

# GaussianEditor: Swift and Controllable 3D Editing with Gaussian Splatting



Yiwen Chen<sup>\*1,2</sup> Zilong Chen<sup>\*3,5</sup> Chi Zhang<sup>2</sup> Feng Wang<sup>3</sup> Xiaofeng Yang<sup>2</sup>  
Yikai Wang<sup>3</sup> Zhongang Cai<sup>4</sup> Lei Yang<sup>4</sup> Huaping Liu<sup>3</sup> Guosheng Lin<sup>\*\*1,2</sup>

<sup>1</sup>S-Lab, Nanyang Technological University

<sup>2</sup>School of Computer Science and Engineering, Nanyang Technological University

<sup>3</sup>Department of Computer Science and Technology, Tsinghua University

<sup>4</sup>SenseTime Research <sup>5</sup>ShengShu

<https://buaacyw.github.io/gaussian-editor>



Figure 1. **Results of GaussianEditor.** GaussianEditor offers swift, controllable, and versatile 3D editing. A single editing session only takes 5-10 minutes. Please note our precise editing control, where only the desired parts are modified. Taking the “*Make the grass on fire*” example from the first row of the figure, other objects in the scene such as the bench and tree remain unaffected.

## Abstract

3D editing plays a crucial role in many areas such as gaming and virtual reality. Traditional 3D editing methods, which rely on representations like meshes and point clouds, often fall short in realistically depicting complex scenes. On the other hand, methods based on implicit 3D representations, like Neural Radiance Field (NeRF), render complex scenes effectively but suffer from slow processing speeds and limited control over specific scene areas. In response to these challenges, our paper presents GaussianEditor, an innovative and efficient 3D editing algorithm based on Gaussian

Splatting (GS), a novel 3D representation. GaussianEditor enhances precision and control in editing through our proposed Gaussian semantic tracing, which traces the editing target throughout the training process. Additionally, we propose Hierarchical Gaussian splatting (HGS) to achieve stabilized and fine results under stochastic generative guidance from 2D diffusion models. We also develop editing strategies for efficient object removal and integration, a challenging task for existing methods. Our comprehensive experiments demonstrate GaussianEditor’s superior control, efficacy, and rapid performance, marking a significant advancement in 3D editing.



<sup>\*\*</sup> Corresponding author.

<sup>\*</sup> The first two authors contributed equally to this work.

## 1. Introduction

In the evolving field of computer vision, the development of user-friendly 3D representations and editing algorithms is a key objective. Such technologies are vital in various applications, ranging from digital gaming to the growing MetaVerse. Traditional 3D representations like meshes and point clouds have been preferred due to their interactive editing capabilities. However, these methods face challenges in accurately rendering complex 3D scenes.

The recent rise of implicit 3D representations, exemplified by the Neural Radiance Field (NeRF) [28], represents a paradigm shift in 3D scene rendering. NeRF's capacity for high-fidelity rendering, coupled with its implicit nature that offers significant expansibility, marks a substantial improvement over conventional approaches[2, 32, 55]. This dual advantage has placed a significant focus on the NeRF framework in 3D editing [12, 31, 45, 46, 57], establishing it as a foundational approach for a considerable duration. However, NeRF's reliance on high-dimensional multilayer perception (MLP) networks for scene data encoding presents limitations. It restricts direct modification of specific scene parts and complicates tasks, like inpainting and scene composition. This complexity extends to the training and rendering processes, hindering practical applications.

In light of these challenges, our research is focused on developing an advanced 3D editing algorithm. This algorithm aims for flexible and rapid editing of 3D scenes, integrating both implicit editing, like text-based editing, and explicit control, such as bounding box usage for specific area modifications. To achieve these goals, we choose Gaussian Splatting (GS) [15] for its real-time rendering and explicit point cloud-like representations.

However, editing Gaussian Splatting (GS) [15] faces distinct challenges. A primary issue is the absence of efficient methods to accurately identify target Gaussians, which is crucial for precise controllable editing. Moreover, it has been observed in [7, 44, 52] that optimizing Gaussian Splatting (GS) using highly random generative guidance like Score Distillation Sampling [36] poses significant challenges. One possible explanation is that, unlike implicit representations buffered by neural networks, GS is directly affected by the randomness in loss. Such direct exposure results in unstable updates, as the properties of Gaussians are directly changed during training. Besides, each training step of GS may involve updates to a vast number of Gaussian points. This process occurs without the moderating influence of neural network-style buffering mechanisms. As a result, the excessive fluidity of the 3D GS scene hinders its ability to converge to finely detailed results like implicit representations when trained with generative guidance.

To counter these issues, in this work, we propose GaussianEditor , a novel, swift, and highly controllable 3D editing algorithm for Gaussian Splatting. GaussianEditor can fulfill

various high-quality editing needs within minutes. A key feature of our method is the introduction of Gaussian semantic tracing, which enables precise control over Gaussian Splatting (GS). Gaussian semantic tracing consistently identifies the Gaussians requiring editing at every moment during training. This contrasts with traditional 3D editing methods that often depend on static 2D or 3D masks. Such masks become less effective as the geometries and appearances of 3D models evolve during training. Gaussian semantic tracing is achieved by unprojecting 2D segmentation masks into 3D Gaussians and assigning each Gaussian a semantic tag. As the Gaussians evolve during training, these semantic tags enable the tracking of the specific Gaussians targeted for editing. Our Gaussian tracing algorithm ensures that only the targeted areas are modified, enabling precise and controllable editing.

Additionally, to tackle the significant challenge of Gaussian Splatting (GS) struggling to fit fine results under highly random generative guidance, we propose a novel GS representation: hierarchical Gaussian splatting (HGS). In HGS, Gaussians are organized into generations based on their sequence in multiple densification processes during training. Gaussians formed in earlier densification stages are deemed older generations and are subject to stricter constraints, aimed at preserving their original state and thus reducing their mobility. Conversely, those formed in later stages are considered younger generations and are subjected to fewer or no constraints, allowing for more adaptability. HGS's design effectively moderates the fluidity of GS by imposing restrictions on older generations while preserving the flexibility of newer generations. This approach enables continuous optimization towards better outcomes, thereby simulating the buffering function achieved in implicit representations through neural networks. Our experiments also demonstrate that HGS is more adept at adapting to highly random generative guidance.

Finally, we have specifically designed a 3D inpainting algorithm for Gaussian Splatting (GS). As demonstrated in Fig. 1, we have successfully removed specific objects from scenes and seamlessly integrated new objects into designated areas. For object removal, we developed a specialized local repair algorithm that efficiently eliminates artifacts at the intersection of the object and the scene. For adding objects, we first request users to provide a prompt and a 2D inpainting mask for a particular view of the GS. Subsequently, we employ a 2D inpainting method to generate a single-view image of the object to be added. This image is then transformed into a coarse 3D mesh using image-to-3D conversion techniques. The 3D mesh is subsequently converted into the HGS representation and refined. Finally, this refined representation is concatenated into the original GS. The entire inpainting process described above is completed within 5 minutes.

