Figure 1: Given a pre-trained NeRF model, the user can (a) choose a view and (b) draw a mask to specify the unwanted object in the 3D scene. Our framework optimizes the NeRF model based on a user-provided mask and removes the unwanted object in the mask region. The optimized NeRF synthesizes inpainted results that match ground truth results from different views. (Noted that the user-chosen view is different from view A and view B.)

KEYWORDS
Wireless sensor networks, media access control, multi-channel, radio interference, time synchronization

1 INTRODUCTION
Recent advancements in neural rendering, such as Neural Radiance Fields (NeRF) [Mildenhall et al. 2020] has emerged as a powerful representation for the task of novel view synthesis, where the goal is to render unseen viewpoints of a scene from a given set of input images. NeRF encodes the volumetric density and color of a scene within the weights of a coordinate-based multi-layer perceptron. Several follow-up works extend original NeRF to handle different tasks, such as pose estimation [Lin et al. 2021; Wang et al. 2021b], 3D-aware image synthesis [Chan et al. 2021; Niemeyer and Geiger 2021; Schwarz et al. 2020], deformable 3D reconstruction [Liu et al. 2021a; Park et al. 2021; Pumarola et al. 2020], and modeling dynamic scenes [Gao et al. 2021; Guo et al. 2021; Xian et al. 2021].

Though NeRF achieves great performance on photo-realistic scene reconstruction and novel view synthesis, there remain enormous challenges in editing the geometries and appearances of a scene represented by a pre-trained NeRF model. Unlike traditional image editing, a user needs to transfer his/her edits on a rendered view to the NeRF model to edit the whole scene, thus introducing multiple challenges. First, it is unclear where the edited regions...
appear on other rendered views. Second, because millions of parameters are used in a pre-trained NeRF model, 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. Previous works [Liu et al. 2021b] enable users to perform color and shape editing on a category-level NeRF. However, these methods require additional category-specific training and data to support the desired editings.

In this paper, we focus on the NeRF inpainting problem, i.e., removing unwanted objects in a 3D scene represented by a pre-trained NeRF. Although we can ask a user to provide a mask and the inpainted image for each rendered view and use these images to train a new NeRF, there are several disadvantages. First, it is labor-intensive to provide masks for many rendered views. Second, there will be visual inconsistency across different inpainted views introduced by separate inpainting.

To address these issues, we propose a framework to help users easily remove unwanted objects by updating a pre-trained NeRF model. Given a pre-trained NeRF, the user first draws a mask over a rendered view. Given the user-drawn mask, our framework first renders a couple of views sampled from a pre-set trajectory. Next, we transfer the user-drawn masks to these sampled views using existing video object segmentation method [Cheng et al. 2021]. Our framework then generates (i) guiding color image regions using [Cao and Fu 2021] and (ii) guiding depth images using Bilateral Solver [Barron and Poole 2016] within these masked regions. Noted that our framework can use any existing methods for generating guiding color and depth images. Finally, we formulate an optimization problem that jointly inpaints the image content within the all transferred masked regions with respect to the guiding color and depth images.

We demonstrate our framework on several scenes represented by pre-trained NeRFs in LLFF dataset. We show that our framework generates visually plausible and consistent results. Furthermore, we also demonstrate our experiments on the custom dataset to show the correctness between inpainted results and ground truth results.

2 RELATED WORK

2.1 Novel view synthesis

Constructing novel views of a scene captured by multiple images is a long-standing problem in computer graphics and computer vision. Traditional methods use structure-from-motion [Hartley and Zisserman 2003] and bundle adjustment [Triggs et al. 1999] to reconstruct explicit point cloud structure and recover camera parameters. Other methods synthesize novel views by interpolating within a 4D light fields [Gortler et al. 1996; Levoy and Hanrahan 1996] and by compositing the warped layers in the multiplane image representations (MPI) [Tucker and Snavely 2020; Zhou et al. 2018]. Recently, the coordinate-based neural representations have shown significant promise as an alternative to discrete, grid representations for scene representations. Neural Radiance Fields (NeRF) [Mildenhall et al. 2020] use a multi-layer perceptron (MLP) and positional encoding to model a radiance field at an unprecedented level of fidelity. However, NeRF works on a large number of input images and requires lengthy training time. Many works attempt to reduce the number of images NeRF requires by introducing depth-supervised loss [Deng et al. 2021; Roessle et al. 2021] and category-specific priors [Yu et al. 2021b]. Meanwhile, previous works try to reduce the training time by optimizing voxel grids of features [Sun et al. 2021; Yu et al. 2021a] and factoring the radiance field [Chen et al. 2022].

