

# 100Editor: 100+ Views per Batch and Minute-Scale View-Consistent 3D Editing

## Supplementary Material

### Contents

<b>A Optimized 3DGS Training</b>	<b>2</b>
A.1. Method . . . . .	2
A.2. Experiments . . . . .	3
<b>B 100Editor Details</b>	<b>4</b>
B.1. Semantic Editing . . . . .	4
B.2. Additive Editing . . . . .	5
B.3. Subtractive Editing . . . . .	5
B.4. Non-rigid Editing . . . . .	5
<b>C Additional Experiments</b>	<b>6</b>
C.1. Implementation Details . . . . .	6
C.2. Qualitative Analysis . . . . .	6
C.3. Quantitative Analysis . . . . .	7
C.4. Analysis of Merging Types . . . . .	8
C.5. Ablation Studies . . . . .	9
<b>D 3D Editing Software</b>	<b>10</b>
D.1. Data Preprocessing . . . . .	10
D.2. 3DGS Visualization . . . . .	10
D.3. 3D Reconstruction . . . . .	13
D.4. 3D Editing . . . . .	13
D.5. Image Fitting . . . . .	16

## A. Optimized 3DGS Training

In this section, we first introduce an optimized 3DGS training method, followed by experiments to demonstrate the effectiveness of our method.

### A.1. Method

In this section, we introduce the gentle depth regularization and appearance embedding modules, designed to mitigate reconstruction artifacts and enhance rendering fidelity. An overview of the optimized 3DGS training pipeline is shown in Fig. 9.

**Gentle Depth Regularization Module.** Given a set of training views  $I_{\text{rec}} = \{I_{\text{rec}}^1, I_{\text{rec}}^2, \dots, I_{\text{rec}}^n\}$ , where  $n$  represents the total number of views. Prior to initializing the 3DGS model, we employ a pre-trained monocular depth estimator to predict the depth for the training view set  $I_{\text{rec}}$ , resulting in a set of predicted depth maps  $D_{\text{pre}} = \{d_{\text{pre}}^1, d_{\text{pre}}^2, \dots, d_{\text{pre}}^n\}$ . Subsequently, the training view set  $I_{\text{rec}}$  is processed using Colmap to obtain the initial point cloud and corresponding camera parameters  $\text{Cam} = \{\text{Cam}_1, \text{Cam}_2, \dots, \text{Cam}_n\}$ . The sparse point cloud is then used to construct the 3DGS  $G_{\text{rec}}$ .

Monocular depth estimators typically predict relative depth, whereas the depth  $d_{\text{render}}^k$  rendered by the 3DGS model for the  $k$ -th view is anchored to the Colmap scale. Consequently, applying a direct loss function (such as squared error) is ineffective. While many studies propose aligning  $d_{\text{render}}^k$  and  $d_{\text{pre}}^k$  by estimating scale and shift parameters, the transformation between these depth maps is not constant across all pixels. A simple alignment may therefore introduce additional errors, leading to inaccuracies in the final result.

To address this problem, we adopt the Pearson correlation coefficient to compute the loss  $\mathcal{L}_d$  between  $d_{\text{render}}^k$  and  $d_{\text{pre}}^k$ :

$$r(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}}, \quad (5)$$

$$\mathcal{L}_d = 1 - r(d_{\text{render}}^k, d_{\text{pre}}^k).$$

Given that the Pearson correlation is closely related to normalization, this loss function encourages high cross-correlation between corresponding patches in the two depth maps, irrespective of variations in the absolute range of depth values. It is thus better equipped to handle larger spatial regions and capture richer local depth structures. We employ a gentle depth regularization module to guide the depth rendered by the 3DGS model to converge with the depth values from the pre-trained monocular estimator. This results in cleaner 3DGS-rendered depth and mitigates artifacts in the final reconstruction.

**Appearance Embedding Module.** Due to non-uniform illumination in the scene, photographs taken from various viewpoints exhibit significant appearance variations. The original 3DGS model compensates for these view-dependent appearance changes by introducing additional artifacts. Therefore, in order to solve the problem, we adopt an appearance embedding method. By incorporating a learnable appearance embedding into the rendered images, we aim to eliminate the artifacts produced during reconstruction.

First, the  $k$ -th view image,  $I_{\text{render}}^k \in \mathbb{R}^{H \times W \times 3}$ , is rendered from the initial 3DGS model  $G_{\text{init}}$ , where  $(H, W)$  represents the size of the rendered image. Furthermore, we downsample  $I_{\text{render}}^k$  to obtain  $I_{\text{down}}^k \in \mathbb{R}^{H' \times W' \times 3}$ , where  $(H', W')$  represents the size of the downsampled image. The downsampling operation ensures that the transformation map can better capture the global information of the entire rendered image and operate in a lower-dimensional space, thereby reducing computational overhead.