GaussianEditor offers swift, controllable, and versatile 3D editing. A single editing session typically only takes 5-10 minutes, significantly faster than previous editing processes. Our contributions can be summarized in four aspects:

1. We have introduced Gaussian semantic tracing, enabling more detailed and effective editing control.
2. We propose Hierarchical Gaussian Splatting (HGS), a novel GS representation capable of converging more stably to refined results under highly random generative guidance.
3. We have specifically designed a 3D inpainting algorithm for Gaussian Splatting, which allows swift removal and addition objects.
4. Extensive experiments demonstrate that our method surpasses previous 3D editing methods in terms of effectiveness, speed, and controllability.

## 2. Related Works

### 2.1. 3D Representations

Various 3D representations have been proposed to address diverse 3D tasks. The groundbreaking work, Neural Radiance Fields (NeRF) [28], employs volumetric rendering and has gained popularity for enabling 3D optimization with only 2D supervision. However, optimizing NeRF can be time-consuming, despite its wide usage in 3D reconstruction [3, 6, 13, 21] and generation [22, 35] tasks.

While efforts have been made to accelerate NeRF training [29, 40], these approaches primarily focus on the reconstruction setting, leaving the generation setting less optimized. The common technique of spatial pruning does not effectively speed up the generation setting.

Recently, 3D Gaussian splatting [15] has emerged as an alternative 3D representation to NeRF, showcasing impressive quality and speed in 3D and 4D reconstruction tasks [15, 26, 47, 50, 51]. It has also attracted considerable research interest in the field of generation [7, 44, 52]. Its efficient differentiable rendering implementation and model design facilitate fast training without the need for spatial pruning.

In this work, we pioneer the adaptation of 3D Gaussian splatting to 3D editing tasks, aiming to achieve swift and controllable 3D editing, harnessing the advantages of this representation for the first time in this context.

### 2.2. 3D Editing

Editing neural fields is inherently challenging due to the intricate interplay between their shape and appearance. EditNeRF [24] stands as a pioneering work in this domain, as they edit both the shape and color of neural fields by conditioning them on latent codes. Additionally, some works [1, 10, 45, 46] leverage CLIP models to facilitate editing through the use of text prompts or reference images.

Another line of research focuses on predefined template models or skeletons to support actions like re-posing or re-rendering within specific categories [30, 33]. Geometry-based methods [20, 48, 49, 54] translate neural fields into meshes and synchronize mesh deformation with implicit fields. Additionally, 3D editing techniques involve combining 2D image manipulation, such as inpainting, with neural fields training [19, 23].

Concurrent works [31, 57] leverage static 2D and 3D masks to constrain the edit area of NeRF. However, these approaches have their limitations because the training of 3D models is a dynamic process, and static masks cannot effectively constrain it. In contrast, our research employs Gaussian semantic tracing to track the target Gaussian throughout the entire training process.

## 3. Preliminary

### 3.1. 3D Gaussian Splatting

GS (Gaussian Splatting) [15] represents an explicit 3D scene using point clouds, where Gaussians are employed to depict the scene’s structure. In this representation, every Gaussian is defined by a center point, denoted as  $x$ , and a covariance matrix  $\Sigma$ . The center point  $x$  is commonly known as the Gaussian’s mean value:

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

The covariance matrix  $\Sigma$  can be decomposed into a rotation matrix  $\mathbf{R}$  and a scaling matrix  $\mathbf{S}$  for differentiable optimization:

$$\Sigma = \mathbf{R} \mathbf{S}^T \mathbf{R}^T, \quad (2)$$

the calculation of gradient flow is detailed in [15].

For rendering new viewpoints, the method of splatting, as described in [53], is utilized for positioning the Gaussians on the camera planes. This technique, originally presented in [58], involves a viewing transformation denoted by  $W$  and the Jacobian  $J$  of the affine approximation of the projective transformation. Using these, the covariance matrix  $\Sigma'$  in camera coordinates is determined as follows:

$$\Sigma' = JW\Sigma W^T J^T. \quad (3)$$

To summarize, each Gaussian point in the model is characterized by a set of attributes: its position, denoted as  $x \in \mathbb{R}^3$ , its color represented by spherical harmonics coefficients  $c \in \mathbb{R}^k$  (where  $k$  indicates the degrees of freedom), its opacity  $\alpha \in \mathbb{R}$ , a rotation quaternion  $q \in \mathbb{R}^4$ , and a scaling factor  $s \in \mathbb{R}^3$ . Particularly, for every pixel, the color and opacity of all Gaussians are calculated based on the Gaussian’s representation as described in Eq. 1. The blending process of  $N$  ordered points overlapping a pixel follows a specific



**Figure 2. Illustration of Gaussian semantic tracing.** **Prompt:** **Turn him into an old lady.** The red mask in the images represents the projection of the Gaussians that will be updated and densified. The dynamic change of the masked area during the training process, as driven by the updating of Gaussians, ensures consistent effectiveness throughout the training duration. Despite starting with potentially inaccurate segmentation masks due to 2D segmentation errors, Gaussian semantic tracing still guarantees high-quality editing results.

formula:

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (4)$$

where  $c_i$  and  $\alpha_i$  signify the color and density of a given point respectively. These values are determined by a Gaussian with a covariance matrix  $\Sigma$ , which is then scaled by optimizable per-point opacity and spherical harmonics (SH) color coefficients.

### 3.2. Diffusion-based Editing Guidance

Recent advancements have seen numerous works elevating 2D diffusion processes to 3D, applying these processes extensively in the realm of 3D editing. Broadly, these works can be categorized into two types. The first type [8, 27, 31, 36, 42, 57], exemplified by Dreamfusion’s [36] introduction of SDS loss, involves feeding the noised rendering of the current 3D model, along with other conditions, into a 2D diffusion model [39]. The scores generated by the diffusion model then guide the direction of model updates. The second type [5, 12, 37, 43] focuses on conducting 2D editing based on given prompts for the multiview rendering of a 3D model. This approach creates a multi-view 2D image dataset, which is then utilized as a training target to provide guidance for the 3D model.

