3D Inpainting with Decouplable Gaussian Representation and 2D priors

Anonymous Authors

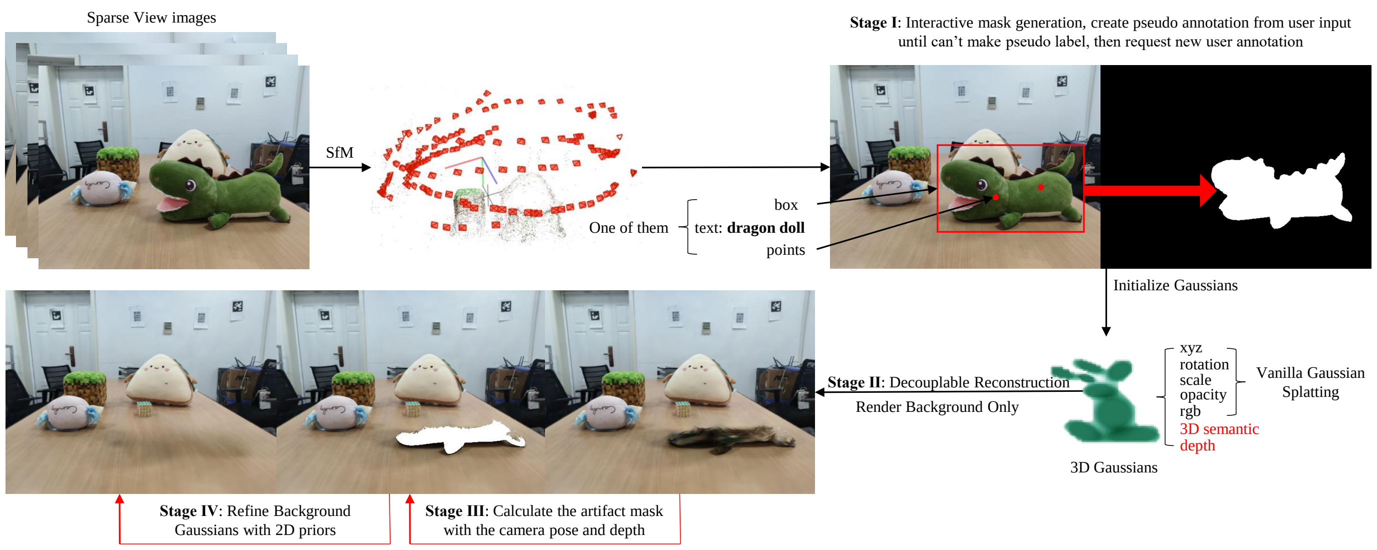
Sparse View images
SfM
Stage I: Interactive mask generation, create pseudo annotation from user input until can't make pseudo label, then request new user annotation
One of them
box
text: dragon doll
points
Initialize Gaussians
xyz
rotation
scale
opacity
rgb
3D semantic
depth
Vanilla Gaussian Splatting
3D Gaussians
Stage II: Decouplable Reconstruction
Render Background Only
Stage IV: Refine Background Gaussians with 2D priors
Stage III: Calculate the artifact mask with the camera pose and depth

Fig. 1. An overview of our 3D inpainting framework. We use the SfM method to estimate camera parameters from sparse views. An interactive mask generation (Sec IV-A) strategy is applied to generate 2D masks from any points, boxes, or text prompts. Then, we propose a decouplable Gaussian Splatting (Sec IV-B) to reconstruct the scene and render the background only to get a raw inpainting result. Third, the artifact mask is calculated from 3D geometry (Sec IV-D), and the final inpainting result is obtained by refining the background Gaussians with corrected 2D priors from 2D inpainting (Sec IV-E).

***Abstract*—Though 2D inpainting has been well-studied in the era of deep learning, 3D inpainting remains challenging. This paper proposes a novel pipeline for inpainting the 3D scene by utilizing the advantages of the 3D reconstruction framework Gaussian Splatting and 2D inpainting models trained on large datasets together. First, we design an interactive process to predict 2D masks with Segment-Anything (SAM). Beginning with user annotation on only one view, our method uses Structure from Motion (SfM) information to generate pseudo annotations for other views and asks users to add annotations if no pseudo ones can be produced for the remaining views. Second, a decouplable Gaussian field is reconstructed from GT images, 2D masks from the first stage, and additional SfM depth supervision. We get raw inpainting results by only rendering the background Gaussians and correcting these results with pre-trained 2D inpainting models. Third, we propose an algorithm leveraging camera pose and stage two depth to find the artifact region in raw inpainting results to ensure concurrent 2D inpainting quality and 3D consistency. Finally, we refine the background Gaussians for minority epochs with the supervision of the corrected inpainting results to get the scene after removal. Additionally, we explicitly derive the gradients with CUDA implementation to maintain the fast speed in Gaussian Splatting. Experiments demonstrate that our method achieves better inpainting quality and can handle challenging scenarios for existing works.**

***Index Terms*—3D inpainting, multiview segmentation, gaussian splatting**

I. Introduction

As the Neural Radiance Field (NeRF) [1] has demonstrated its superiority in 3D reconstruction, a series of derivative works [2]–[6] have also emerged, including adapting NeRF to 3D editing [7]–[11]. 3D inpainting to remove masked portions from a 3D scene, as an essential component of editing functionality, has also attracted significant research interest. Nevertheless, 3D inpainting remains a challenging task. When using neural radiance fields to reconstruct 3D scenes, one field is specific to that particular scene, and reconstructing another requires training a new neural field from scratch. For reconstruction tasks, the per-scene, per-network paradigm is acceptable as long as sufficient 2D views are available for supervision. However, considering inpainting based on 3D reconstruction means lacking information about the background after deletion, significantly increasing the difficulty of reasonable inpainting.

A natural approach to address this issue is introducing prior knowledge from 2D inpainting. Since 2D inpainting networks [12], [13] are trained on large datasets, they can infer unseen data as well as predict the background texture for 3D inpainting. However, effectively utilizing 2D inpainting is not straightforward. On the one hand, directly applying the mask of the desired region may yield suboptimal results because the portion to be filled is too extensive, even for well-trained

2D inpainting networks. On the other hand, 2D inpainting networks do not consider any 3D information, often leading to inconsistency. In 3D scenes, occluded background pixels in one view may be visible in another. However, 2D inpainting only provides a potential answer based on its learned data distribution, likely causing conflicts in the 3D context.

While several works have focused on 3D inpainting, they still face the challenge of simultaneously achieving realistic 2D rendering and 3D consistency. The ObjectNeRF [7] and ObjSDF [8] methods achieve editing functionality by decoupling the neural radiance field into the foreground and background. Compared to reconstruction networks, they introduce a learnable object ID and use 2D segmentation maps as supervision to learn this value. Implementing the inpainting functionality involves not rendering the unwanted parts controlled by the object ID. Not surprisingly, due to the absence of the covered background, if these pixels are not visible from any view, their color will be set to the default background color black. Consequently, "black hole" artifacts appear in the inpainting results. SPIn-NeRF [10], NeRF-Object-Removal [9] and OR-NeRF [14] directly use the results of 2D inpainting [13] as the ground truth to reconstruct the deleted scene. However, as analyzed earlier, these methods do not perform well in complex scenes, especially with occlusions, as they ignore 3D consistency. Additionally, if 2D inpainting produces suboptimal results, improving the render quality is challenging for these methods. Figure 2 illustrates the discussed two flaws.

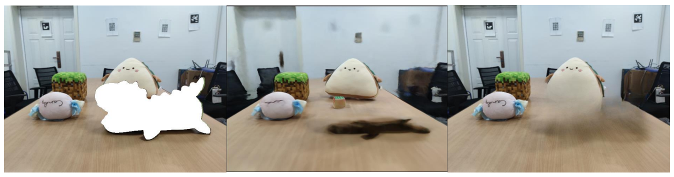
Fig. 2. Flaws of the current method: the masked image (left), the inpainting results of ObjSDF [8] (middle) and SPIn-NeRF [10] (right). We can see the "black hole" artifact in ObjSDF and 3D inconsistency in SPIn-NeRF.

In this paper, we propose a novel 3D inpainting pipeline that leverages the characteristics of 3D reconstruction to maintain 3D consistency and the ability of 2D inpainting to generate background priors simultaneously. Figure 1 is the overview of our method. Our key idea is finding the unseen pixels from all views to bridge 3D reconstruction and 2D inpainting. First (Sec IV-A), we design an interactive mask generation strategy with the 2D segmentation model Segment-Anything (SAM) [15]. To begin with, users annotate an arbitrary image with points, text, or box prompts as long as SAM can provide the desired mask. Then, Structure from Motion (SfM) [16] information will be used to map points in 3D and 2D space. The foreground 3D point cloud will be maintained to aggregate 3d points in each view's mask and generate pseudo annotations for other views. If creating new reliable pseudo annotations is impossible, the user is asked to make a new annotation. The above strategy is made by the observations in practice that annotating once is hard to guarantee all views' masks, while annotating each image is unnecessary as we can utilize 3D information to annotate adjacent views automatically.