These recent advances greatly improve the practical use of NeRF. However, it is still unintuitive how a user can edit a pre-trained NeRF model. The main reason is because the neural network of a NeRF model has millions of parameters. Which parameters control the different aspects of the rendered shape and how to change the parameters to achieve desired edits are still unknown. Previous works enable users to select certain object [Ren et al. 2022], edit a NeRF model using strokes [Liu et al. 2021b], natural language [Wang et al. 2021a], and by manipulating 3D model [Yuan et al. 2022] directly. However, these methods require to learn additional category-level conditional radiance fields or segmentation network to facilitate such edits. Unlike these methods, our framework did not requires the user to prepare any additional category-specific training data and training procedure for removing unwanted objects in a pre-trained NeRF model.

2.2 Image inpainting

In recent years, two broad approaches to image inpainting exist. Patch-based method [Barnes et al. 2009; Efros and Leung 1999; Simakov et al. 2008] fill the holes by searching for patches with similar low-level image features such as rgb values. The search space can be the non-hole region of the input image or from other reference images. The inpainted results are obtained by a global optimization after the relevant patches are retrieved. These methods often fail to handle large holes where the color and texture variance is high. Meanwhile, these methods often cannot make semantically aware patch selections. Deep learning-based methods often predict the pixel values inside masks directly in a semantic-aware fashion. Thus they can synthesize more visually plausible contents especially for images like faces [Li et al. 2017; Yeh et al. 2017], objects [Pathak et al. 2016] and natural scenes [Iizuka et al. 2017]. However, these methods often focus on regular masks only. To handle irregular masks, partial convolution [Liu et al. 2018] is proposed where the convolution is masked and re-normalized to utilize valid pixels only. Yu et al. [Yu et al. 2018] uses GAN mechanism to maintain local and global consistency in the final results. Nazeri et al. [Nazeri et al. 2019] focus on improving the image structure in the inpainting results by conditioning their image inpainting network on edges in the masked regions. MST inpainting [Cao and Fu 2021] and ZITS [Dong et al. 2022] further consider both edge and line structure to synthesize more reasonable results. In this work, we use MST inpainting network [Cao and Fu 2021] to obtain the guided inpainted result because of its superior performance on inpainting images while preserving structures. Our framework can replace MST inpainting with other inpainting methods since we only used the inpainted results as a guiding signal for our optimization problem.

3 METHOD

In this section, we first summarize the mechanism of NeRF [Mildenhall et al. 2020] and formulate our problem setting.
where the rendered image of a pre-trained NeRF model $F$ as a function that takes a camera position ($\mathbf{r}$) and 2D viewing direction $d = (\theta, \phi)$ as input and outputs volume density $\sigma$ and directional emitted color $c$. NeRF renders the color of each camera rays passing through the scene by computing the volume rendering integral using numerical quadrature. The expected color $\hat{C}(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}$ is defined as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T(t_i) (1 - \exp(-\sigma(t_i) \delta_i)) c(t_i), \quad (1)$$

where $T(t_i) = \exp(-\sum_{j=1}^{i-1} \sigma(t_j) \delta_j)$, \quad (2)

where $N$ denotes the total quadrature points sampled between near plane $t_n$ and far plane $t_f$ of the camera, and $\delta_i = t_{i+1} - t_i$ is the distance between two adjacent points. We denote the color and density at point $t_i$ produced by NeRF model $F_{\Theta}$ as $c(t_i)$ and $\sigma(t_i)$.

Using the above differentiable rendering equation, we can propagate the errors and update $\Theta$ through mean square error:

$$L_{mse} = \sum_{\mathbf{r} \in \mathcal{R}} \| (\hat{C}(\mathbf{r}) - C(\mathbf{r})) \|^2 + \| (\hat{C}'(\mathbf{r}) - \hat{C}'(\mathbf{r})) \|^2_2, \quad (3)$$

where $\mathcal{R}$ is a ray batch, $C(\mathbf{r}), \hat{C}(\mathbf{r}), \hat{C}'(\mathbf{r})$ are the ground truth, coarse volume predicted, and fine volume predicted RGB colors for ray $\mathbf{r}$ respectively. For simplicity, we further define $\hat{C}(\mathbf{r})$ as a function that takes a camera position ($\mathbf{o}$) as input, and outputs the rendered image of a pre-trained NeRF model $F_{\Theta}$.