Subsequently, a learnable appearance embedding  $E \in \mathbb{R}^{H \times W \times C}$  is introduced, where  $C$  represents the number of channels in the appearance embedding. The rendered image is augmented at each position with the corresponding appearance embedding  $E$  to obtain  $\tilde{I}^k \in \mathbb{R}^{H \times W \times (C+3)}$ . Then, a convolutional neural network (CNN) is utilized to derive the corresponding transformation map  $S \in \mathbb{R}^{H \times W \times 3}$  from  $\tilde{I}^k$ . The corresponding appearance-enhanced image  $I_a^k \in \mathbb{R}^{H \times W \times 3}$  is then obtained through element-wise multiplication:

$$I_a^k = S \odot I_{\text{render}}^k, \quad (6)$$

where  $\odot$  represents the Hadamard product. The loss for this component:

$$\mathcal{L}_a = (1 - \lambda)\mathcal{L}_1(I_a^k, I^k) + \lambda\mathcal{L}_{\text{SSIM}}(I_{\text{render}}^k, I^k), \quad (7)$$

where  $\lambda$  is a weighting parameter,  $\mathcal{L}_1$  represents the L1 loss, and  $\mathcal{L}_{\text{SSIM}}$  denotes the SSIM-based loss function. Since  $\mathcal{L}_{\text{SSIM}}$  primarily penalizes differences in structural information between images, it is used here to ensure the structural information of  $I_{\text{render}}^k$  remains close to that of the ground truth image  $I^k$ . As for the  $\mathcal{L}_1$  loss, it mainly targets appearance variations. We apply it here to ensure that the rendered image  $I_a^k$ , which may have appearance changes relative to other views, can better adapt to the ground truth image  $I^k$ . This helps to avoid excessive artifacts in the reconstruction caused by differences in illumination intensity. The total loss function  $\mathcal{L}$  for the entire 3DGS model training process is:

$$\mathcal{L} = \mathcal{L}_a + \lambda_d \mathcal{L}_d, \quad (8)$$

where  $\lambda_d$  is the weight coefficient for the depth loss  $\mathcal{L}_d$ .

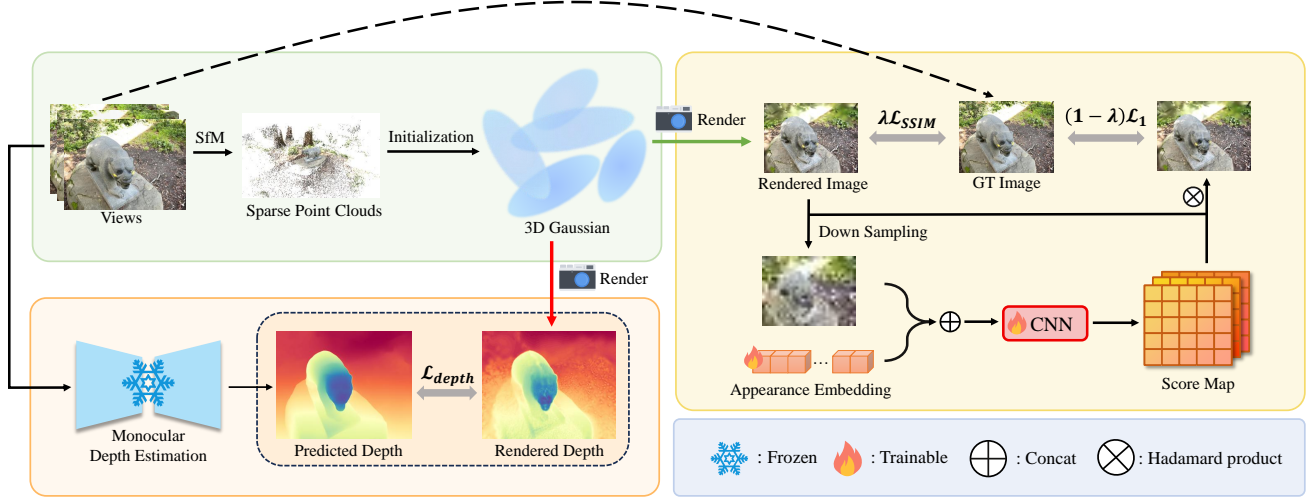


Figure 9. **Overview of Reconstruction.** The improved 3DGS training collectively enhances the fidelity of 3D reconstruction by incorporating a Gentle Depth Regularization Module and an Appearance Embedding Module, thereby yielding a scene representation with minimal artifacts for subsequent scene editing.

## A.2. Experiments

In this section, we first present implementation details, followed by comparisons with the baseline 3DGS and the ablation study.

**Implementation Details** In our method, we set the total number of training iterations to 30,000. The loss function weights  $\lambda$  and  $\lambda_d$  are set to 0.2 and 0.5, respectively, to better balance the L1, SSIM, and depth losses. Our evaluation was performed on the scenes from LLFF, IN2N, as well as additional real-world scenes captured by ourselves. Furthermore, we utilize Depth-Anything2 as the pre-trained depth predictor.

**Qualitative Analysis** Fig. 10 presents the comparison of reconstruction quality between our method and 3DGS. The 3DGS model tends to generate significant artifacts in the reconstructed scenes, which severely disrupts 3D editing. In contrast, our method effectively suppresses these artifacts, thereby significantly enhancing reconstruction fidelity.