Second (Sec IV-B), we turn the SOTA 3D reconstruction method Gaussian Splatting [17] decouplable by adding an attribute to each Gaussian indicating its belongings (foreground or background) and learn this variable through 2D mask supervision. Further, we add depth supervision to the reconstruction process for rendering the depth map used to calculate the artifact mask (Sec IV-D). Raw 3D inpainting results can be gained by rendering background Gaussians only. There will be "black hole" artifacts caused by the background pixels that are never seen from all the views. Third (Sec IV-D), we utilize the scene depth obtained from the second stage with camera poses to calculate the artifact mask and inpainting this with pre-trained 2D inpainting models [12], [13]. Finally (Sec IV-E), we refine the trained background Gaussians supervised by the corrected raw results. Experiments show that 1/15 epochs of the reconstruction stage is enough for refinement. Additionally, we follow the practice of Gaussian Splatting to explicitly calculate the gradients of our additions to the vanilla version with CUDA implementation.

To conclude, our contributions are:

- A novel 3D inpainting pipeline with better and more robust performance.
- An interactive multiview segmentation strategy that automatically makes pseudo annotations with 3D geometry, balancing precision and human labor with broader scenes.
- A decouplable Gaussian field with CUDA implementation that maintains the speed advantage of vanilla one.
- An artifact mask finding algorithm with depth and camera poses that bridges 3D reconstruction and 2D inpainting.

II. Related Work

A. Multiview Segmentation

While 2D segmentation has been extensively explored in various studies [15], [18]–[31], multiview segmentation for 3D scenes remains relatively under-researched, despite its significant role in applications like 3D inpainting. Although some self-supervised methods have been developed for this purpose [32], [33], they often struggle with complex scenes and fail to deliver satisfactory mask accuracy. Semi-supervised strategies [10], [34]–[36] requiring partial labels or user annotations have been proposed to mitigate the challenge. SemanticNeRF [36] observes that partially annotated labels are enough to predict full masks as 3D information is aggregated during training. Specifically, they add a semantic label to the model output and use the exact volume rendering in NeRF [1] to render this label to 2D segmentation maps. Furthermore, SPIn-NeRF [10] constructs a comprehensive multiview segmentation pipeline using point annotations from a single view. They initiate the process with one-shot segmentation [19] to generate an initial mask. This is followed by a video segmentation approach [13], [18], which treats the image sequence as a video to produce masks for all views. Finally, they follow the practice of SemanticNeRF by changing the reconstruction backbone from NeRF to InstatnNGP [37] to refine the masks. This procedure consumes considerable resources. Besides, OR-NeRF [14] achieves multiview segmentation with points annotation on a single view by mapping points between 2D and 3D with projection. However, this method only works for forward-facing scenes as they do not consider projection errors caused by occlusions.

B. 3D Inpainting

NeRF has significantly advanced the field of 3D scene editing and research [38]–[41], fostering the development of various editing techniques. Existing works focus on geometry modification [2]–[4], texture replacing [5], [42], and multi-functional capabilities [7], [8], [11], [43]–[52]. ObjectNeRF [7] and ObjSDF [8] decompose NeRF training into distinct background and object branches, enabling the rendering of specific objects controlled by assigned IDs. However, these methods often produce artifacts in the regions where objects are removed due to the absence of supervision during the deletion process. Methods such as SPIn-NeRF [10], NeRF-Object-Removal [9], and OR-NeRF [14] address this issue by leveraging 2D inpainting techniques [13] to reconstruct scenes using 2D inpainting results as ground truth. Despite achieving better rendering quality, these approaches suffer from 3D inconsistency and are limited by the performance of 2D inpainting methods. Additionally, they require masks for all views, which depend on either labor-intensive human efforts [9] or computationally expensive processes [10]. Furthermore, methods like n3f [53] and distilled-feature-fields [46] integrate pre-trained language models [18], [21], [53], [54] to facilitate text-based editing, thereby circumventing the need for masks. Nevertheless, these techniques exhibit poor rendering quality in removal regions due to the lack of algorithms designed for ”unseen” pixels and fail if the text prompt does not accurately identify the inpainting area.

III. Background: Gaussian Splatting

Gaussian Splatting [17] is the recent SOTA method for 3D reconstruction. It represents the 3D scene with 3D Gaussians and renders 2D images with rasterization. The 3D Gaussian is defined by its probability density function as:

$$G(x) = e^{-\frac{1}{2}(x)^T\Sigma^{-1}(x)}, \tag{1}$$

where x is the mean of Gaussian, Σ is the covariance matrix for random variable x. In the 3D scene, the Gaussian representation has geometry meaning as x represents the 3D coordinates and Σ is constructed with the scale matrix S and rotation matrix R for the 3D Gaussian ellipsoid: $\Sigma = RSS^TR^T$.

Then, 3D Gaussians are projected [55] to the image plane by $\Sigma^{'} = JW\Sigma W^TJ^T$ for later rendering. Here, W is the viewing transformation, and J is the Jacobian of the affine approximation of the projective transformation. The 2D projection of a 3D Gaussian is then approximated from an ellipse to a circle. After this, the image plane is split into 16x16 tiles. The projected 2D Gaussians in each tile are sorted by their depth. Finally, the image RGBs are calculated by an α blending with the 2D Gaussians overlapping on that pixel, similar to volume rendering:

$$C = \sum_{i\in\mathcal{N}} c_i\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j). \tag{2}$$

Here, c_i is the color of each Gaussian, and α_i is calculated by covariance Σ multiplying with a learned per-Gaussian opacity. Finally, the Gaussians are optimized with the classic gradient descent, supervised by RGB values only using the weighted sum of $\mathcal{L}_1$ term and D-SSIM term.

$$\mathcal{L}_{gs} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{D-SSIM}. \tag{3}$$

Gradients are explicitly derived to avoid slow computation with automatic differentiation, and the forward and backward passes are implemented with CUDA code.

IV. Method

In this section, we first explain how we generate 2D masks interactively in Sec IV-A, second how to turn vanilla Gaussian Splatting decouplable in Sec IV-B and IV-C, third how to find the mask of the unseen pixels Sec IV-D and how to refine the Gaussians with 2D inpainting priors in Sec IV-E.

A. Stage I: Interactive Mask Generation

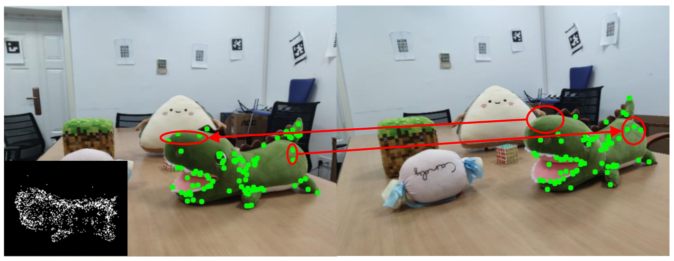

Fig. 3. This figure displays a toy demo explaining how our mask generation strategy works. With the views changing from left to right, one more point can be found near the tail. Three more points can be observed near the head from right to left. As the lower left corner shows, the foreground point cloud P^3 aggregates points from new mask predictions as the view changes and finally becomes accurate.

To begin with, we use SAM [15] to predict masks for the foreground to be inpainted in this stage. Vanilla SAM uses points or boxes as prompts, and we first explain how our method works with points prompts and then describe other prompts like boxes and text.

Given a set of 2D sparse view images $\mathcal{I} = \{I_i\}_{i=1}^n$ belonging to a 3D scene, we first reconstruct the camera parameters for this scene with the SfM method. We follow Gaussian Splatting to use COLMAP [16]. COLMAP first extracts features from 2D images and matches each view with the extracted features to form a sparse point cloud. This process typically gives key points with accurate mappings between 2D and 3D space. Formally, assume each image I_i has a set of 2D key points $\{P_i^2\}_{j=1}^m$ from COLMAP, where m is the number of points in each image I_i. We can query the corresponding 3D points P_i^3 of 2D key points P_i^2 directly with COLMAP. Intuitively, we can aggregate P_i^3 from all views whose corresponding 2D points P_i^2 are in the mask area to form the foreground's point cloud P^3.

Specifically, we require the user to annotate the desired region for an arbitrary view i with points. The annotation is acceptable as long as SAM can produce the desired mask.

Then, we query P_i^3 with P_i^2 in the mask area and initialize P^3 with P_i^3. Then, we query 2D key points on all remaining views with P^3 and choose the view j with the maximum number of 2D key points. We generate a mask for this chosen view j with queried 2D key points as pseudo annotations. We update P^3 with P_j^3 whose corresponding P_j^2 are in the mask of view j. The updation can be done by union P_i^3 and P_j^3. Repeat the above process until we cannot provide pseudo labels for the rest of the views. We request that the user add annotations for one of the remaining views and continue the above process until we handle all the views.

However, using key points only does not guarantee sufficient pseudo-point annotations. This is because the key points numbers are too small compared to the whole image's pixel numbers. Figure 11 in Sec V-C also shows a corner case in our method due to the lack of key points in the desired inpainting region. Views cannot have adequate key points in the mask area. To alleviate this issue, we can use the camera pose to calculate the mappings between 2D and 3D under projection. Given view i's camera pose $[R_i|t_i]$ formed with rotation matrix R_i (Note R_i here is for the camera, and different from the one in Sec III for Gaussians) and translation matrix t_i, and camera intrinsic matrix K_i, we can find P_i^3 by projecting 2D points P_i^2 in the mask to 3D:

$$P_i^3 = [R_i|t_i]^{-1}[P_i^2 * d, 1]^T, \tag{4}$$

where d is the depth at pixel P_i^2. For the reverse mapping, we calculate 2D points from corresponding P_i^3 by:

$$P_i^2 = (R_i(P_i^3.(x,y)) + t_i)/P_i^3.z. \tag{5}$$

The problem is that we do not have depth d from SfM alone. As a workaround, we implement the 2D to 3D pass by projecting all 3D key points to the 2D image plane and then choosing the 3D points whose projected 2D pixels are marked as the mask. Another problem is that projection does not consider occlusions. 3D points in the background can project into the mask region, so we find 3D points from 2D points by query and calculate the 2D pixel coordinates from 3D points by projection. The 2D to 3D projection is an upper bound for the foreground point cloud. Figure 3 shows how our algorithm aggregates 3D points to form the accurate foreground point cloud.