3.1 Preliminaries: NeRF

NeRF is a continuous volumetric radiance field $F_{\Theta} : (\mathbf{x}, d) \rightarrow (c, \sigma)$ represented by a MLP network with $\Theta$ as its weights, $F_{\Theta} : (\mathbf{x}, d) \rightarrow (c, \sigma)$ takes a 3D position $\mathbf{x} = (x, y, z)$ and 2D viewing direction $d = (\theta, \phi)$ as input and outputs volume density $\sigma$ and directional emitted color $c$. NeRF renders the color of each camera rays passing through the scene by computing the volume rendering integral using numerical quadrature. The expected color $\hat{C}(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}$ is defined as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T(t_i) (1 - \exp(-\sigma(t_i) \delta_i)) c(t_i), \quad (1)$$

where $T(t_i) = \exp(-\sum_{j=1}^{i-1} \sigma(t_j) \delta_j)$, \quad (2)

where $N$ denotes the total quadrature points sampled between near plane $t_n$ and far plane $t_f$ of the camera, and $\delta_i = t_{i+1} - t_i$ is the distance between two adjacent points. We denote the color and density at point $t_i$ produced by NeRF model $F_{\Theta}$ as $c(t_i)$ and $\sigma(t_i)$.

Using the above differentiable rendering equation, we can propagate the errors and update $\Theta$ through mean square error:

$$L_{mse} = \sum_{\mathbf{r} \in \mathcal{R}} \| (\hat{C}(\mathbf{r}) - C(\mathbf{r})) \|^2 + \| (\hat{C}'(\mathbf{r}) - \hat{C}'(\mathbf{r})) \|^2_2, \quad (3)$$

where $\mathcal{R}$ is a ray batch, $C(\mathbf{r}), \hat{C}(\mathbf{r}), \hat{C}'(\mathbf{r})$ are the ground truth, coarse volume predicted, and fine volume predicted RGB colors for ray $\mathbf{r}$ respectively. For simplicity, we further define $\hat{C}(\mathbf{r})$ as a function that takes a camera position ($\mathbf{o}$) as input, and outputs the rendered image of a pre-trained NeRF model $F_{\Theta}$.

3.2 Overview

Given a pre-trained NeRF model: $F_{\Theta}$, a user can specify the unwanted region by drawing a mask $M_o$ over a user-chosen rendered view $I_u = F_{\Theta}(\mathbf{o}_u)$, where $M_o = 1$ for pixel outside the masked region, and $\mathbf{o}_u$ is the user-chosen camera position. Our goal is to obtain an updated NeRF model $F_{\Theta}$ such that the unwanted region masked by $M_o$ is removed in every rendered views. As shown in Figure 2, our method first sample $K$ camera positions $O = \{\mathbf{o}_s = 1,...K\}$ along the test-set camera trajectory used in LLFF [Mildenhall et al. 2019] (Figure 3). For each camera position, we rendered a rgb image $I_o$ and depth image $D_o$ using $F_{\Theta}$ and obtained all rendered views $I = \{I_s = 1,...K\}$ and their depth images $D = \{D_s = 1,...K\}$. We will use $I_o$ and $I(\mathbf{o}_s)$ to represent the image rendered from camera position $\mathbf{o}_s$ interchangeably throughout the
We obtain the updated parameters $\tilde{\Theta}$ where $\tilde{\Theta} = \arg\min_\Theta L_{\text{color}}(\Theta) + L_{\text{depth}}(\Theta)$  

3.4 NeRF inpainting optimization

We obtain the updated parameters $\tilde{\Theta}$ that removes the unwanted object in the 3D scene by optimizing: 

$$L_{\text{mmse}} = \sum_{r \in R} \| (\hat{C}_s^{\tau}(r) - C_s^{\tau}(r)) \odot M(r) \|_2^2 + \| (\hat{C}_d^{\tau}(r) - C_d^{\tau}(r)) \odot M(r) \|_2^2,$$

where $M$ is the transferred mask on the same view as the sample ray $r$. However, because there is no explicit guidance on what image content and structure should be in the masked region, the unwanted object will remain in the result of optimizing Equation (5). To provide explicit guidance, our method takes the user-drawn mask $M_u$, sampled rendered views $I$, and sampled depth images $D$ as input, and outputs:

- guiding user-chosen image and guiding depth image: $I_s^G$ and $D_s^G$.
- transferred masks: $M = \{M_s|s = 1...K\}$.
- guiding sampled images and their guiding depth images: $I^G = \{I_s^G|s = 1...K\}$ and $D^G = \{D_s^G|s = 1...K\}$.

Finally, our method obtains updated parameters $\tilde{\Theta}$ by optimizing our nerf inpainting formulation: $\Phi(M_u, I^G, M, D^G)$.

