@mails.tsinghua.edu.cn

Jianchao Tan
Kuaishou Technology
Beijing
China
E-mail address: tanjianchaoustc@gmail.com

Kun Xu
Key Laboratory of Pervasive Computing
Ministry of Education
Beijing National Research Center for Information Science and Technology
Department of Computer Science and Technology
Tsinghua University
Beijing
China
E-mail address: xukun@tsinghua.edu.cn

Received September 16, 2023