As for the box prompt, this is straightforward as the first image starts with the box prompt while the rest are the same as points. For the text prompt, users can use 2D open text detection models like GroundingDINO [56], [57] to get the bounding box for the desired region first, then apply the strategies described above to handle the remaining views. As the text prompt can apply to all views directly, we suggest users choose the way that can produce the best segmentation results. Either points or text prompt has drawbacks: points prompt has difficulty processing inadequate views with sharp camera motion, while text fails when it cannot locate the desired region. Our interactive multiview mask generation algorithm is summarized in algorithm 1.

Algorithm 1: Multiview Mask Generation

```
Data: images $\mathcal{I} = \{I_i\}_{i=1}^n$, mapping between 2D and
      3D by COLMAP query or projection $f$, SAM
      model $\Theta_s$, GroundingDINO model $\Theta_g$.
Result: segmentation maps $\mathcal{S} = \{S_i\}_{i=1}^n$.

Initialization: User annotation $\mathcal{A}$;
if $\mathcal{A}$ is points or boxes then
    $\mathcal{P}_{3d} = f_{2d\to 3d}(\Theta_s(I_1, \mathcal{A}), 1)$;
else if $\mathcal{A}$ is text then
    if use text only then
        $\mathcal{S} = \Theta_s(\Theta_g(\mathcal{I}, \mathcal{A}))$;
        return $\mathcal{S}$;
    else if follow points prompt then
        $\mathcal{P}_{3d} = f_{2d\to 3d}(\Theta_s(I_1, \Theta_g(I_0, \mathcal{A})), 1)$;
waiting list $\mathcal{L} = \{I_i\}_{i=2}^n$;

while $len(\mathcal{L})$ do
    for $I_i \in \mathcal{L}$ do
        $\mathcal{P}_i^{2d} = f_{3d\to 2d}(\mathcal{P}_{3d}, i)$;
    $\mathcal{P}_{2d} = \max(len(\mathcal{P}_i^{2d}))$;
    $j = argmax(len(\mathcal{P}_i^{2d}))$;

    if $len(\mathcal{P}_{2d}) = 0$ then
        Add user annotation $\mathcal{P}_{2d}$;
    $S_j = \Theta_s(I_j, \mathcal{P}_{2d})$;
    $\mathcal{L}.pop(I_j)$;
    $\mathcal{P}_{3d} = \mathcal{P}_{3d} \cup f_{2d\to 3d}(\mathcal{P}_{2d}, j)$;

Filter $\mathcal{P}_{3d}$ (denoise);
for $i \leftarrow 1$ to $n$ do
    $S_i = \Theta_s(I_i, f_{3d\to 2d}(\mathcal{P}_{3d}, i))$;
return $\mathcal{S}$;
```

B. Stage II: Decouplable Gaussian Splatting

To decompose the scene into foreground and background, the natural idea is to assign each Gaussian a new attribute indicating its belonging. That is to say, learn a 3D mask for the foreground. Then, we can realize inpainting by simply rasterizing the background Gaussians only. However, previous practices [8], [10], [36] based on NeRF [1] architectures usually create and volume rendering an attribute that has no geometry meaning to the segmentation map. After that, 2D ground truth masks are applied as supervision. Though multiview segmentation has been achieved, the attribute bound to each input point cannot represent the 3D mask.

We create a new attribute s to each Gaussian to meet our demand. We use this s to render 2D segmentation maps by imitating the color blending process in Equation 2 like previous research. At the same time, we try to make this attribute indicate that the Gaussian belongs to the foreground or background. Ideally, s should be either 0 (background) or 1 (foreground), while it can be rasterized to the 2D segmentation

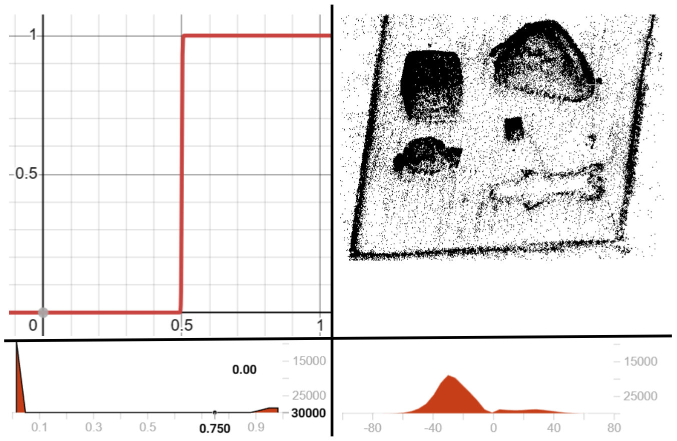

Fig. 4. Top left is the function curve of f_{db} with $k = 1000$, it cuts inputs to either 0 or 1 while is differentiable. The top right is the decoupled background Gaussians. We can see that s serves as the 3D mask satisfactorily. The bottom left is the histogram of s with Equation 8 transformation at training step 30000, and the bottom right is without transformation. We can see that raw s values have the rudiment of binarization, while our proposed algorithm binarizes s successfully.

map for loss computation. First, we add s to Gaussians and render the 2D segmentation map by changing the color c_i in Equation 2 to s_i:

$$S = \sum_{i \in \mathcal{N}} s_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{6}$$

where S is the segmentation map. We optimize s with binary cross entropy loss $\mathcal{L}_{bce}$:

$$\mathcal{L}_s = \mathcal{L}_{bce}(S, S_{gt}), \tag{7}$$

here S_{gt} is the ground truth 2D segmentation map obtained from Sec IV-A. We investigated the distribution of the trained s under this setting and found that it is already roughly binary with biased dwarf peaks (see Figure 4). So, an easy way is to constrain s to $(0, 1)$ first and then apply a binarization with a threshold of 0.5. However, binarization is not differentiable. With the experience from previous works [8], [58], [59], we apply a differentiable binary function f_{db} to s before it goes to Equation 6:

$$s = f_{db}(f_{sigmoid}(s)), \text{where } f_{db}(x) = \frac{1}{1 + e^{-k(x-t)}}, \tag{8}$$

here, k is a hyper-parameter to control the "sharpness" of the f_{db} function while t is the threshold for binarization. We first use the Sigmoid function to constrain s in $(0, 1)$ and set the threshold of the differentiable binary function f_{db} to 0.5. We can now optimize s with Equation 7 to render a 2D segmentation map and learn a 3D mask simultaneously. Figure 4 shows the features of f_{db} and the performance of our proposed s.

Go one step further to find the "black hole" artifact mask. Recall that a 2D to 3D mapping needs depth information, and occlusions in 3D geometry are also closely related to depth. We try to render the depth map in the reconstruction stage

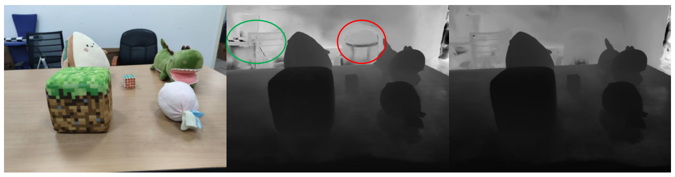

Fig. 5. From left to right are GT, depth map without and with supervision, respectively. The part in the green circle shows higher quality, while we can see over smoothness in the red circle.

for future usage in Sec IV-D. We obtain the depth d of each Gaussian with a slightly change to Equation 5:

$$d = R_i(P_i^3).z + t_i. \tag{9}$$

Note that Gaussians are not like points. They have shapes and occupy a specific space. We cannot use the depth of Gaussian as depth for the pixel directly. Therefore, we render the 2D depth map like s Equation 6:

$$D = \sum_{i \in \mathcal{N}} d_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{10}$$

where D is the depth map. Experiments show that only forward rendering without supervision can produce reasonable results (see Figure 5). Additionally, we follow DSNeRF [60] to make a COLMAP depth supervision:

$$\mathcal{L}_d = w * (D - D_{gt})^2, \tag{11}$$

here, w is the weights calculated pixel-wise from COLMAP's projection error, and D_{gt} is the ground truth depth from COLMAP. As COLMAP can only provide discrete depth (actually, only key points have depth), we multiply a weights mask $\mathcal{M}$ to $\mathcal{L}_d$ in which pixels without GT depth will set to 0, and others to w. Figure 5 shows the performance of the added depth supervision.