### 3.3 Guiding material generation

For each sampled rendered view $I_s$, our goal is to generate a mask $M_s$ that covers the same object as the user-drawn mask $M_u$. We use a video object segmentation method (STCN) [Cheng et al. 2021] to generate $M_s$. With the transferred masks $M_s$, we need to generate the guiding images and guiding depth images. The guiding image generation can be described as:

$$I_s^G = \rho(I_s, M_s),$$

where $I_s^G$ is the guiding image, and $\rho$ is a single image inpainting method (we used MST inpainting network [Cao and Fu 2021]). After obtaining $I_s^G$, we can obtain the guiding depth image using:

$$D_s^G = \tau(D_s, M_s, I_s^G),$$

where $D_s^G$ is the guiding depth image, and $\tau$ is a depth image completion method (we used Fast Bilateral solver [Barron and Poole 2016]). Noted that our framework can replace $\rho$ to any other single image inpainting method and $\tau$ to any other single depth image completion method.

4.2 Evaluation

4.2.1 Datasets. To verify our framework’s performance, we create a custom dataset that contains three custom scenes: $\text{figyua}$, $\text{desuku}$, and $\text{terebi}$. The purpose is to obtain the ground truth results of NeRF inpainting. For each custom scene, we collect a pair of photo set, i.e., (original and removed). For original set, we keep all the objects in the scene and take photos from 24 camera positions. For removed set, we remove one object in the scene manually and take

4.2.2 NeRF training. We implement our framework in PyTorch [Paszke et al. 2019] and Python 3.9. We test our framework on a machine with Intel i7-7800X and a GTX-1080 graphics card to train our models. For each scene, we first train a model initialized to random weights and optimize it for 200,000 steps with a batch size of 4,096 using Adam [Kingma and Ba 2014], which takes about 18 to 20 hours. The sample points used in fine and coarse model are 128 and 64, respectively. To inpaint each scene, we optimize Eq. 7 for 50,000 steps which takes about five hours in total.
We then compared the inpainted result with the image generated by the pre-trained NeRF model using the removed set and the baseline2 method and two baseline methods using LLFF dataset in Figure 4. We show the qualitative comparison between our LLFF dataset by the pre-trained NeRF model using the original set. For each scene, we trained two separate models to the pre-trained NeRF model using the NeRF models for the original set and the removed set. We apply the three methods on each pre-trained dataset described in Section 4.2.1. For LLFF dataset, we perform qualitative evaluation by applying three methods: per-view color updating, masked mse retraining. We update the pre-trained NeRF model using all guiding images \( I^G \) by optimizing:

\[
\hat{\Theta} := \arg \min_\Theta \sum_{I^G \in G} (F^\text{image}_\Theta(o_s) - T^G_s) \quad (12)
\]

**baseline2: masked mse retraining** We re-train a new NeRF model using all guiding images \( I^G \) by optimizing:

\[
\hat{\Theta} := \arg \min_\Theta \sum_{I^G \in G} (F^\text{image}_\Theta(o_s) - T^G_s) \odot M_s, \quad (13)
\]

where \( \Theta \) is randomly initialized.

Both baseline1 and baseline2 did not consider depth information during updating the pre-trained NeRF model or re-train a new NeRF model.

We compared the inpainted results of our framework to inpainted results of two baseline methods on LLFF dataset and our custom dataset described in Section 4.2.1. For LLFF dataset, we perform qualitative evaluation by applying three methods on each pre-trained model. For our custom dataset, we perform both qualitative and quantitative evaluations. For each scene, we trained two separate NeRF models for the original set and the removed set. We apply three methods to the pre-trained NeRF model using the original set. We then compared the inpainted result with the image generated by the pre-trained NeRF model using the removed set.

**4.2.3 Results and discussions.**

**LLFF dataset** We show the qualitative comparison between our method and two baseline methods using LLFF dataset in Figure 4 and Figure 6. In Figure 4, we observed that the depth maps of the inpainted NeRF generated by baseline1 did not match the inpainted image content. In Figure 6, we showed that there are obvious visual inconsistencies between views in the results generated by baseline1. To avoid these visual inconsistency, we choose to provide color guidance using only the user-chosen view and let the NeRF model maintain the view consistency by itself. Baseline2 recovers visual satisfactory image content without any color guidance inside the masked region. However, baseline2 still generate results that losses fine structures or synthesize some unnatural patches at complicated regions, which can be oberserved at Horns and Orchids.