**Loss Function Comparison** As shown in Fig. 11, we conducted a comparison of three loss functions. Both the L1 and L2 loss functions introduce associated artifacts, with the L2 loss function generating the most spatial artifacts. This is primarily because the depth rendered by the 3DGS model and the depth predicted by the monocular depth estimator lack direct correspondence, rendering a direct application of L1 and L2 loss ineffective. Benefiting from the inherent invariance of the Pearson Correlation Coefficient (PCC) to linear scale and shift variations, our model is able

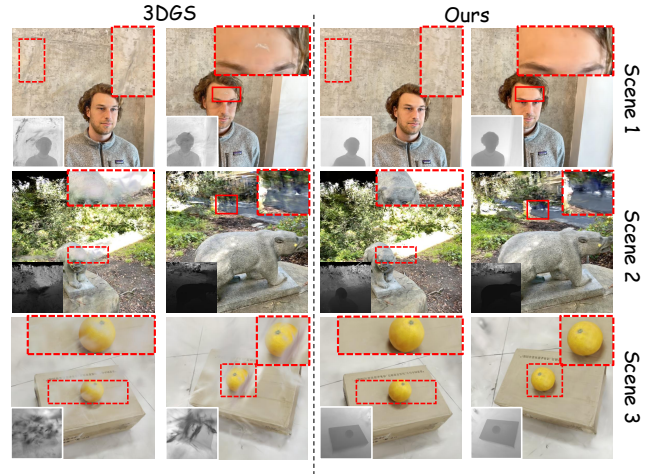


Figure 10. **Reconstruction Comparison.** Compared to the vanilla 3DGS, the improved method mitigates artifacts and yields cleaner depth maps.

to focus on learning consistent local geometric structures without being confounded by global discrepancies, such as the absolute range of depth values or overall offsets. As a result, our model significantly reduces artifacts produced during the reconstruction process.

**Quantitative Analysis** Table 4 presents the quantitative reconstruction results of our improved 3DGS. Our model outperforms the baseline on the LLFF dataset under a challenging 3-view sparse setting. Specifically, improvements in SSIM ranging from 0.095 to 0.325 are observed, accom-

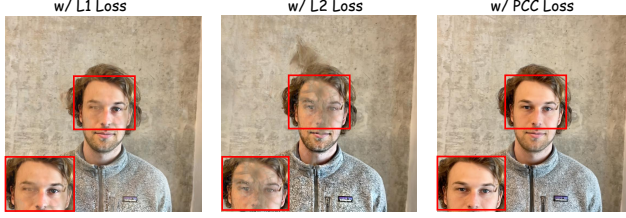


Figure 11. **Depth Loss Comparison.** The PCC-based loss function, compared to other loss formulations, contributes to reduced artifacts in the reconstruction results and improves reconstruction fidelity.

Table 4. **Quantitative Analysis of Reconstruction.** Across multiple scene test cases, the reconstruction results achieved by our improved 3DGS exhibit superior metrics, particularly in the context of sparse-view reconstruction.

Scene	Model	SSIM $\uparrow$	PSNR $\uparrow$	LPIPS $\downarrow$
Flower	3DGS	0.470	16.009	0.369
	Ours	<b>0.565</b>	<b>18.503</b>	<b>0.338</b>
Fern	3DGS	0.431	15.091	0.449
	Ours	<b>0.529</b>	<b>15.339</b>	<b>0.400</b>
Horns	3DGS	0.559	16.416	0.362
	Ours	<b>0.654</b>	<b>18.011</b>	<b>0.313</b>
Room	3DGS	0.457	12.175	0.536
	Ours	<b>0.782</b>	<b>20.558</b>	<b>0.247</b>
Face	3DGS	0.825	26.062	0.210
	Ours	<b>0.826</b>	<b>26.445</b>	<b>0.206</b>
Fangzhou	3DGS	<b>0.893</b>	30.069	0.177
	Ours	0.890	<b>30.785</b>	<b>0.179</b>

panied by PSNR increases from 0.248 to 8.383 and LPIPS reductions from 0.031 to 0.289. Furthermore, on the IN2N dataset under the standard dense-view setting, our method continues to exhibit superiority, notably achieving PSNR gains of 0.383 and 0.716 on the Face and Fangzhou scenes, respectively. These advancements are attributed to our proposed gentle depth regularization module and appearance embedding module, which effectively regularize the geometry and mitigate the overfitting and artifacts.