To summarize, we rewrite the color loss used in Gaussian Splatting Equation 3 for clarity as:

$$\mathcal{L}_c = (1 - \lambda)\mathcal{L}_1(C, C_{gt}) + \lambda \mathcal{L}_{D-SSIM}(C, C_{gt}), \tag{12}$$

where C is the rendered 2D image calculated by Equation 2 and C_{gt} is the ground truth color. And our total loss L for decouplable reconstruction is:

$$\mathcal{L} = \lambda_1 \mathcal{L}_c + \lambda_2 \mathcal{L}_s + \lambda_3 \mathcal{L}_d. \tag{13}$$

λ_1, λ_2, and λ_3 are hyper-parameters for each loss's weight. To maintain vanilla Gaussian Splatting's fantastic speed, we explicitly derive gradients for s and d and integrate them into the CUDA implementation as introduced in subsection IV-C.

C. Gradients Calculation

Recall the rasterization and optimization process in Gaussian Splatting involving input 3D Gaussian center coordinates p_{3d}, color (spherical harmonic representation) [61] h, opacity o, scale and rotation (quaternion representation) matrix l and q. We add semantic s to the input. Our goal is to calculate the gradient of each input regarding loss function $\mathcal{L} = \mathcal{L}_{img} + \mathcal{L}_{mask} + \mathcal{L}_{depth}$ (For clarity, we rewrite the loss function in the perspective of the model outputs img, $mask$, and $depth$).

The backward process is realized by reversing the forward process step by step. For the top-level loss function $\mathcal{L}$, gradients $\partial\mathcal{L}_{img}/\partial img$, $\partial\mathcal{L}_{mask}/\partial mask$, and $\partial\mathcal{L}_{depth}/\partial depth$ are calculated by PyTorch.

img is calculated by α blending in Equation 2. Thus, we can calculate $\partial\mathcal{L}_{img}/\partial c = (\partial\mathcal{L}_{img}/\partial img) * (\partial img/\partial c)$ and $\partial\mathcal{L}_{img}/\partial alpha = (\partial\mathcal{L}_{img}/\partial img) * (\partial img/\partial \alpha)$ using the chain rule, respectively. As the calculation of $mask$ and $depth$ share a similar process to α blending in Equation 2, we can get $\partial\mathcal{L}_{mask}/\partial\hat{s}$, $\partial\mathcal{L}_{mask}/\partial\alpha$, $\partial\mathcal{L}_{depth}/\partial\hat{d}$, and $\partial\mathcal{L}_{depth}/\partial\alpha$. Here, $\hat{s}$ represents the 3D semantic value after our differentiable binarization of s, and $\hat{d}$ represents the intermediate variable depth for each Gaussian. As $\mathcal{L}_{img}$, $\mathcal{L}_{mask}$ and $\mathcal{L}_{depth}$ only relates to img, $mask$, and $depth$, respectively, their crossing terms like $\partial\mathcal{L}_{img}/\partial mask$ will always be zero.

In the forward pass, α is gained by multiplying opacity o with the result of some calculations to (p_{3d}, l, q). Here, the calculation represents projecting the 3D Gaussian to a 2D plane and calculating its exponential falloff from the mean [55]. For opacity o, it only affects α, and α performs the same blending process Equation 2 3 times repeatedly for img, $mask$, and $depth$. Therefore, the gradients for opacity o is:

$$\frac{\partial\mathcal{L}}{\partial o} = (\frac{\partial\mathcal{L}_{img}}{\partial\alpha} + \frac{\partial\mathcal{L}_{mask}}{\partial\alpha} + \frac{\partial\mathcal{L}_{depth}}{\partial\alpha}) * \frac{\partial\alpha}{\partial o}. \tag{14}$$

Following the original implementation, we can get $\partial\alpha/\partial o$.

The spherical harmonic representation h converts to Gaussian color c then blends to the output img by Equation 2. We can see that h only affects img. Therefore, its gradients will not change compared to the original one. $\partial c/\partial h$ is the same with the original implementation.

$$\frac{\partial\mathcal{L}}{\partial h} = \frac{\partial\mathcal{L}_{img}}{\partial c} * \frac{\partial c}{\partial h}. \tag{15}$$

Similar to h, the gradients of our added semantic attribute s can be calculated by:

$$\frac{\partial\mathcal{L}}{\partial s} = \frac{\partial\mathcal{L}_{mask}}{\partial\hat{s}} * \frac{\partial\hat{s}}{\partial s}. \tag{16}$$

Recall that $\hat{s}$ is converted from s by Sigmoid function f_{sig} and our differentiable binarization function f_{db}:

$$\hat{s} = f_{db}(f_{sig}(s)), \text{where } f_{db}(x) = f_{sig}(k(x-t)). \tag{17}$$

In this case, $\partial\hat{s}/\partial s$ is trivial as:

$$\frac{\partial\hat{s}}{\partial s} = \frac{\partial f_{db}}{\partial f_{sig}(s)} * \frac{\partial f_{sig}(s)}{\partial s}. \tag{18}$$

Here, $f'_{sig}(x) = f_{sig}(x)(1 - f_{sig}(x))$ and $f'_{db}(x) = kf_{db}(x)(1 - f_{db}(x))$.

For $\hat{d}$, this is slightly different from the above as we do not create a new input but use the projected z value as the Gaussian depth directly. p_{3d} will be affected as we calculate depth $\hat{d}$ as:

$$p_{view} = p_{3d} \odot V_{4\times4}, \tag{19}$$

here, V is the 4×4 view matrix and we get $\hat{d} = p_{view}.z$. Naturally, we get $\partial\hat{d}/\partial p_{3d} = \{V_{02}, V_{12}, V_{22}\}$.

Note that p_{3d}, l and q all contribute to α. Additionally, p_{3d} contributes to $\hat{d}$ and h as $h \rightarrow c$ involving p_{3d} [61]. We can calculate Gaussian's mean position p_{3d}, scale matrix l and rotation matrix q's gradiens by:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial p_{3d}} &= (\frac{\partial\mathcal{L}_{img}}{\partial\alpha} * \frac{\partial\alpha}{\partial p_{3d}} + \frac{\partial\mathcal{L}_{img}}{\partial c} * \frac{\partial c}{\partial p_{3d}}) \\ &+ \frac{\partial\mathcal{L}_{mask}}{\partial\alpha} * \frac{\partial\alpha}{\partial p_{3d}} \\ &+ (\frac{\partial\mathcal{L}_{depth}}{\partial\alpha} * \frac{\partial\alpha}{\partial p_{3d}} + \frac{\partial\mathcal{L}_{depth}}{\partial\hat{d}} * \frac{\partial\hat{d}}{\partial p_{3d}}), \\ \frac{\partial\mathcal{L}}{\partial l} &= (\frac{\partial\mathcal{L}_{img}}{\partial\alpha} + \frac{\partial\mathcal{L}_{mask}}{\partial\alpha} + \frac{\partial\mathcal{L}_{depth}}{\partial\alpha}) * \frac{\partial\alpha}{\partial l}, \\ \frac{\partial\mathcal{L}}{\partial q} &= (\frac{\partial\mathcal{L}_{img}}{\partial\alpha} + \frac{\partial\mathcal{L}_{mask}}{\partial\alpha} + \frac{\partial\mathcal{L}_{depth}}{\partial\alpha}) * \frac{\partial\alpha}{\partial q}.\end{aligned} \tag{20}$$

$\partial\alpha/\partial p_{3d}$, $\partial\alpha/\partial l$ and $\partial\alpha/\partial q$ are the same with the original Gaussian Splatting.

We have now explained the gradients of all inputs for our decouplable Gaussian Splatting method.

D. Stage III: Artifact Mask Calculation

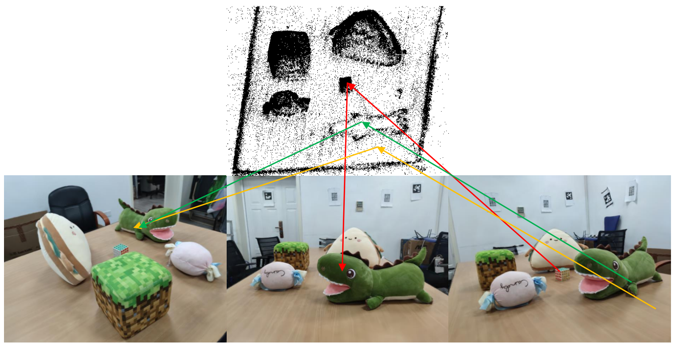

Fig. 6. Suppose the bottom right view is the source while the left and middle are targets. The dragon is the desired inpainting region with masks omitted. As the red line shows, we can find the non-mask pixel on the cube from the source view, which falls in the masked region of the middle view. The green line can serve as a reference. However, although the pixel indicated by the yellow line can also be found in the masked region of the left view, it cannot be a reference as its depth in the left view misaligns with the background depth. The green line indicates the actual pixel we need, which is, unfortunately, covered by all views.