**Custom dataset** We show the qualitative comparison between our method and two baseline methods using our custom dataset in Figure 1 and Figure 5. In Figure 5, we can observe that although baseline1 can synthesize rgb content closer to ground truth, it still fail to generate correct depth map. On the other hand, baseline2 recovers the content in the masked region guided by the content from different views but still creates noisy and blurry content. Our framework generates closer color and depth images to the ground truth rendered results compared to the two baseline methods. Overall, our inpainting optimization updates a pre-trained NeRF model to obtain correct geometry and preserve visual consistency across views.

**4.3 Ablation study**

**4.3.1 How important is the depth-guiding loss?** Introducing the depth-guiding loss \( L_{\text{depth}} \) is one of the major contribution of our framework. We validate its effectiveness by comparing with the optimization results using color-guiding loss \( L_{\text{color}} \) only, depth-guiding loss \( L_{\text{depth}} \) only, and both losses.

We showed the results in Figure 7. We observe that optimizing using \( L_{\text{depth}} \) only already leads to correct geometries inside the masked region but introduces color noises outside the masked region. Our method, compared to these two baselines, have better color and correct geometry on final results.
Figure 5: Qualitative comparison - custom dataset. For each custom scene, we demonstrate the ground truth rendered image, results generated by our framework, \textit{baseline1} (b1), and \textit{baseline2} (b2). Our framework generates more accurate depth maps and synthesize more fine structures compared to \textit{baseline1}. Compared to \textit{baseline2}, our framework synthesizes more realistic and shape results.
might output undesired or incorrect results. Meanwhile, our framework shares the same limitations as the image inpainting method we used. For example, our framework fails to inpaint the image region with high reflectance content (Figure 10) or with a thin structure. It is possible to design a fusion method to fuse color and depth guidances from multiple methods.

**Update masks and guidance materials.** In our current framework, we used fixed masks and guidance materials during the optimization. However, this is sub-optimal when the unwanted object is occluded in some views. In the future, we plan to extend our framework to update the masks in every optimization step using 3D volume features extracted from the NeRF model. We also plan to use a discriminator to constrain the synthesized results and improve the view consistency.

**Volume feature for mask transferring.** Our current framework uses the existing video-based object segmentation method to transfer the user-drawn mask. It is possible to perform mask transferring by conducting 3D volume segmentation using the volume feature extracted from the pre-trained NeRF.

### 6 CONCLUSION

In this paper, we propose the first framework that enables users to remove unwanted objects or retouch undesired regions in a 3D scene represented by a pre-trained NeRF. Our framework requires no additional category-specific data and training. Instead, we formulated a novel optimization to inpaint the pre-trained NeRF with the generated RGB-D guidances. We demonstrated our framework handles a variety of scenes well, and we also validate our framework using a custom dataset where ground truth inpainted results are available. We believe that the custom dataset we proposed and our framework can foster future research on neural radiance field editing.

### REFERENCES


Update masks and guidance materials. In our current framework, we used fixed masks and guidance materials during the optimization. However, this is sub-optimal when the unwanted object is occluded in some views. In the future, we plan to extend our framework to update the masks in every optimization step using 3D volume features extracted from the NeRF model. We also plan to use a discriminator to constrain the synthesized results and improve the view consistency.

**Volume feature for mask transferring.** Our current framework uses the existing video-based object segmentation method to transfer the user-drawn mask. It is possible to perform mask transferring by conducting 3D volume segmentation using the volume feature extracted from the pre-trained NeRF.

### 5 LIMITATIONS AND FUTURE WORK

**Fuse color and depth guidance.** Our framework leverages existing image inpainting and depth completion method to generate initial guidance materials. Once there are appearances or geometries artifacts in the initial guidance materials, the optimized NeRF
Figure 7: Depth-guiding ablation result. We show the optimization results using different guiding losses. We observed that the geometry of the unwanted object could not be removed using color-guiding term only (depth inside the blue box). On the other hand, using depth-guiding term only helps to get correct geometry but introduces color noises outside the masked region (red box and yellow box). Our method combines both terms to generate correct geometry (blue box) without introducing any color noises (red box and yellow box).

Figure 8: Depth-guiding losses discussion. The unwanted object in the region with high depth variations can be removed by using depth-guiding loss (red box). However, using depth-guiding loss only loses color information in the flat region (blue box). Combining both losses removes the unwanted object without losing any color information.

Figure 9: Color-guiding ablation result. We show the optimization results using different number of views to compute \( L_{all \color{red}{\text{color}}} \). The visual inconsistency becomes larger in the masked region (red box) when we increase the number of views used to compute \( L_{all \color{red}{\text{color}}} \).


Figure 10: Our framework fails to inpaint the mask region in the left region (red box) and introduces artifacts in the optimized results.