**Ablation Study** We conduct an ablation study to validate the effectiveness of our proposed modules, as illustrated in Fig. 12. As observed in the first row, the proposed appearance embedding module effectively mitigates artifacts caused by inconsistent illumination, thereby enhancing reconstruction quality. Furthermore, the results in the second row demonstrate that incorporating the PCC-based depth regularization reduces artifacts and maintains superior con-

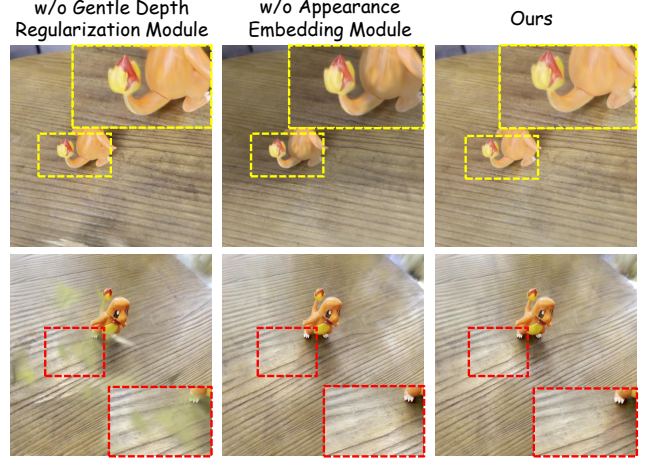


Figure 12. **Ablation Study of Reconstruction.** The Gentle Depth Regularization Module is effective in mitigating floaters that arise during the reconstruction process, while the Appearance Embedding Module reduces artifacts induced by varying illumination conditions across different training viewpoints.

tinuity in the corresponding depth maps.

## B. 100Editor Details

### B.1. Semantic Editing

Compared with existing 3D editing approaches, our proposed Views Token Merging significantly reduces the computational complexity of self-attention. Specifically, during the self-attention operation where tokens are projected into queries  $Q$ , keys  $K$ , and values  $V$ , the complexity for  $x_i \in \mathbb{R}^{b_i \times N \times C}$  with our method is reduced to  $O((1.08N + 0.12b_iN)^2)$ . In contrast, DGE[10], which achieves multi-view consistency by extending self-attention into a spatio-temporal mechanism, incurs a complexity of  $O(b_i^2N^2)$ . Clearly, when  $b_i > 1$ , the cost of DGE substantially exceeds ours, with the gap widening quadratically as the number of views  $b_i$  increases. Through a comparison between 100Editor and DGE, we demonstrate that our approach not only maintains superior or comparable multi-view consistency but also significantly alleviates the computational burden, delivering substantial improvements in both effectiveness and efficiency.

**3D Segmentation for 360° Scenes.** Segmenting objects in 360° scenes on a per-image basis can result in incomplete or inconsistent masks across different perspectives, as shown in Fig. 13. To address this challenge, we reframe the task as a video object segmentation problem and leverage the capabilities of SAM2. Our approach begins by selecting a single perspective where the target object is clearly visible. We provide positive and negative point prompts to generate an accurate initial mask for this view, which we



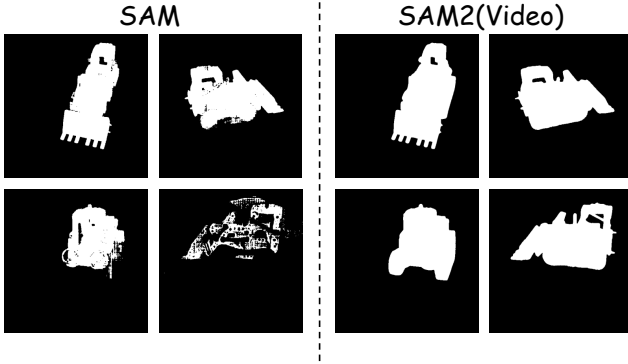


Figure 13. **360° scene segmentation.** Applying the video segmentation in SAM2 to 360° scenes can yield more complete 2D masks, thereby promoting more accurate 3D local editing.

then treat as the first frame of a pseudo-video sequence. The remaining perspective images are then sorted to form the subsequent frames. The initial mask is then propagated through this sequence, ensuring the generation of consistent and complete segmentation masks for all views.

**Blending Strategy.** To preserve background consistency between the original and edited images, we leverage a blending strategy based on the masks  $M$  acquired via multi-view segmentation:

$$I'_e = (1 - \mathcal{G}(M)) \odot I_v + \mathcal{G}(M) \odot I_e, \quad (9)$$

where  $I_v$  and  $I_e$  respectively represent the original image for batch editing and the editing result output by IP2P, and  $\mathcal{G}(\cdot)$  denotes the Gaussian blur operation, used to smooth jagged edges. Finally, we use  $I'_e$  to drive the update of 3DGS parameters. In each optimization step, we minimize the weighted combination of L1 loss and LPIPS loss to update the 3DGS:

$$\mathcal{L}_e = \lambda_1 \mathcal{L}_1(I_v^i, I_e^i) + \lambda_2 \mathcal{L}_{\text{LPIPS}}(I_v^i, I_e^i), \quad (10)$$

where  $\lambda_1$  and  $\lambda_2$  are loss weights. This mechanism ensures the precision of local editing and global consistency.