With a decoupable Gaussian field, we can get raw inpainting results by only rasterizing the background Gaussians. Unsurprisingly, the raw results show "black holes" like ObjSDF and ObjectNeRF, as we had not considered background texture before. Note that the "black hole" artifacts mean these pixels are unseen from all available views. Thus, the reconstruction method gives it the default background value as black. In other words, if the masked pixel in target view i can be seen from another source view j, the 3D model can learn this pixel value in target view i regarding source view j, and the "black hole" artifact will not happen to this pixel. Formally, we project pixels in the non-mask region from all views to 3D space to get 3D points collection $\mathcal{P}^3 = \{P_i^3\}_{i=1}^n$ using Equation 4, and depth can be gained from Equation 10 as described in Sec IV-B. To continue, we traverse the image set $\mathcal{I} = \{I_i\}_{i=1}^n$ one by one and mark each image as the target view. For each target view I_i, we traverse $\mathcal{P}^3 = \{P^3_j\}^n_{j=1}$ to project P^3_j to I_i with Equation 5 and Equation 9 to get the 2D projected pixel coordinates and depth, respectively. Suppose the projected pixel is in the mask area of the target view I_i, and the corresponding projected depth is under a certain threshold ε compared to the background depth at this pixel. In that case, we can confirm that the pixel from the source view j can be the reference of its corresponding pixel in the target view i. The background depth can be obtained with Equation 10 by considering only the background Gaussians decided by s described in Sec IV-B. After the loop has been completed, the pixels left behind with no reference pixels will form the "black hole" region. In Figure 6, we draw a toy example to show how to find the artifact mask, and our artifact mask calculation algorithm is summarized as algorithm 2. The threshold ε is a fine-tunable hyper-parameter. We do not demand exact depth matching as depth from Gaussian Splatting and points projection can accumulate errors.

Algorithm 2: Artifact Mask Calculation

```
Data: segmentation maps $\mathcal{S} = \{S_i\}^n_{i=1}$, depth maps
      $\mathcal{D} = \{D_i\}^n_{i=1}$ and background depth maps
      $\mathcal{D}^{bg} = \{D^{bg}_i\}^n_{i=1}$ from our decouplable
      Gaussian Splatting, mapping between 2D and
      3D by projection $f$, threshold $\varepsilon$.
Result: artifact mask $\mathcal{M} = \{M_i\}^n_{i=1}$.

Initialization: $\mathcal{P}^{3d}$;
for $i \leftarrow 1$ to $n$ do
    $\mathcal{P}^{3d}_i = f_{2d\rightarrow 3d}(\mathcal{P}^{2d}_i \notin S_i)$;
    $M_i = 1$;

for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $n$ do
        $P^{2d}_{ij} = f_{3d\rightarrow 2d}(\mathcal{P}^{3d}_i, j)$;
        if $P^{2d}_{ij} \in S_j$ then
            if $abs(D_j[P^{2d}_{ij}] - D^{bg}_j[P^{2d}_{ij}]) < \varepsilon$ then
                $M_j[P^{2d}_{ij}] = 0$;

return $\mathcal{M}$;
```

E. Stage IV: Refine with 2D Priors

We observe unclean contour if we apply the mask to 2D inpainting models [12], [13] directly as Figure 7 shows. Following the experience in [10], mask dilation with fine-tunable parameters, including dilate kernel size, dilate iterations, and contour scale size is applied to masks before 2D inpainting.

After we process the masks properly, we can feed them and the raw inpainting results to a 2D inpainting model [12], [13] to get background priors. Suppose the corrected 2D results are C'_{gt}. Besides, we can inpainting depth maps similarly to get D'_{gt}. Additionally, we follow SPIn-NeRF's [10] experience to add a perceptual loss [62] term $\mathcal{L}_p$ to the inpainting area to expect not totally the same reconstruction,

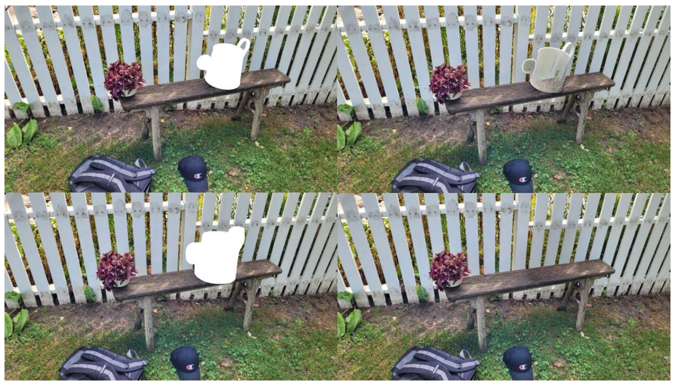
Fig. 7. In the above raw, an accurate mask makes unclean results, while a dilated mask in the second row with dilate kernel size and dilate iterations equal to 5 makes a clear prediction.

but perceptual similar RGBs. Combine above, we refine the decoupled Gaussian field from background Gaussians:

$$\mathcal{L} = \lambda_1 \mathcal{L}_c(C, C'_{gt}) + \lambda_2 \mathcal{L}_2(D, D'_{gt}) + \lambda_3 \mathcal{L}_p(C, C'_{gt}). \quad (21)$$

λ_1, λ_2 and λ_3 are hyper-parameters for each loss's weight. We do not need the semantic loss term during the refinement as all Gaussians are background. We use L_2 loss to optimize the depth map as we already obtain a complete depth image.

A calculation detail in our pipeline is that the point coordinates during calculation in 3D and 2D space are natively float numbers. However, the image plane uses int-pixel coordinates. To ensure the mapping is precise, we check the eight adjacent pixels to the floating point when we need to judge whether a point is in the mask region. Only all eight pixels in the mask indicate the floating point is in the mask region.

V. Experiments

A. Datasets

We select 37 scenes from various commonly used 3D datasets, including NeRF [1], IBRNet [63], LLFF [64], Mip-NeRF360 [65], and SPIn-NeRF [10]. We utilize pixel-wise accuracy (Acc) and intersection over union (IoU) to evaluate our masks. Since no ground truth is available for the 3D inpainting task, we conduct a qualitative analysis of our inpainting results.

B. implementation details

1) Mask Stage Generation: With interactively user annotation, our strategy can get raw segmentation maps for sparse views with the 2D segmentation method SAM [15], which accepts either points or box prompts. For the points prompt, the user will first annotate its desired region to inpainting by several points on an arbitrary view. The program will traverse each view and try to generate pseudo-point annotations by COLMAP [16] querying and camera pose projection for these images until it cannot create reasonable pseudo labels. Then, the user is requested to annotate a remaining arbitrary view, and the above process repeats until all views have been processed. The user can first apply a detection method like GroundingDINO [56] to get the target bounding box for the

table I
COMPARISON OF MASK BETWEEN OUR INTERACTIVE METHOD (SEC IV-A) AND SPIN-NERF [10]. THE FIRST ROW INDICATES THE SCENE NAME.

	1	2	3	4	7	9	10	12	trash	Mean	SPIn-NeRF [10]	book
acc↑	99.80	99.82	99.73	99.79	99.81	99.78	99.87	99.30	99.51	99.71↑	98.91	98.99
IoU↑	96.77	96.47	97.48	98.50	97.43	96.29	95.47	91.73	88.68	95.42↑	91.66	83.07

text prompt. Next, the user can spread text labels to all views or let the program generate pseudo-point annotations as we do for the points prompt. For the box prompts, the user first annotates an arbitrary view; the rest is the same as the points prompt.

2) *SAM Iterations:* When describing the method, we treat SAM [15] as an idea segmentation tool that gives the proper segmentation results as long as we input suitable prompts. However, in practice, we should pay attention to some details. As Figure 8 shows, SAM can produce zigzag prediction if we apply it directly. However, the artifact can be alleviated by passing the output back to SAM to make a new prediction with several iterations.

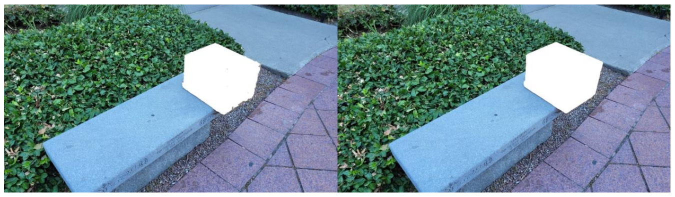

Fig. 8. White region is the SAM predict mask. The left is without iteration, while the right is iterating SAM output five times. We can see the zigzag on the left disappear on the right.

3) *Annotation Points Number:* It is expected that more points indicate the desired region more precisely. Strangely, when the number of annotation points exceeds a specific limit, SAM will treat the unmarked part as the mask as Figure 9 shows. Two solutions are proposed to address this issue. One can either randomly choose some points as SAM input if the pseudo points number is too large or filter the pseudo points by selecting the nearest to the user annotation. We suggest choosing the way with better results in practical usage as both methods are unsuitable for all scenes. The chosen points can be biased, thus making the partial mask.

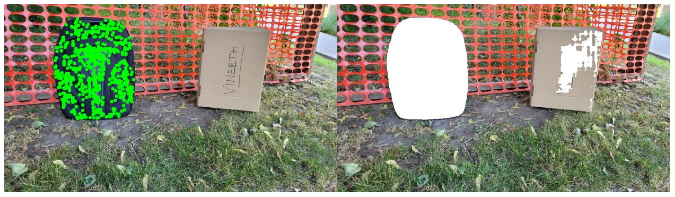

Fig. 9. Green points are generated pseudo labels. We can see that even though there are no points on the box, SAM makes the wrong prediction.