Our work centers on leveraging the exemplary properties of Gaussian Splatting’s explicit representation to enhance 3D editing. Consequently, we do not design specific editing guidance mechanisms but instead directly employ the guidance methods mentioned above. Both types of guidance can be applied in our method. For simplicity, we denote the guidance universally as  $D$ . Given the parameters of a 3D model,  $\Theta$ , along with the rendered camera pose  $p$  and prompt  $e$ , the editing loss from the 2D diffusion prior can be formulated as follows:

$$\mathcal{L}_{\text{Edit}} = D(\Theta; p, e) \quad (5)$$

## 4. Method

We define the task of 3D editing on Gaussian Splatting (GS) as follows: Given a prompt  $y$  and a 3D scene represented by 3D Gaussians, denoted by  $\Theta$ , where each  $\Theta_i = \{x_i, s_i, q_i, \alpha_i, c_i\}$  represents the parameters of the  $i$ -th Gaussian as detailed in Sec. 3.1, the objective is to achieve an edited 3D Gaussians, referred to as  $\Theta_y$ , that aligns with or adheres to the specifications of the prompt  $y$ .

We then introduce our novel framework for performing editing tasks on GS. We first introduce Gaussian semantic tracing in Sec. 4.1, along with a new representation method known as Hierarchical Gaussian Splatting (HGS) in Sec. 4.2. The GS semantic tracing enables precise segmentation and tracing within GS, facilitating controllable editing operations. Compared to the standard GS, the HGS representation demonstrates greater robustness against randomness in generative guidance and is more adept at accommodating a diverse range of editing scenarios. Additionally, we have specifically designed 3D inpainting for GS, which encompasses object removal and addition (Sec. 4.3).

### 4.1. Gaussian Semantic Tracing

Previous works [31, 57] in 3D editing usually utilize static 2D or 3D masks to apply loss only within the masked pixels, thus constraining the editing process to only edit the desired area. However, this method has limitations. As 3D representations dynamically change during training, static segmentation masks would become inaccurate or even ineffective. Furthermore, the use of static masks to control gradients in NeRF editing poses a significant limitation, as it confines the editing strictly within the masked area. This restriction prevents the edited content from naturally extending beyond the mask, thus ‘locking’ the content within a specified spatial boundary.

Even with the implementation of semantic NeRF [56], the gradient control is still only effective at the very beginning of the training since the ongoing updates to NeRF lead to a loss of accuracy in the semantic field.

To address the aforementioned issue, we have chosen Gaussian Splatting (GS) as our 3D representation due to its explicit nature. This allows us to directly assign semantic labels to each Gaussian point, thereby facilitating semantic tracing in 3D scenes.

Specifically, we enhance the 3D Gaussians  $\Theta$  by adding a new attribute  $m$ , where  $m_{ij}$  represents the semantic Gaussian mask for the  $i$ -th Gaussian point and the  $j$ -th semantic label. With this attribute, we can precisely control the editing process by selectively updating only the target 3D Gaussians. During the densification process, newly densified points inherit the semantic label of their parent point. This ensures that we have an accurate 3D semantic mask at every moment throughout the training process.

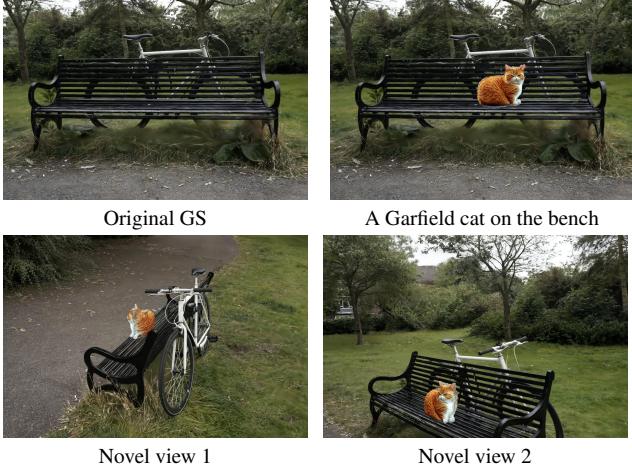


Figure 3. **3D inpainting for object incorporation.** GaussianEditor is capable of adding objects at specified locations in a scene, given a 2D inpainting mask and a text prompt from a single view. The whole process takes merely five minutes.

As illustrated in Fig. 2, Gaussian semantic tracing enables continuous tracking of each Gaussian’s categories during training, adjusting to their evolving properties and numbers. This feature is vital, as it permits selective application of gradients, densification and pruning of Gaussians linked to the specified category. Additionally, it facilitates training solely by rendering the target object, significantly speeding up the process in complex scenes. The semantic Gaussian mask  $m$  functions as a dynamic 3D segmentation mask, evolving with the training, allowing content to expand freely in space. This contrasts with NeRF, where content is restricted to a fixed spatial area.

Next, we discuss Gaussian Splatting unprojection, the method we propose to obtain semantic Gaussian mask  $m$ . For a set of 3D Gaussians  $\Theta$ , we render them from multiple viewpoints to generate a series of renderings  $\mathcal{I}$ . These renderings are then processed using 2D segmentation techniques [18] to obtain 2D segmentation masks  $\mathcal{M}$ , with each  $\mathcal{M}_j$ , representing the  $j$ -th semantic labels.

To obtain the semantic label for each Gaussian, we unproject the posed 2D semantic label back to the Gaussians with inverse rendering. Concretely, we maintain a weight and a counter for each Gaussian. For pixel  $p$  on the semantic maps, we unproject the semantic label back to the Gaussians that affects it by

$$w_i^j = \sum o_i(p) * T_i^j(p) * \mathcal{M}^j(p), \quad (6)$$

where  $w_i^j$  represents the weight of the  $i$ -th Gaussian for the  $j$ -th semantic label, while  $o_i(p)$ ,  $T_i^j(p)$ , and  $\mathcal{M}^j(p)$  denote the opacity, transmittance from pixel  $p$ , and semantic mask of pixel  $p$  for the  $i$ -th Gaussian, respectively. After updating all the Gaussian weights and counters, we determine whether a Gaussian belongs to the  $j$ -th semantic class



Figure 4. **3D inpainting for object removal.** Typically, removing the target object based on a Gaussian semantic mask generates artifacts at the interface between the target object and the scene. To address this, we generate a repainted image using a 2D inpainting method and employ Mean Squared Error (MSE) loss for supervision. The whole process takes merely two minutes.

based on whether its average weight exceeds a manually set threshold.

The entire labeling process is remarkably fast, typically taking less than a second. Once this semantic label assignment is completed, the entire Gaussian scene becomes parsed by us, making a variety of operations possible. These include manually changing colors, moving properties of a specific category, and deleting certain categories. Notably, 2D diffusion guidance often struggles to effectively edit small objects in complex scenes. Thanks to Gaussian semantic tracing, we can now render these small objects independently and input them into the 2D diffusion model, thereby achieving more precise supervision.