## B.2. Additive Editing

As shown in Fig. 14, we propose a coarse-to-fine framework for additive editing. In the initial coarse adding stage, we first specify a mask on a single-view image at the desired location for the new object. This mask, along with the original image, is fed into BrushNet[39] to perform single-view additive editing. Subsequently, we employ the SAM2[50] model to segment the newly added object and utilize DreamGaussian[56] to generate a coarse 3DGS representation from the segmented object image. Leveraging the depth information predicted from the previous single-view editing step, this newly generated 3DGS object is then

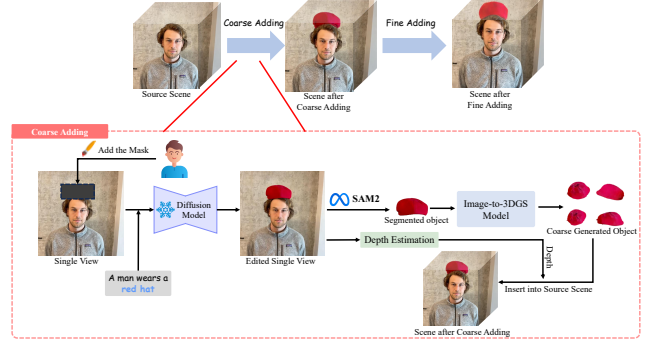


Figure 14. **Overview of Additive Editing in 100Editor.** Additive editing primarily adopts a coarse-to-fine editing, where it first generates a relevant coarse 3DGS and adds it to the original scene, and then further optimizes the entire scene.

inserted into the original 3D scene. The Fine Stage pipeline is consistent with the semantic editing process, where we perform batch multi-view optimization with our large-batch editing module to refine the scene, yielding more accurate and high-fidelity additive edits.

## B.3. Subtractive Editing

We propose a two-stage framework for subtractive editing in 3D scenes: coarse and fine editing, as shown in Fig. 15. In the coarse removal stage, we employ an interactive 3D segmentation module to identify and remove the Gaussians corresponding to the target object. This initial removal, however, often leaves undesirable artifacts at the interface between the object and the surrounding scene.

To address this, the fine removal stage refines the result using a conditional diffusion model. First, we generate a clean reference by providing an original single-view rendering and a removal instruction to a large multimodal model (Gemini), which inpaints the target object in 2D. This edited image then serves as a visual condition for the subsequent steps. We render multi-view images and corresponding masks from the coarsely edited scene and input them, along with the conditional image, into our diffusion model, LeftRefill[6]. By conditioning the inpainting of all views on a single reference image, our approach ensures high multi-view consistency. Finally, newly generated and consistent images are used to fine-tune the 3D Gaussian Splatting representation, achieving the object removal.

## B.4. Non-rigid Editing

We achieve non-rigid editing by building on PDS [34] and SC-GS [25] respectively. To address the limitation of the original PDS, which processes only a single image during each optimization step, we integrate our batch-consistent multi-view editing module into the framework, yielding higher-fidelity deformations with fewer artifacts. For SC-

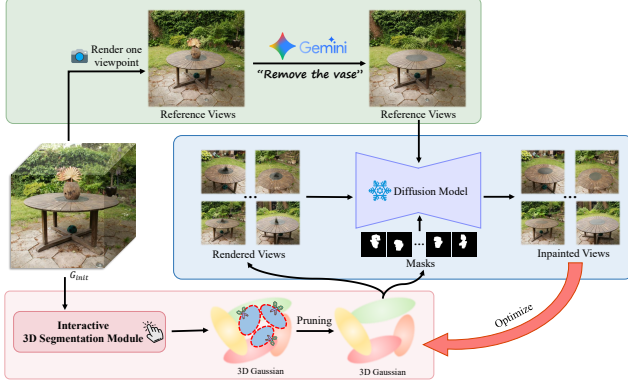


Figure 15. **Overview of Subtractive Editing in 100Editor.** Subtractive editing primarily uses a two-stage approach: in the first stage, it directly deletes the corresponding Gaussian primitives in the scene, and in the second stage, it performs relevant inpainting on the scene after the object has been deleted.

GS, which manipulates 3DGS via sparse control points but often suffers from unintended distortions in non-target regions, we incorporate our proposed interactive 3D segmentation module. This allows us to spatially constrain the deformation to user-specified areas, enabling precise local editing.

## C. Additional Experiments

### C.1. Implementation Details

Throughout the entire 3D scene editing process, we performed three rounds of iterative batch editing. Additionally, for the Views Token Merging process, we set  $\rho$ ,  $\rho_g$ , and  $p$  to 0.9, 0.8, and 0.5, respectively.

### C.2. Qualitative Analysis

Fig. 16 reports the editing results of 100Editor on our self-captured real-world scenes. Even on this real-world data, which is subject to environmental disturbances, 100Editor achieves high-fidelity and semantically precise editing, while perfectly preserving the original details in unedited regions and the overall consistency of the scene.

**Comparison with NeRF-based models.** As shown in Fig. 17, we conduct a quantitative comparison between 100Editor and two representative NeRF-based baselines, including Instruct-NeRF2NeRF[21] and VICA-NeRF[15]. The results from the baseline models generally exhibit artifacts and suffer from over-editing problems. This is mainly because both NeRF-based models cannot implement local editing. In contrast, 100Editor, by integrating our proposed interactive 3D segmentation module and batch-consistent multi-view editing module, achieves higher quality and more precise editing results.