4) *Mapping Method:* As described before, either COLMAP query (key points) or projection can map points between 2D and 3D space. The COLMAP query is accurate, but it may not be enough to cover all the views. Projection can produce more correspondence, but 3D points outside the mask can be mapped to the mask region. In this case, projection can provide an upper bound for the 3D points as 3D points found by projection must contain the 3D points corresponding to the desired 2D region. At the same time, the COLMAP query provides a one-to-one mapping. In practical usage, we find that using COLMAP query only is enough for 360-degree scenes and usually even found more pseudo points as described in Sec V-B3. In this case, we apply projection to constrain that the points found by querying must be in the points set defined by projection from user annotation. On the other hand, projection is likely to ensure precise mapping for forward-facing scenes with small camera motion. Thus, our algorithm 1 can be accelerated by directly using the initialized 3D points as the final 3D foreground and creating 2D pseudo annotations by projection.

5) *Prompt Type:* While users can use points, text, or box prompts to get the 2D segmentation results, we suggest using the one that can produce the best results as none of the above choices is suitable for versatile 3D scenes, as Figure 10 shows.

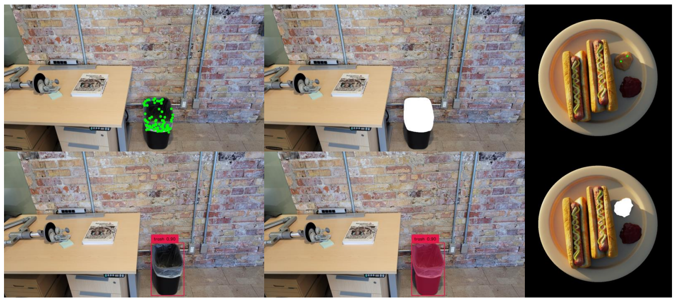

Fig. 10. In the left scene, it is hard for points prompt due to the lack of key points in the trash bin region. In contrast, the text prompt (spread text label to all views) can produce satisfactory results. In the right scene, it is natural for the points prompt to locate the desired region, but it is hard for the text to describe the target.

C. Mask Generation

Table I shows that our mask generation method is better than SPIn-NeRF considering accuracy and IoU. We omit the "book" scene from the average calculation as we find the ground truth of this scene is not accurate. The low metrics for "trash" are because there are nearly no key points at the trash bin's lower half region (see Figure 11). This also reveals the bottleneck of our mask generation method mentioned in Sec IV-A. However, we may get accurate masks if we apply text prompts to challenging scenes with point prompts for all views. Another concern is the human labor needed at this stage. Most scenes with dense camera trajectories can automatically annotate all the views except for the initial annotation. Only scenes from the NeRF Synthetic dataset have views where the desired foreground region can be unseen, which needs human judgment. However, human annotations will not exceed 10% for these scenes.

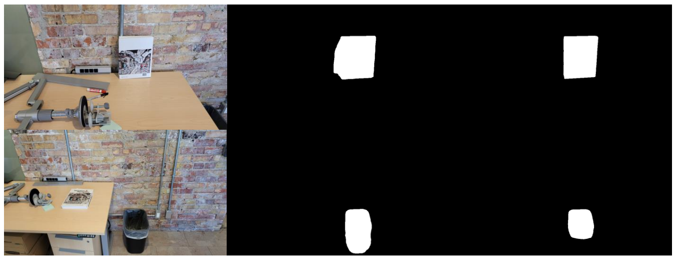
Fig. 11. From the left to right are the images, GT masks, and our results. Upper is the "book" scene, and lower is the "trash" scene.

D. Decouplable Reconstruction

First, as mentioned in Sec V-C, masks generated by Sec IV-A can be inaccurate sometimes if key points are inadequate in the desired region or SAM [15] cannot produce an accurate mask with unbalanced pseudo annotation. However, Figure 12 shows that our decouplable reconstruction can repair the partial mask after training.

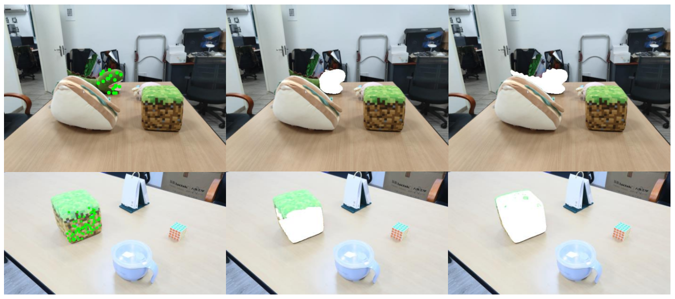
Fig. 12. From left to right are the image with pseudo annotations, mask predicted by SAM, and mask rendered by our decouplable Gaussian field. We can see that even if SAM generates a partial mask with unbalanced point annotations, it can be repaired in the later reconstruction stage. Though the result of the second row is imperfect, it can be post-processed with dilation.

Second, our depth supervision can regularize the scene with a large region covered by the same texture, like walls or mirrors, as an additional benefit besides calculating the artifact mask. Figure 13 illustrates this phenomenon. This is perhaps because depth supervision adds geometry information to Gaussian Splatting to avoid the ambiguity of a large, continuous, smooth region.

Finally, experiments show that our addition to the vanilla Gaussian Splatting can be trained without an apparent speed drop. Table III shows an approximate time analysis between our method and vanilla Gaussian Splatting. As the time for other stages, the interactive mask generation part usually takes below 1 second per image (it varies from different resolutions), and the artifact mask calculation usually takes 2 seconds per image. The compassion of rendering quality between our decouplable Gaussian Field and the original one is shown in Table II.

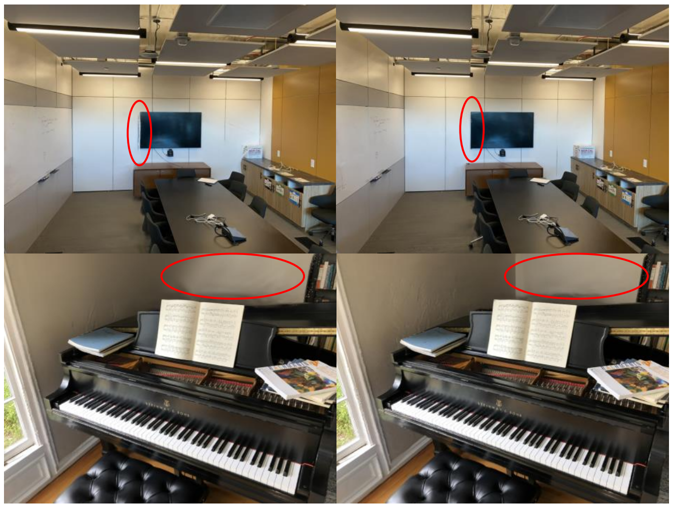
Fig. 13. Left is the results from original Gaussian Splatting directly, and right is the results of our method. We can see the left edge of the TV on the wall warped, and the circled wall area is blurred in the original implementation. With our depth supervision, both situations have improved.

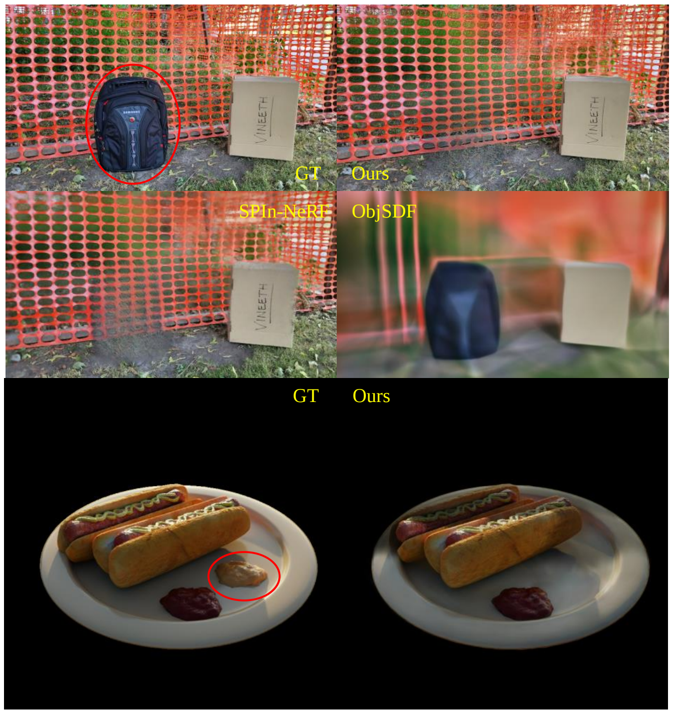

Fig. 14. In the above scene, though SPIn-NeRF can give a result, our rendering quality is higher. ObjSDF fails as it cannot converge on this scene. For the below scene, even generating masks is a big problem for the current methods. Though we provide masks for SPIn-NeRF and ObjSDF, they fail due to unstable training.

table II
Comparison of our method and the original Gaussian Splatting. The row name stands for the scene name.