## 4.2. Hierarchical Gaussian Splatting

The effectiveness of vanilla GS [17] in reconstruction tasks lies in the high-quality initialization provided by point clouds derived from SfM [41], coupled with stable supervision from ground truth datasets.

However, the scenario changes in the field of generation. In previous work involving GS in text-to-3D and image-to-3D [7, 44, 52], GS has shown limitations when facing the randomness of generative guidance due to its nature as a point cloud-like representation. This instability in GS is mainly due to their direct exposure to the randomness of loss functions, unlike neural network-based implicit representations. GS models, which update a large number of Gaussian points each training step, lack the memorization and moderating ability of neural networks. This leads to erratic updates and prevents GS from achieving the detailed results seen in neural network-based implicit representations, as GS’s excessive fluidity hampers its convergence in generative training.

To address these challenges, we introduce Hierarchical Gaussian Splatting (HGS), a structured representation of GS that is more suitable for generative and editing scenarios. HGS categorizes GS into different generations based on the

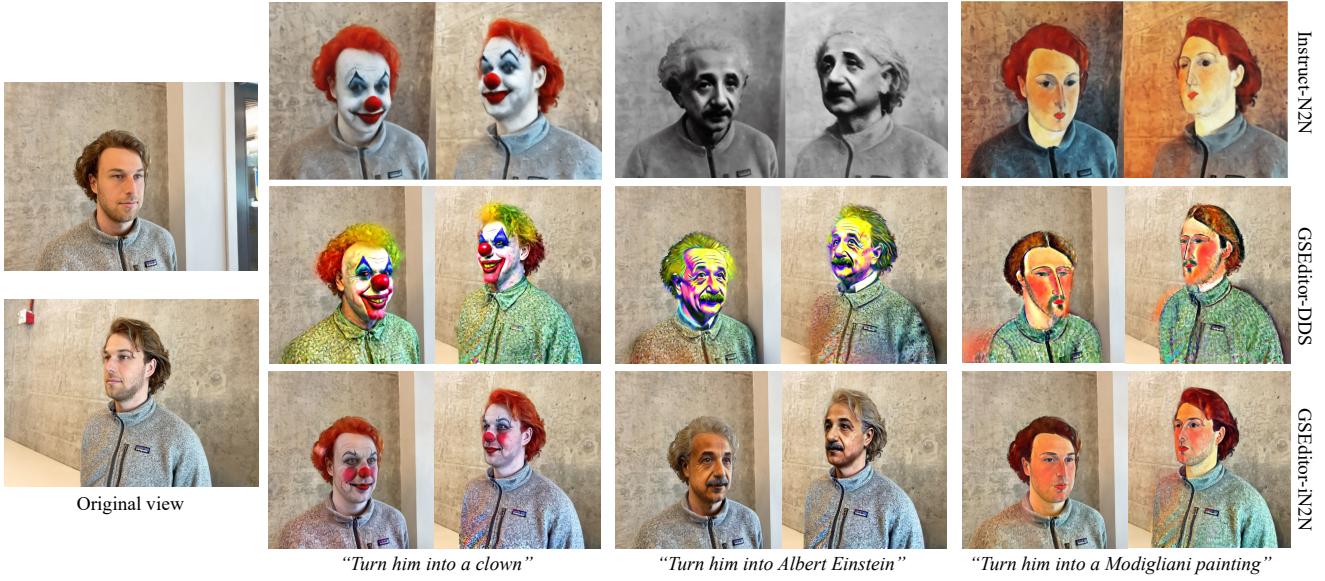


Figure 5. **Qualitative comparison.** It’s important to note the level of control we maintain over the editing area (the whole body of the man). Background and other non-target regions are essentially unaffected, in contrast to Instruct-Nerf2Nerf [12] where the entire scene undergoes changes. GaussianEditor-DDS and GaussianEditor-iN2N indicate that we utilize delta denoising score [14] and Instruct-Nerf2Nerf [12] respectively, as guidance for editing.

densification round in which a particular Gaussian point is produced. The initial Gaussians  $\Theta$ , are all assigned a generation of 0. During the training process for editing, points generated in the  $k$ -th densification round are marked as generation  $k$ .

Subsequently, we impose varying constraints on Gaussians from different generations to control their degree of flexibility. The older the generation, the stronger the constraints applied. Anchor loss is utilized to enforce these constraints. At the beginning of training, HGS records the attributes of all Gaussians as anchors. These anchors are then updated to reflect the current state of the Gaussians at each densification process. During training, MSE loss between the anchor state and the current state is employed to ensure that the Gaussians do not deviate too far from their respective anchors:

$$\mathcal{L}_{anchor}^P = \sum_{i=0}^n \lambda_i (P_i - \hat{P}_i)^2 \quad (7)$$

where  $n$  represents the total number of Gaussians and  $P$  denotes a certain property of the current Gaussian, including elements from the set  $x, s, q, \alpha, c$ . Here,  $\hat{P}$  refers to the same property recorded in the anchor state. The term  $\lambda_i$  indicates the strength of the anchor loss applied to the  $i$ -th Gaussian, which varies based on its generation. The overall training loss is defined as:

$$\mathcal{L} = \mathcal{L}_{Edit} + \sum_{P \in \{x, s, q, \alpha, c\}} \lambda_P \mathcal{L}_{anchor}^P \quad (8)$$