**Iterative Scene Editing.** As shown in Fig. 29, we implement a multi-turn iterative process for 3D scene editing to further validate the effectiveness of 100Editor. Each iteration requires a corresponding editing instruction and the selection of the target editing region. 100Editor utilizes the output from the preceding turn as the input for the current one, progressively accumulating editing effects. This capability facilitates the construction of complex scenes that are challenging to realize with a single command. For example, as shown in the third-row case, to meet personalized user requirements, the male undergoes three successive edits to fulfill the complex directive: *“a man wears white pants with an Iron Man suit and a bronze statue face.”*

**Additive Editing.** Fig. 18 presents additional additive editing results. 100Editor precisely inserts target objects into the original scene while preserving high visual coherence with the surrounding content.

**Subtractive Editing.** Fig. 19 shows additional results for subtractive editing. For the *“Remove the mouse”* instruction, the result demonstrates that 100Editor can precisely remove the mouse, even in complex scenes containing multiple objects. Furthermore, the edited desktop area maintains its original flatness and suppresses the generation of artifacts.

**Non-rigid Editing.** Fig. 20 demonstrates the efficacy of our proposed module in Non-rigid Editing. For the prompt *“A photo of a man raising his arms”*, the employment of Batch Editing and Views Token Merging effectively mitigates artifacts in the edited scene, thereby enhancing the overall quality of the 3D editing. Conversely, for the instruction *“Shorten his neck”*, the interactive 3D segmentation module ensures that the manipulation is strictly confined to the target region, minimizing unintended interference with the background.

**Complex Editing.** Fig. 21 presents qualitative results on more challenging editing cases, including non-color attribute edits, cluttered scenes with heavy occlusion, and repeated objects with similar appearance. These scenarios are difficult because the editor must infer fine-grained semantic changes, localize the correct target under ambiguity, and avoid undesired modifications to nearby distractors. Nevertheless, 100Editor remains effective in these cases, producing the intended edits while preserving surrounding content and maintaining multi-view coherence. These results suggest that our batch-consistent editing and interactive 3D segmentation modules generalize beyond simpler editing settings.

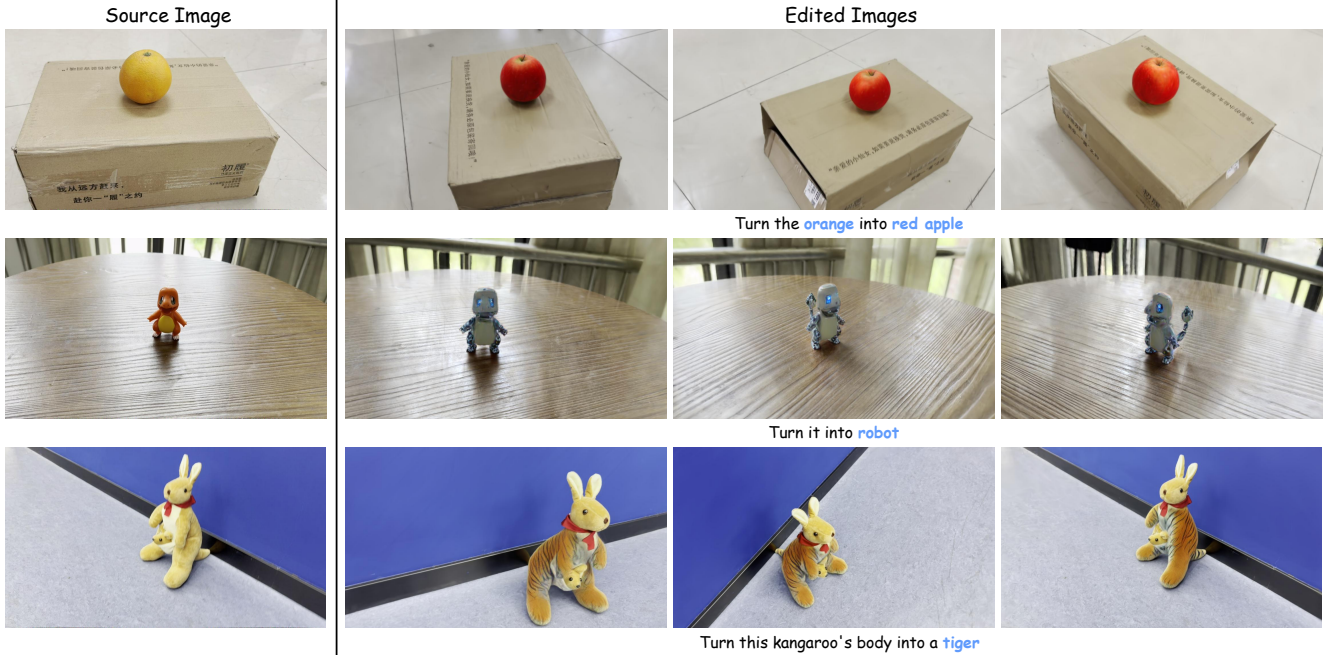


Figure 16. **Performance on Self-Collected Scene.** 100Editor also achieved promising results on the real-world scene dataset we collected ourselves.