	our_desk_2		our_desk_1		1		2		3		4	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	33.21	33.53	27.29	28.95	26.24	26.90	23.66	23.21	22.05	21.54	29.23	29.22
SSIM↑	0.961	0.962	0.926	0.935	0.825	0.832	0.743	0.666	0.813	0.796	0.857	0.852
LPIPS↓	0.114	0.113	0.278	0.264	0.159	0.155	0.236	0.320	0.132	0.155	0.136	0.153
	7		9		10		12		book		trash	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	27.62	27.79	25.88	26.05	24.31	24.24	20.90	21.40	29.69	29.80	29.11	28.79
SSIM↑	0.863	0.866	0.731	0.706	0.854	0.852	0.754	0.761	0.917	0.918	0.904	0.901
LPIPS↓	0.150	0.148	0.237	0.272	0.142	0.146	0.183	0.180	0.146	0.151	0.107	0.111
	ficus		hotdog		lego		materials		ship		pinecone	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	27.45	31.50	38.33	38.33	32.58	35.01	32.01	32.99	30.17	31.89	26.43	26.78
SSIM↑	0.958	0.980	0.990	0.990	0.976	0.987	0.976	0.979	0.906	0.916	0.874	0.876
LPIPS↓	0.052	0.029	0.026	0.026	0.037	0.024	0.052	0.049	0.133	0.127	0.138	0.137
	vasedeck		bonsai		garden		kitchen		data5_piano		flower	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	27.49	28.22	32.24	33.85	29.60	30.22	33.46	34.13	28.77	28.81	31.03	31.41
SSIM↑	0.881	0.888	0.961	0.967	0.912	0.919	0.965	0.968	0.911	0.909	0.927	0.930
LPIPS↓	0.154	0.147	0.082	0.072	0.086	0.081	0.050	0.047	0.156	0.167	0.083	0.080
	fortress		horns		orchids		room		trex		qq3	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	34.03	34.33	29.58	28.34	25.84	26.06	36.52	33.23	30.13	27.34	31.83	29.98
SSIM↑	0.949	0.952	0.934	0.919	0.903	0.904	0.977	0.966	0.953	0.933	0.930	0.908
LPIPS↓	0.074	0.073	0.096	0.118	0.093	0.091	0.060	0.090	0.082	0.107	0.126	0.165
	qq6		qq10		qq11		qq13		qq16		qq17	
	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori	ours	ori
PSNR↑	34.48	34.23	26.25	27.38	24.97	23.57	34.27	34.09	28.73	29.00	35.73	36.02
SSIM↑	0.963	0.960	0.895	0.895	0.837	0.766	0.941	0.935	0.930	0.923	0.965	0.967
LPIPS↓	0.087	0.096	0.145	0.153	0.179	0.236	0.083	0.094	0.093	0.117	0.054	0.057
	qq21		mean									
	ours	ori	ours	ori								
PSNR↑	39.59	39.96	29.75↓	29.95								
SSIM↑	0.980	0.982	0.9065↑	0.9017								
LPIPS↓	0.030	0.028	0.1154↓	0.1238								

table III
Approximate time comparison between ours and the vanilla version. As time varies for different scenes, we provide data from the same scene with RTX 4090 GPU.

	Gaussian Splatting	ours
training time (30000 steps)	7m0s	7m3s
refine time (2000 steps)	/	1m5s

E. *Refined inpainting*

Figure 14 shows examples where the previous methods fail or perform poorly, but our method performs robustly or better. As mentioned in the main paper, ObjSDF [8] may not converge on forward-facing scenes as its backbone VolSDF [66] fails. Another severe problem is the training time. VolSDF default is 2000 epochs, which requires around one day, while ObjSDF default is 10000 epochs, which takes up to 5 days (SPIn-NeRF is based on DSNeRF [60] and instantNGP [37], which takes nearly an hour, but still slow compared to our Gaussian field). GaussianEditor [67] also allows deletion but focuses on text prompt-driven editing with flexible functionality. It does not consider 3D geometry, so it may get stuck at the segmentation stage when a scene has several occluded objects, as shown in Figure 1. As for metrics, indeed, SPIn-NeRF provides ground truth images. However, according to [10], [14] and the experiment results, the PSNR value is usually below 15 with no meaning. This is because the inpainting results are still far from the ground truth. We show part of our inpainting results in Figure 15.

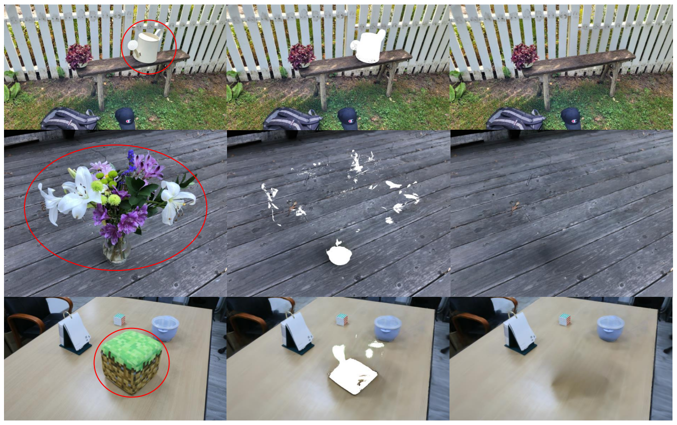
Fig. 15. From left to right are GT images, raw inpainting results with artifacts masked, and the final inpainting results with our method.

VI. Conclusion

This paper proposes a novel 3D inpainting pipeline with better performance than previous methods. However, artifacts still exist, especially for forward scenes where we cannot acquire 3D reference pixels and errors accumulated in our pipeline.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, 2020, pp. 405–421. [Online]. Available: https://doi.org/10.1007/978-3-030-58452-8_24

[2] Y. Peng, Y. Yan, S. Liu, Y. Cheng, S. Guan, B. Pan, G. Zhai, and X. Yang, "CageNeRF: Cage-based Neural Radiance Field for Generalized 3D Deformation and Animation," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 31 402–31 415. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/cb78e6b5246b03e0b82b4acc8b11cc21-Paper-Conference.pdf

[3] T. Xu and T. Harada, "Deforming Radiance Fields with Cages," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*, 2022, pp. 159–175. [Online]. Available: https://doi.org/10.1007/978-3-031-19827-4_10

[4] Y.-J. Yuan, Y.-T. Sun, Y.-K. Lai, Y. Ma, R. Jia, and L. Gao, "NeRF-Editing: Geometry Editing of Neural Radiance Fields," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 18 332–18 343.

[5] F. Xiang, Z. Xu, M. Hašan, Y. Hold-Geoffroy, K. Sunkavalli, and H. Su, "NeuTex: Neural Texture Mapping for Volumetric Neural Rendering," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 7115–7124.

[6] B. Yang, C. Bao, J. Zeng, H. Bao, Y. Zhang, Z. Cui, and G. Zhang, "NeuMesh: Learning Disentangled Neural Mesh-Based Implicit Field for Geometry and Texture Editing," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVI*, 2022, pp. 597–614. [Online]. Available: https://doi.org/10.1007/978-3-031-19787-1_34

[7] B. Yang, Y. Zhang, Y. Xu, Y. Li, H. Zhou, H. Bao, G. Zhang, and Z. Cui, "Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 13 759–13 768.

[8] Q. Wu, X. Liu, Y. Chen, K. Li, C. Zheng, J. Cai, and J. Zheng, "Object-Compositional Neural Implicit Surfaces," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVII*, 2022, pp. 197–213. [Online]. Available: https://doi.org/10.1007/978-3-031-19812-0_12

[9] S. Weder, G. Garcia-Hernando, A. Monszpart, M. Pollefeys, G. Brostow, M. Firman, and S. Vicente, "Removing Objects From Neural Radiance Fields," 2022. [Online]. Available: http://arxiv.org/abs/2212.11966

[10] A. Mirzaei, T. Aumentado-Armstrong, K. G. Derpanis, J. Kelly, M. A. Brubaker, I. Gilitschenski, and A. Levinshtein, "SPIn-NeRF: Multiview Segmentation and Perceptual Inpainting with Neural Radiance Fields," 2023. [Online]. Available: http://arxiv.org/abs/2211.12254

[11] R. Goel, D. Sirikonda, S. Saini, and P. J. Narayanan, "Interactive Segmentation of Radiance Fields," 2023. [Online]. Available: http://arxiv.org/abs/2212.13545

[12] L. Zhang and M. Agrawala, "Adding Conditional Control to Text-to-Image Diffusion Models," 2023. [Online]. Available: http://arxiv.org/abs/2302.05543

[13] R. Suvorov, E. Logacheva, A. Mashikhin, A. Remizova, A. Ashukha, A. Silvestrov, N. Kong, H. Goka, K. Park, and V. Lempitsky, "Resolution-robust Large Mask Inpainting with Fourier Convolutions," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022, pp. 3172–3182.

[14] Y. Yin, Z. Fu, F. Yang, and G. Lin, "Or-nerf: Object removing from 3d scenes guided by multiview segmentation with neural radiance fields," 2023.

[15] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment Anything," 2023. [Online]. Available: http://arxiv.org/abs/2304.02643

[16] J. L. Schönberger, T. Price, T. Sattler, J.-M. Frahm, and M. Pollefeys, "A vote-and-verify strategy for fast spatial verification in image retrieval," in *Asian Conference on Computer Vision (ACCV)*, 2016.

[17] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[18] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging Properties in Self-Supervised Vision Transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9630–9640.

[19] Y. Hao, Y. Liu, Z. Wu, L. Han, Y. Chen, G. Chen, L. Chu, S. Tang, Z. Yu, Z. Chen, and B. Lai, "EdgeFlow: Achieving Practical Interactive Segmentation with Edge-Guided Flow," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, pp. 1551–1560.