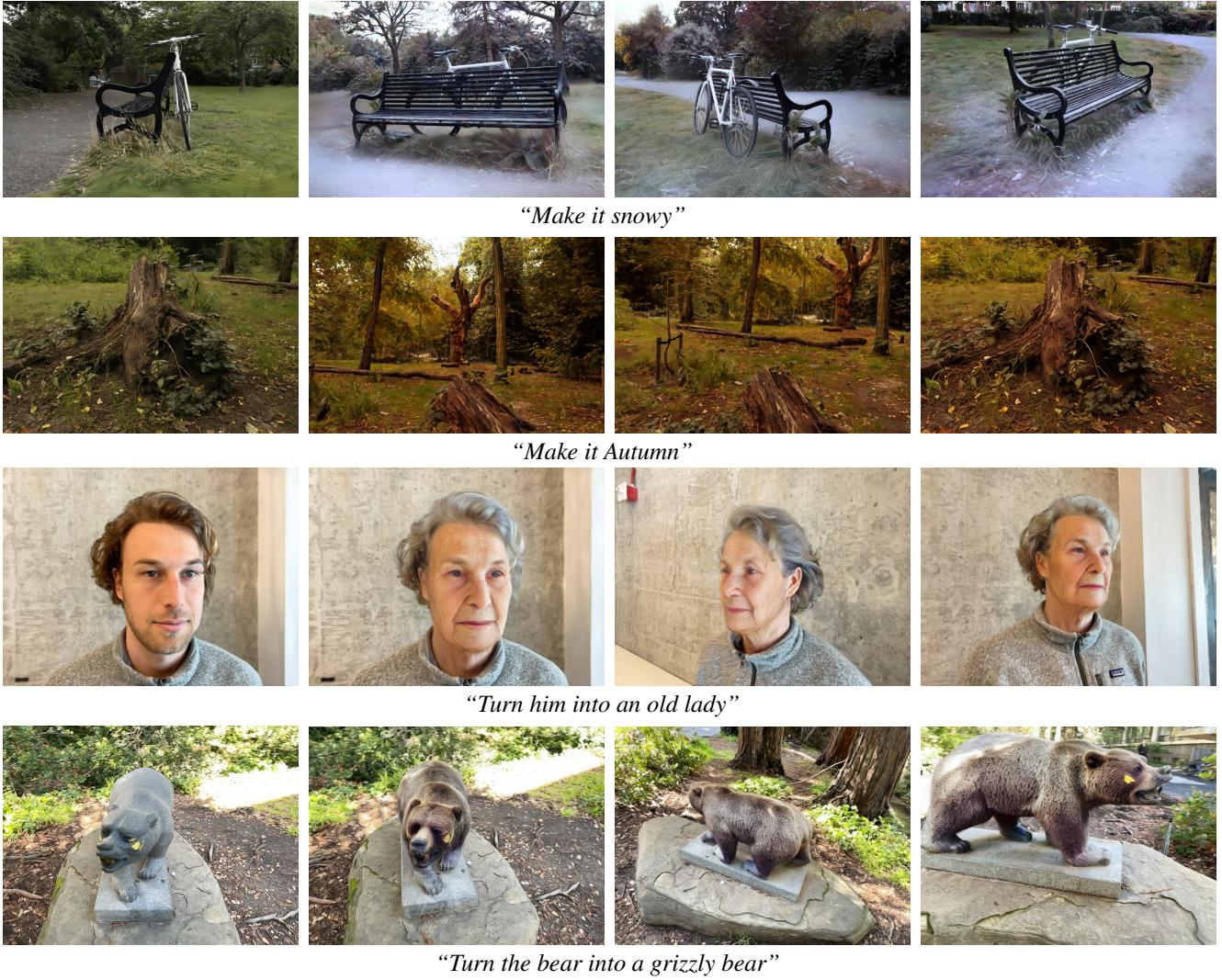
In this equation,  $\lambda_P$  signifies the strength of the anchor loss applied to property  $P$ , and  $\mathcal{L}_{Edit}$  is the edit loss defined in Sec. 3.2.

This generational design in HGS prevents the issue of excessive flexibility in GS when faced with stochastic loss. With each densification, the anchor loss weight  $\lambda_i$  for all previous generations of Gaussians is increased. As a result, the fluidity of the existing generations gradually decreases until it nearly solidifies. This approach ensures stable geometry formation under stochastic losses, relying on the almost unconstrained Gaussians from new densifications to carve out details. Furthermore, this method of applying anchor loss can effectively meet various editing needs. For instance, to limit changes in the original GS, one can increase the anchor loss weight for generation 0. Similarly, if there is no desire to alter color or geometry during editing, a stronger anchor loss can be applied to these specific properties.

Additionally, to address the challenge of manually determining a densification threshold, we regulate the densification process based on a percentage criterion. In this method, during each densification step, we selectively densify only those Gaussians whose 3D position gradients are within the top  $k\%$ . This strategy proves to be more manageable and intuitive than directly setting a threshold value in the Hierarchical Gaussian Splatting (HGS) framework.

### 4.3. 3D Inpainting

**Object Removal.** Simply removing Gaussians identified by a mask can lead to artifacts, especially at the interfaces where the object intersects with other Gaussians. To address this, we employ 2D inpainting techniques to provide guidance for filling these areas. However, effective 2D inpainting requires precise masks to offer better guidance. To generate these masks, after deletion, we use the KNN algorithm to identify Gaussians nearest to the ones removed, which are



**Figure 6. Extensive Results of GaussianEditor.** Our method is capable of various editing tasks, including face and scene editing. In face and bear editing, we restrict the editing area to the face using Gaussian semantic tracing, ensuring that undesired areas remain unchanged. The leftmost column demonstrates the original view, while the right three columns show the images after editing.

likely at the interface. These are then projected onto various views. We subsequently dilate the mask and fix any holes to accurately represent the interface area, thus creating a refined mask for the boundary zones. The whole object removal procedure typically takes only two minutes.

**Object Incorporation.** We define this task as follows: Within the 3D Gaussians  $\theta$ , given a camera pose  $p$  and the corresponding rendering  $I$  from this viewpoint, the user provides a 2D mask  $M$  on  $I$  indicating the area they wish to inpaint. Additionally, a prompt  $y$  is provided to specify the content of the inpainting. We then update  $\theta$  to fulfill the inpainting request.

Given  $I$ ,  $M$ , and  $y$ , the process begins with generating a 2D inpainted image  $I^M y$ , utilizing a 2D inpainting diffusion model as per [34]. Subsequently, the foreground object from  $I^M y$ , created by [34], is segmented and input into the

image-to-3D method referenced in [25] to generate a coarse 3D mesh. This coarse mesh is then transformed into 3D Gaussians  $\theta_y$ , and refined with HGS detailed in Sec. 4.2.

For aligning the coordinate system of  $\theta_y$  with  $\theta$ , the depth of  $I_y^M$  is first estimated using the technique from [38]. This depth is then aligned with the depth map rendered by  $\theta$  at camera pose  $p$ , using the least squares method. With this alignment, we can accurately determine the coordinates and scale of the inpainted foreground object in the coordinate system of  $\theta$ . After transforming  $\theta_y$  into the coordinate system of  $\theta$ , we simply concatenate them to produce the final inpainted 3D Gaussians.

It is important to note that due to our efficient design, the entire object incorporation procedure can be completed in approximately 5 minutes.

## 5. Experiments

### 5.1. Implementation Details



**Figure 7. Ablation study on Hierarchical Gaussian Splattering (HGS).** **Prompt:** make the grass on fire. Even when specifying the editing area with prompts, generative methods like Instruct-Pix2Pix [4] tend to edit the entire 2D image. Without HGS, Gaussians tend to conform to this whole-image editing by spreading and densifying across the entire scene, leading to uncontrollable densification and blurring of the image. With HGS, however, this kind of diffusion is effectively restrained.