### C.3. Quantitative Analysis

The CLIP scores and editing time in Table 1 are the average metrics calculated across all test cases presented in Table 6, and the editing instructions for each case are illustrated in Table 7. For the User Study, the evaluation was conducted using a designed questionnaire with questions covering 11 cases, an example of which is shown in Fig. 24. To ensure measurement accuracy, the options for each question in the questionnaire were randomized. We randomly selected 28 participants to complete the questionnaire. Finally, we calculate the proportion of votes that each model’s editing results received relative to the total number of votes. The specific results are presented in Fig. 22. 100Editor received the most votes, accounting for 41.88% of the total, which indicates that 100Editor is capable of providing higher-quality edits that are more faithful to the users’ instructions. As shown in Fig. 23, we tallied the number of votes received by each model for each case. 100Editor received the highest number of votes in 6 out of 11 cases, demonstrating its ability to achieve higher editing quality compared to other models.

Fig. 25 reports the time consumption of various models at each editing step. Taking the editing instruction “*Turn him into Harry Potter*” as an example, 100Editor not only demonstrates higher editing efficiency but also achieves finer detail fidelity. Compared to GaussianEditor [11] and DGE [10], 100Editor avoids repeated iterative updates on single views and complex modifications to Self-Attention

through batch editing and the Views Token Merging mechanism, thereby significantly improving editing speed. Furthermore, 100Editor employs parallel diffusion inference and an efficient renderer, further reducing the time required to optimize 3DGS parameters. Moreover, the early stopping strategy we introduced ensures that the editing process can be completed in fewer time steps. The final results indicate that 100Editor achieves the best visual effects with the shortest time and number of editing steps.

Table 5 reports the batch scalability on scene datasets featuring dense training views. Unlike DGE and GaussCtrl, which encounter Out of Memory (OOM) errors with large batches, 100Editor exhibits a slow increase in VRAM usage relative to batch size. This efficiency allows for larger batch editing, thereby facilitating better editing performance.

Fig. 26 shows the effect of batch size on editing consistency and runtime in the *garden* scene. As the batch size increases, the MET3R score consistently decreases, indicating that jointly optimizing more views improves cross-view consistency. This improved consistency also makes CPS more effective. We therefore do not use CPS at a batch size of 20, because each optimization round covers too few views to provide a reliable stopping signal, and enable it from 40 onward, where larger batches make the optimization more stable. As the batch size further increases, both the consecutive-check threshold  $P$  and the number of batch editing rounds can be reduced, allowing CPS to terminate each optimization round earlier and thus shorten the total