[20] L. Ke, M. Ye, M. Danelljan, Y. Liu, Y.-W. Tai, C.-K. Tang, and F. Yu, "Segment Anything in High Quality," 2023. [Online]. Available: http://arxiv.org/abs/2306.01567

[21] B. Li, K. Q. Weinberger, S. Belongie, V. Koltun, and R. Ranftl, "Language-driven Semantic Segmentation," in *ICLR*, 2022. [Online]. Available: https://openreview.net/forum?id=RriDjddCLN

[22] L. Qi, J. Kuen, T. Shen, J. Gu, W. Li, W. Guo, J. Jia, Z. Lin, and M.-H. Yang, "High Quality Entity Segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4047–4056. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2023/html/Qi_High_Quality_Entity_Segmentation_ICCV_2023_paper.html

[23] X. Wang, X. Zhang, Y. Cao, W. Wang, C. Shen, and T. Huang, "SegGPT: Segmenting Everything In Context," 2023. [Online]. Available: http://arxiv.org/abs/2304.03284

[24] Q. Wang, Y.-Y. Chang, R. Cai, Z. Li, B. Hariharan, A. Holynski, and N. Snavely, "Tracking Everything Everywhere All at Once," 2023. [Online]. Available: http://arxiv.org/abs/2306.05422

[25] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C. S. Hong, "Faster Segment Anything: Towards Lightweight SAM for Mobile Applications," 2023. [Online]. Available: http://arxiv.org/abs/2306.14289

[26] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang, "Fast Segment Anything," 2023. [Online]. Available: http://arxiv.org/abs/2306.12156

[27] W. Liu, G. Lin, T. Zhang, and Z. Liu, "Guided co-segmentation network for fast video object segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 4, pp. 1607–1617, 2021.

[28] X. Sun, C. Chen, X. Wang, J. Dong, H. Zhou, and S. Chen, "Gaussian dynamic convolution for efficient single-image segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 5, pp. 2937–2948, 2022.

[29] H. Zhou, L. Yang, X. Xie, and J. Lai, "Selective intra-image similarity for personalized fixation-based object segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 11, pp. 7910–7923, 2022.

[30] J. Chen, W. Lu, Y. Li, L. Shen, and J. Duan, "Adversarial learning of object-aware activation map for weakly-supervised semantic segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 8, pp. 3935–3946, 2023.

[31] Y. Tang, T. Chen, X. Jiang, Y. Yao, G.-S. Xie, and H.-T. Shen, "Holistic prototype attention network for few-shot video object segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 8, pp. 6699–6709, 2024.

[32] Z. Fan, P. Wang, Y. Jiang, X. Gong, D. Xu, and Z. Wang, "NeRF-SOS: Any-View Self-supervised Object Segmentation on Complex Scenes," 2022. [Online]. Available: http://arxiv.org/abs/2209.08776

[33] X. Liu, J. Chen, H. Yu, Y.-W. Tai, and C.-K. Tang, "Unsupervised Multi-View Object Segmentation Using Radiance Field Propagation," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 17 730–17 743. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/70de9e3948645a1be2de657f14d85c6d-Paper-Conference.pdf

[34] M. Wallingford, A. Kusupati, A. Fang, V. Ramanujan, A. Kembhavi, R. Mottaghi, and A. Farhadi, "Neural Radiance Field Codebooks," in *ICLR*, 2023. [Online]. Available: https://openreview.net/forum?id=mX56bKDybu5

[35] X. Fu, S. Zhang, T. Chen, Y. Lu, L. Zhu, X. Zhou, A. Geiger, and Y. Liao, "Panoptic NeRF: 3D-to-2D Label Transfer for Panoptic Urban Scene Segmentation," 2022. [Online]. Available: http://arxiv.org/abs/2203.15224

[36] S. Zhi, T. Laidlow, S. Leutenegger, and A. J. Davison, "In-Place Scene Labelling and Understanding with Implicit Scene Representation," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 15 818–15 827.

[37] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 102:1–102:15, 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3528223.3530127

[38] C. Bao, Y. Zhang, B. Yang, T. Fan, Z. Yang, H. Bao, G. Zhang, and Z. Cui, "SINE: Semantic-driven Image-based NeRF Editing with Prior-guided Editing Field," 2023. [Online]. Available: http://arxiv.org/abs/2303.13277

[39] S. Benaim, F. Warburg, P. E. Christensen, and S. Belongie, "Volumetric Disentanglement for 3D Scene Manipulation," 2022. [Online]. Available: http://arxiv.org/abs/2206.02776

[40] A. Mikaeili, O. Perel, D. Cohen-Or, and A. Mahdavi-Amiri, "SKED: Sketch-guided Text-based 3D Editing," 2023. [Online]. Available: http://arxiv.org/abs/2303.10735

[41] E. Sella, G. Fiebelman, P. Hedman, and H. Averbuch-Elor, "Vox-E: Text-guided Voxel Editing of 3D Objects," 2023. [Online]. Available: http://arxiv.org/abs/2303.12048

[42] Z. Chen, K. Yin, and S. Fidler, "AUV-Net: Learning Aligned UV Maps for Texture Transfer and Synthesis," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 1455–1464.

[43] K. Rematas, R. Martin-Brualla, and V. Ferrari, "Sharf: Shape-conditioned Radiance Fields from a Single View," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 8948–8958. [Online]. Available: https://proceedings.mlr.press/v139/rematas21a.html

[44] H.-X. Yu, L. Guibas, and J. Wu, "Unsupervised Discovery of Object Radiance Fields," in *ICLR*, 2022. [Online]. Available: https://openreview.net/forum?id=rwE8SshAlxw

[45] V. Lazova, V. Guzov, K. Olszewski, S. Tulyakov, and G. Pons-Moll, "Control-NeRF: Editable Feature Volumes for Scene Rendering and Manipulation," 2022. [Online]. Available: http://arxiv.org/abs/2204.10850

[46] S. Kobayashi, E. Matsumoto, and V. Sitzmann, "Decomposing NeRF for Editing via Feature Field Distillation," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 23 311–23 330. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/93f250215e4889119807b6fac3a57aec-Paper-Conference.pdf

[47] B. Wang, L. Chen, and B. Yang, "DM-NeRF: 3D Scene Geometry Decomposition and Manipulation from 2D Images," in *ICLR*, 2023. [Online]. Available: https://openreview.net/forum?id=C_PRLz8bEJx

[48] S. Liu, X. Zhang, Z. Zhang, R. Zhang, J.-Y. Zhu, and B. Russell, "Editing Conditional Radiance Fields," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5753–5763.

[49] J. Zhu, Y. Huo, Q. Ye, F. Luan, J. Li, D. Xi, L. Wang, R. Tang, W. Hua, H. Bao, and R. Wang, "I$2̂$-SDF: Intrinsic Indoor Scene Reconstruction and Editing via Raytracing in Neural SDFs," 2023. [Online]. Available: http://arxiv.org/abs/2303.07634

[50] W. Ye, S. Chen, C. Bao, H. Bao, M. Pollefeys, Z. Cui, and G. Zhang, "IntrinsicNeRF: Learning Intrinsic Neural Radiance Fields for Editable Novel View Synthesis," 2023. [Online]. Available: http://arxiv.org/abs/2210.00647

[51] A. Mirzaei, Y. Kant, J. Kelly, and I. Gilitschenski, "LaTeRF: Label and Text Driven Object Radiance Fields," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III*, 2022, pp. 20–36. [Online]. Available: https://doi.org/10.1007/978-3-031-20062-5_2

[52] Z. Kuang, F. Luan, S. Bi, Z. Shu, G. Wetzstein, and K. Sunkavalli, "PaletteNeRF: Palette-based Appearance Editing of Neural Radiance Fields," 2023. [Online]. Available: http://arxiv.org/abs/2212.10699

[53] V. Tschernezki, I. Laina, D. Larlus, and A. Vedaldi, "Neural Feature Fusion Fields: 3D Distillation of Self-Supervised 2D Image Representations," in *2022 International Conference on 3D Vision (3DV)*, 2022, pp. 443–453.

[54] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[55] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "EWA volume splatting," in *Proceedings Visualization, 2001. VIS '01.*, 2001, pp. 29–538.

[56] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499*, 2023.

[57] IDEA-Research, "Grounded-sam," https://github.com/IDEA-Research/Grounded-Segment-Anything, 2023.

[58] M. Liao, Z. Wan, C. Yao, K. Chen, and X. Bai, "Real-Time Scene Text Detection with Differentiable Binarization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 474–11 481, 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6812

[59] M. Liao, Z. Zou, Z. Wan, C. Yao, and X. Bai, "Real-Time Scene Text Detection With Differentiable Binarization and Adaptive Scale Fusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 919–931, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9726868

[60] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan, "Depth-supervised NeRF: Fewer Views and Faster Training for Free," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 12 872–12 881.

[61] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5752–5761.

[62] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.

[63] Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser, "Ibrnet: Learning multi-view image-based rendering," in *CVPR*, 2021.

[64] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines," *ACM Transactions on Graphics (TOG)*, 2019.

[65] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5460–5469.

[66] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume Rendering of Neural Implicit Surfaces," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds., vol. 34, 2021, pp. 4805–4815. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/25e2a30f44898b9f3e978b1786dcd85c-Paper.pdf

[67] Y. Chen, Z. Chen, C. Zhang, F. Wang, X. Yang, Y. Wang, Z. Cai, L. Yang, H. Liu, and G. Lin, "Gaussianeditor: Swift and controllable 3d editing with gaussian splatting," 2023.