**Figure 8. Ablation study on Semantic Tracing.** **Tracing Prompt:** Man As shown in Fig. 7, generative guidance techniques such as Instruct-Pix2Pix[4] exhibit a tendency to modify the entire 2D image. With the assistance of semantic tracing, we can confine the editing region to the desired area.

We utilize the highly optimized renderer implementation from [16] for Gaussian rendering and base our implementation on Threestudio [11]. All the original 3D Gaussians used in this work are trained using the methods described in [16]. Our experiments are conducted on a single RTX A6000 GPU. As detailed in Sec. 4.1, once we obtain segmentation masks from the 2D segmentation method outlined in [18], segmenting the 3D Gaussians takes only about 1 second.

For editing large scenes, the camera poses employed during the editing process are selected from a subset of the multi-view image dataset initially used for reconstruction. In the case of editing targeted objects, as facilitated by the GS segmentation detailed in Sec. 4.1, we generate a set of camera poses closely surrounding the segmented object. This

approach is adopted to increase the resolution of the object in the rendering, thereby enhancing the effectiveness of the editing process. Moreover, when the target object has a low degree of association with the scene, we opt to render only the target object to reduce computational load.

Depending on the complexity of the scene, the number of camera poses used in our experiments varies from 24 to 96. The editing process, influenced by the specified prompt and the complexity of the scene, typically involves optimizing for 500-1000 steps, taking about 5-10 minutes in total.

Regarding 3D inpainting for object incorporation, as detailed in Sec. 4.3, it takes approximately 3 minutes to generate a 3D mesh using the method from [25] and an additional 2 minutes to transfer this mesh into 3D Gaussians and refine it, while the composition process of two Gaussians takes less than 1 second.

### 5.2. Qualitative Comparisons

As illustrated in Fig. 5, GaussianEditor-iN2N surpasses other methods in both the quality of edits and controllability. Instruct-Nerf2Nerf, while producing edits with insufficient detail, cannot also control the editing area. GaussianEditor-DDS, due to the more challenging control of guidance offered by DDS loss [14] compared to Instruct-pix2pix [4], tends to result in oversaturated colors and less precise editing outcomes.

Additionally, our method exhibits exceptional control over the editing area. This is achieved through Gaussian semantic tracing, which identifies the Gaussians that require editing at each training step, for example, the entire human body in Fig. 5. It's important to note that in Instruct-Nerf2Nerf [12], the use of static 2D or 3D masks restricts the spatial freedom of the edits, as the permissible area for the edited subject is limited by these masks. Furthermore, the effectiveness of static masks diminishes as the geometries and appearances of 3D models evolve during training.

In Fig. 6, we demonstrate that GaussianEditor can accommodate a variety of scenarios, such as editing in large-scale scenes and facial swaps. In the case of large scenes, we did not apply Gaussian semantic tracing. However, for facial swaps, we traced the Gaussians corresponding to facial regions, achieving controllable and realistic editing.

We will include additional qualitative results in our supplemental materials to demonstrate the advantages of our method.

### 5.3. Quantitative Comparisons

As shown in Table 1, we conduct quantitative comparisons on user study and CLIP directional similarity (as shown in InstructPix2Pix [4] and StyleGAN-Nada [9]). GaussianEditor-iN2N not only demonstrates superior outcomes in user studies but also excels in CLIP Directional Similarity. Besides, Instruct-Nerf2Nerf [12] typically requires more than 30 minutes to complete the editing of a scene, whereas our method



	iN2N [12]	Ours (DDS)	Ours (iN2N)
User study	15.45%	12.27%	<b>72.28%</b>
CLIP Directional Similarity	0.1600	0.1813	<b>0.2071</b>

Table 1. **Quantitative Comparation.** GaussianEditor-iN2N outperforms in both user study evaluations and CLIP Directional Similarity [9] metrics.

only takes between 5 to 10 minutes.

#### 5.4. Ablation Study

As demonstrated in Fig. 7 and Fig. 8, We conducted ablation experiments on Hierarchical Gaussian Splatting(HGS) and Semantic Tracing. Without HGS, Gaussians tend to spread and densify across the scene, leading to uncontrolled densification and image blurring. This is typically caused by the tendency of methods like Instruct-Pix2Pix to edit the entire 2D image when prompts are used to define editing areas. However, HGS effectively circumvents this issue by constraining the mobility of the Gaussian in the old generation, ensuring that the overall scene does not exhibit excessive mobility. On the other hand, Semantic tracing assists GaussianEditor in limiting editing to a specified area without restricting the expansiveness of the editing region.

### 6. Conclusion

In our research, we introduce GaussianEditor , an innovative 3D editing algorithm based on Gaussian Splatting, designed for enhanced control and efficiency. Our method employs **Gaussian semantic tracing** for precise identification and targeting of editing areas, followed by Hierarchical Gaussian Splatting (HGS) to balance fluidity and stability in achieving detailed results under stochastic guidance. Additionally, we developed a specialized 3D inpainting algorithm for Gaussian Splatting, streamlining object removal and integration, and greatly reducing editing time.

**Limitation.** Similar to previous 3D editing works based on 2D diffusion models, GaussianEditor relies on these models to provide effective supervision. However, current 2D diffusion models struggle to offer effective guidance for certain complex prompts, leading to limitations in 3D editing.

### References

- [1] Chong Bao, Yinda Zhang, and Bangbang et al. Yang. Sine: Semantic-driven image-based nerf editing with prior-guided editing field. In *CVPR 2023*, pages 20919–20929, 2023. [3](#)
- [2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. [2](#)
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. [3](#)
- [4] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. *arXiv preprint arXiv:2211.09800*, 2022. [8](#)
- [5] Yiwen Chen, Chi Zhang, Xiaofeng Yang, Zhongang Cai, Gang Yu, Lei Yang, and Guosheng Lin. It3d: Improved text-to-3d generation with explicit view synthesis. *arXiv preprint arXiv:2308.11473*, 2023. [4](#)
- [6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022. [3](#)
- [7] Zilong Chen, Feng Wang, and Huaping Liu. Text-to-3d using gaussian splatting. *arXiv preprint arXiv:2309.16585*, 2023. [2, 3, 5](#)
- [8] Xinhua Cheng, Tianyu Yang, Jianan Wang, Yu Li, Lei Zhang, Jian Zhang, and Li Yuan. Progressive3d: Progressively local editing for text-to-3d content creation with complex semantic prompts. *arXiv preprint arXiv:2310.11784*, 2023. [4](#)
- [9] Rinon Gal, Or Patashnik, Haggai Maron, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. Stylegan-nada: Clip-guided domain adaptation of image generators. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022. [8, 9](#)
- [10] William Gao, Noam Aigerman, Thibault Groueix, Vladimir G Kim, and Rana Hanocka. Textdeformer: Geometry manipulation using text guidance. *arXiv preprint arXiv:2304.13348*, 2023. [3](#)
- [11] Yuan-Chen Guo, Ying-Tian Liu, Chen Wang, Zi-Xin Zou, Guan Luo, Chia-Hao Chen, Yan-Pei Cao, and Song-Hai Zhang. threestudio: A unified framework for 3d content generation. <https://github.com/threestudio-project/threestudio>, 2023. [8](#)
- [12] Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. *arXiv preprint arXiv:2303.12789*, 2023. [2, 4, 6, 8, 9](#)
- [13] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. [3](#)
- [14] Amir Hertz, Kfir Aberman, and Daniel Cohen-Or. Delta denoising score. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2328–2337, 2023. [6, 8](#)
- [15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. [2, 3, 12](#)
- [16] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ToG*, 42(4):1–14, 2023. [8](#)
- [17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. [5](#)
- [18] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer White-

- head, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 5, 8, 12
- [19] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. *arXiv preprint arXiv:2205.15585*, 2022. 3
- [20] Yuan Li, Zhi-Hao Lin, David Forsyth, Jia-Bin Huang, and Shenlong Wang. Climatenerf: Physically-based neural rendering for extreme climate synthesis. *arXiv e-prints*, pages arXiv–2211, 2022. 3
- [21] Zhaozhou Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *CVPR*, 2023. 3
- [22] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *CVPR*, pages 300–309, 2023. 3
- [23] Hao-Kang Liu, I Shen, Bing-Yu Chen, et al. Nerf-in: Free-form nerf inpainting with rgb-d priors. *arXiv preprint arXiv:2206.04901*, 2022. 3
- [24] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *ICCV 2021*, pages 5773–5783, 2021. 3
- [25] Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, et al. Wonder3d: Single image to 3d using cross-domain diffusion. *arXiv preprint arXiv:2310.15008*, 2023. 7, 8
- [26] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *arXiv preprint arXiv:2308.09713*, 2023. 3
- [27] Aryan Mikaeili, Or Perel, Mehdi Safaee, Daniel Cohen-Or, and Ali Mahdavi-Amiri. Sked: Sketch-guided text-based 3d editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14607–14619, 2023. 4
- [28] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 2, 3, 12
- [29] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 3
- [30] Atsuhiro Noguchi, Xiao Sun, Stephen Lin, and Tatsuya Harada. Neural articulated radiance field. In *ICCV 2021*, pages 5762–5772, 2021. 3
- [31] Jangho Park, Gihyun Kwon, and Jong Chul Ye. Ed-nerf: Efficient text-guided editing of 3d scene using latent space nerf. *arXiv preprint arXiv:2310.02712*, 2023. 2, 3, 4
- [32] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021. 2
- [33] Sida Peng, Yuanqing Zhang, Yinghao Xu, and et al. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR 2021*, pages 9054–9063, 2021. 3
- [34] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sndl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 7
- [35] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 3
- [36] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 2, 4
- [37] Amit Raj, Srinivas Kaza, Ben Poole, Michael Niemeyer, Nataniel Ruiz, Ben Mildenhall, Shiran Zada, Kfir Aberman, Michael Rubinstein, Jonathan Barron, et al. Dreambooth3d: Subject-driven text-to-3d generation. *arXiv preprint arXiv:2303.13508*, 2023. 4
- [38] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12179–12188, 2021. 7
- [39] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022. 4
- [40] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 3
- [41] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 5
- [42] Etaï Sella, Gal Fiebelman, Peter Hedman, and Hadar Averbuch-Elor. Vox-e: Text-guided voxel editing of 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 430–440, 2023. 4
- [43] Ruizhi Shao, Jingxiang Sun, Cheng Peng, Zerong Zheng, Boyao Zhou, Hongwen Zhang, and Yebin Liu. Control4d: Dynamic portrait editing by learning 4d gan from 2d diffusion-based editor. *arXiv preprint arXiv:2305.20082*, 2023. 4
- [44] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023. 2, 3, 5
- [45] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3835–3844, 2022. 2, 3
- [46] Can Wang, Ruixiang Jiang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Nerf-art: Text-driven neural radiance fields stylization. *IEEE Transactions on Visualization and Computer Graphics*, 2023. 2, 3

- [47] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023. 3
- [48] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*, pages 159–175. Springer, 2022. 3
- [49] Bangbang Yang, Chong Bao, and Junyi et al. Zeng. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In *ECCV 2022*, pages 597–614. Springer, 2022. 3
- [50] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023. 3
- [51] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *arXiv preprint arXiv:2310.10642*, 2023. 3
- [52] Taoran Yi, Jiemin Fang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. Gaussian-dreamer: Fast generation from text to 3d gaussian splatting with point cloud priors. *arXiv preprint arXiv:2310.08529*, 2023. 2, 3, 5
- [53] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztureli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 3
- [54] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, and et al. Nerf-editing: geometry editing of neural radiance fields. In *CVPR 2022*, pages 18353–18364, 2022. 3
- [55] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 2
- [56] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15838–15847, 2021. 4
- [57] Jingyu Zhuang, Chen Wang, Lingjie Liu, Liang Lin, and Guanbin Li. Dreameditor: Text-driven 3d scene editing with neural fields. *arXiv preprint arXiv:2306.13455*, 2023. 2, 3, 4
- [58] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001. 3

## Appendix

### A. Introduction

The content of our supplementary material is organized as follows:

- Firstly, we provide more qualitative results in Section B.
- Secondly, we demonstrate WebUI for GaussianEditor in Section C along with specifically tailored algorithm for WebUI 3D editing.
- We attach the video of using our WebUI, including tracing, editing, deleting, and adding objects.

### B. More Results

In Fig. 11, we demonstrate more results of GaussianEditor . Our method provides controllable, diverse, high-resolution 3D editing, needing only 2-7 minutes.

### C. WebUI

Although works in the neural radiance fields (NeRF) [28] domain also incorporate WebUIs, the slow rendering speeds of NeRF mean that users are confined to low resolutions and very low frame rates when using the WebUI, resulting in a subpar user experience. Fortunately, thanks to our adoption of Gaussian Splatting [15] in GaussianEditor , a method known for its rapid rendering capabilities, our WebUI can comfortably support usage at 2K resolution and 60fps. Besides, we leverage the interactivity of the webUI to enhance both semantic tracing and object incorporation applications, which will be discussed in the following two subsections.

#### C.1. Semantic Tracing with Point-base Prompts

Interactive WebUI applications with user interfaces are extremely important. In practical scenarios, users often intend to edit only specific areas of a complete scene, a task that can be challenging to specify solely through text prompts. For instance, it becomes difficult to determine through text which object a user wants to edit when multiple objects of the same type are present in a scene, and the user wishes to change just one of them. To address this issue, we propose semantic tracing with point-based prompts.

Semantic tracing with point-based prompts requires users to click on the screen to add 2D points from a specific view. Specifically, when the user clicks a point on the screen, we back-project this point into a spatial point based on the intrinsic and extrinsic parameters of the current viewpoint camera:

$$[x, y, z]^T = [\mathbf{R}|\mathbf{t}]z(\mathbf{p})\mathbf{K}^{-1}[p_x, p_y, 1]^T, \quad (9)$$

where  $[\mathbf{R}|\mathbf{t}]$  and  $\mathbf{K}$  denote the extrinsic and intrinsic of the current camera,  $\mathbf{p}$ ,  $z(\mathbf{p})$  and  $[x, y, z]^T$  refer to the user-clicked pixel, its corresponding depth, and the spatial point, respectively.

Subsequently, in other views, we re-project these spatial points onto the camera’s imaging plane, identifying the pixels corresponding to these 3D points. We use the projection points of these 3D points in the reference views as the point prompts for semantic segmentation with SAM [18]. Then, we unproject these semantic segmentation maps back to Gaussians, as demonstrated in the main text.

As can be seen in Fig. 9, with only about five points indicated by the users, semantic tracing with point-based prompts enables finer granularity control over the areas to be tracked.

#### C.2. Object Incorporation with WebUI

As detailed in the main paper, our proposed method for 3D inpainting with object incorporation allows for the addition of objects specified by text in designated areas. The webUI facilitates users in easily drawing 2D masks to define these areas. Moreover, in this method of 3D inpainting for object incorporation, depth information is crucial for seamlessly integrating new objects into the original Gaussian Scene. Current methods for monocular depth estimation, however, can’t always provide completely accurate depth maps, leading to imprecise alignment. Therefore, as depicted in Fig. 10, we utilize the webUI to modify the scale of the estimated depth, enabling users to achieve a more accurate alignment of the objects.

To be more specific, users control the Gaussian scale by sliding a slider. After obtaining a new depth scale, we update the position and size of the added objects according to the new depth scale. Since the entire process involves only minor adjustments to the position and scale parameters of a few Gaussians, real-time scaling can be achieved.

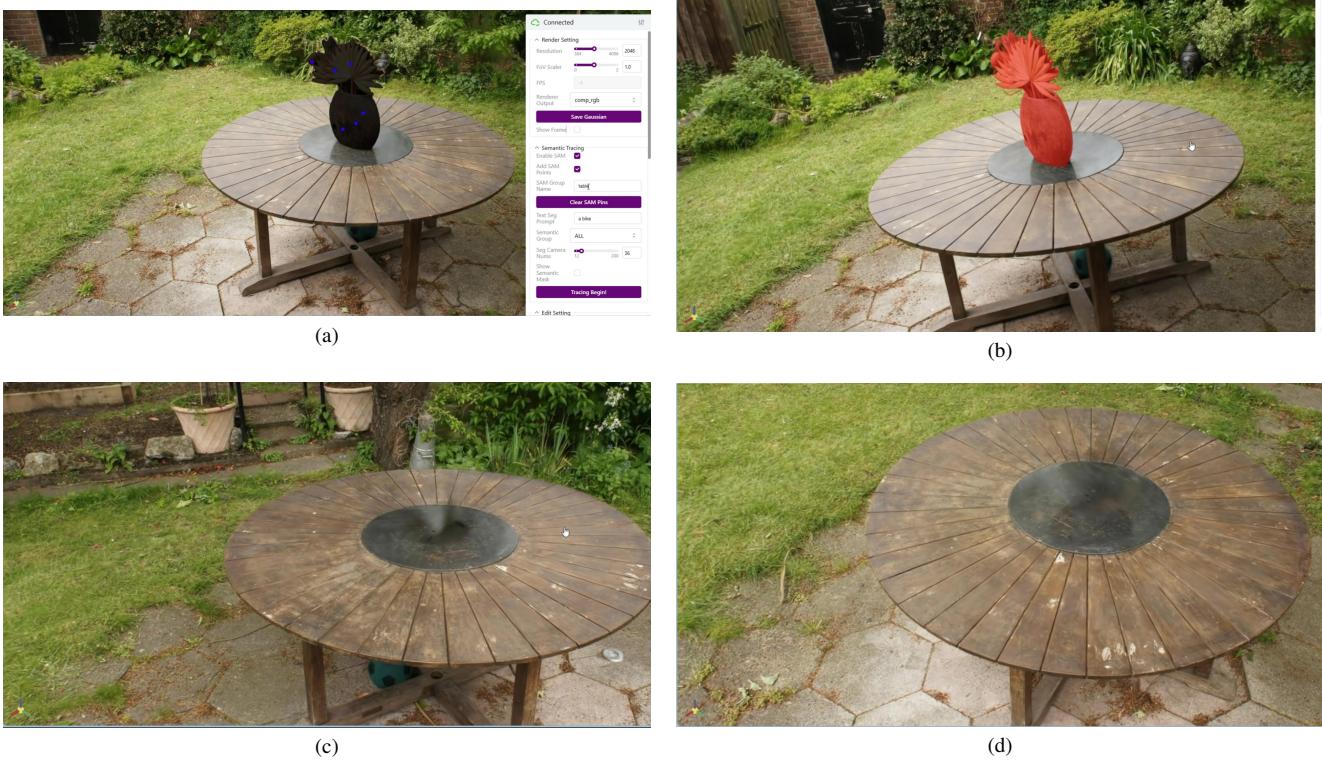


Figure 9. Semantic Tracing with Point-based Prompts. In (a), users provide key points on a view by clicking the screen with the mouse. In (b), we segment the target object based on these points. (c) and (d) depict the results after removing the segmented objects. It can be seen from the above that our point-based tracing method offers high precision and interactivity.



Figure 10. Object Incorporation with WebUI. Empowered by our interactive WebUI, the depth scale addresses the limitation of monocular depth estimation, which cannot guarantee precise depth map predictions. As can be seen in (a), inaccurate results can lead to failures when aligning generated objects with Gaussian scenes. We leverage the interactive nature of our WebUI to dynamically adjust the estimated depth in real-time, thus resolving this issue, demonstrated in other images.



*“Remove the vase”*



*“Add a teddy bear on the bench”*



*“Turn him into Hulk”*



*“Turn him into an elf”*

**Figure 11. More results of GaussianEditor.** GaussianEditor allows for fast, versatile, high-resolution 3D editing, requiring only 2-7 minutes and 10-20GB of GPU memory on a single A6000 GPU. Please note that the background of face editing scenes remains unchanged.