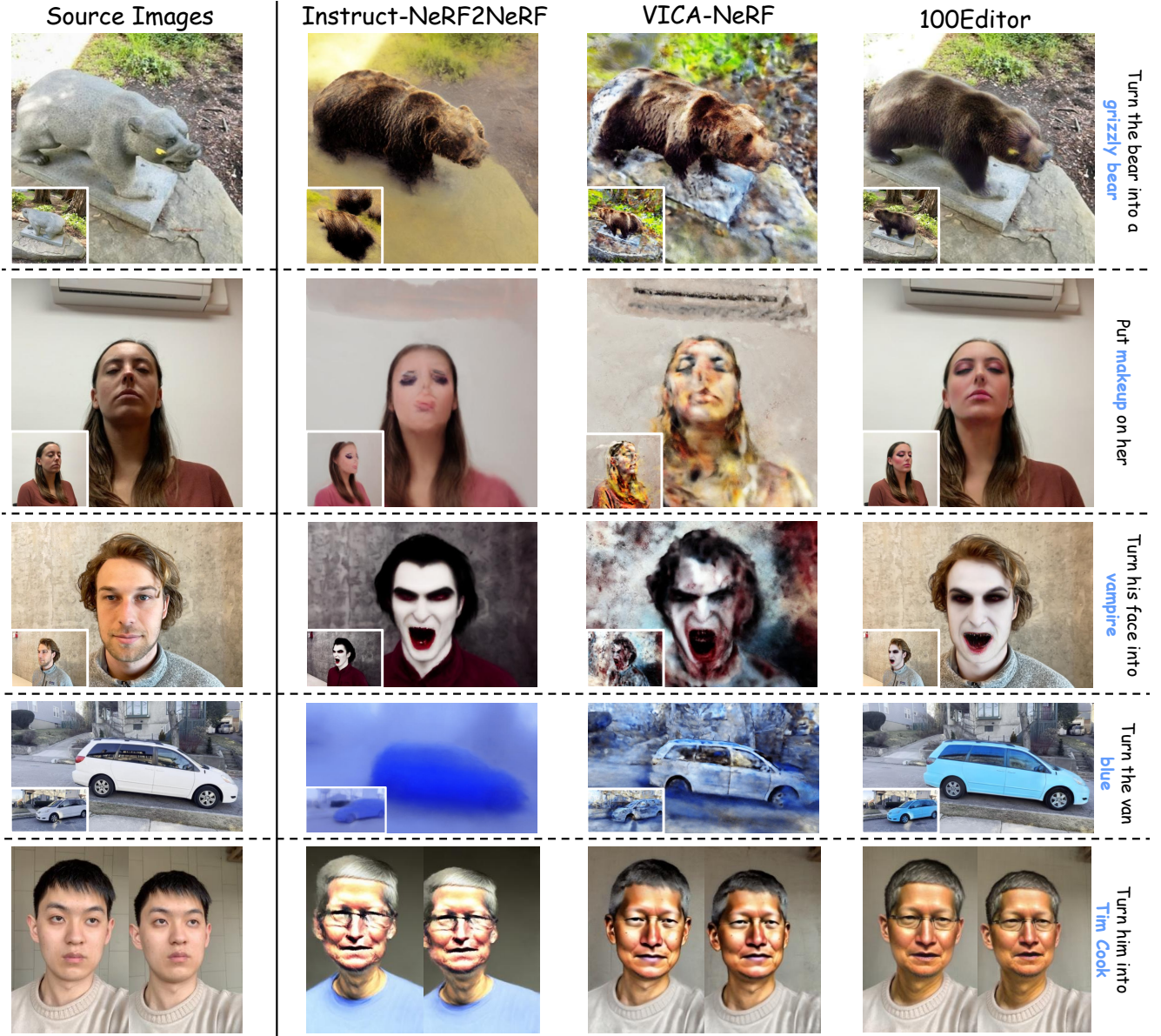


Figure 17. **Comparison with NeRF-based models.** Compared with NeRF-based 3D editing methods, 100Editor enables more precise local editing and achieves higher quality in scene editing.

editing time. However, when the batch size exceeds 100, the stopping point becomes similar across settings, and the additional overhead mainly comes from the batch editing stage itself. Consequently, further increasing the batch size increases the overall runtime.

#### C.4. Analysis of Merging Types

The choice of token merging strategy significantly impacts the final 3D editing result. We evaluated four distinct merging methods, as shown in Fig. 30. Using the editing instruction “Turn him into a Thanos”, the “Max Merging”

strategy resulted in over-editing, causing excessive alterations to facial color and features. Conversely, the “Min Merging” strategy introduced color and structural artifacts, particularly in the hair region. Although the “Mean Merging” strategy produced stable results, it lacked sufficient detail and color richness. In contrast, our proposed “Replace Merging” strategy which directly replaces merged tokens with target tokens achieved the most precise edits while effectively preserving high-fidelity details. Consequently, all experiments presented in this paper utilize the “Replace Merging” strategy.





Figure 18. **Results of Additive Editing.** 100Editor can achieve high-quality editing results in additive editing under extra scenes.

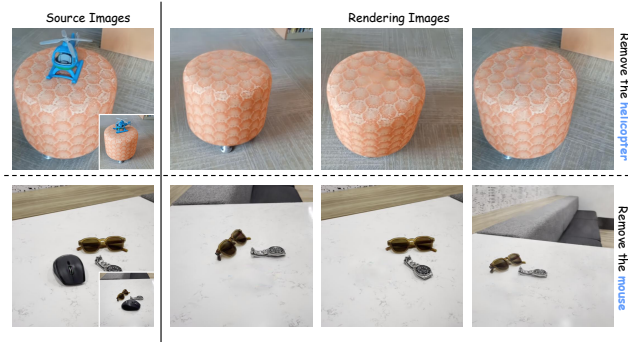


Figure 19. **Result of Subtractive Editing.** In subtractive editing with additional scenes, 100Editor can accurately achieve object deletion within the scene.

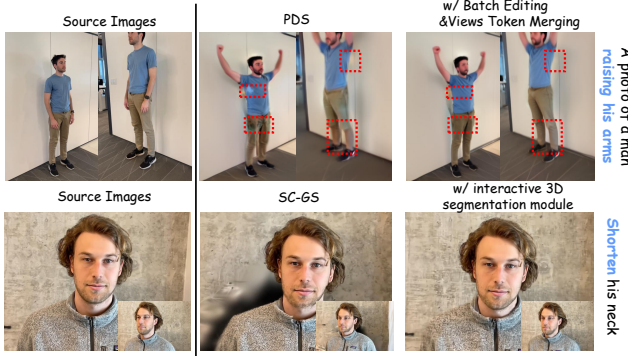


Figure 20. **Non-rigid Editing Comparison.** Compared to PDS, adopting Batch Editing and Views Token Merging can enhance editing consistency. For SC-GS, employing an Interactive 3D Segmentation Module can prevent irrelevant backgrounds from being affected.

## C.5. Ablation Studies

Fig. 27 highlights the effectiveness of the global token in achieving multi-view editing consistency. If the global token is not utilized during the Views Token Merging process, obvious visual defects arise, disrupting multi-view color consistency. For example, the edited bear’s head exhibits noticeable color discrepancies across different viewpoints,

Table 5. **Comparison of Batch Scalability.** 100Editor supports multi-view editing with larger batch sizes while achieving more efficient memory usage.

Batch	VRAM[GB]				
	GaussianEditor[11]	GaussCtrl[63]	DGE[10]	EditSplat[35]	100Editor
1	6.05	16.27	<b>5.42</b>	15.35	5.53
10	-	18.21	8.39	-	<b>7.12</b>
20	-	OOM	11.70	-	<b>7.27</b>
40	-	OOM	15.44	-	<b>7.56</b>
50	-	OOM	18.09	-	<b>7.70</b>
60	-	OOM	OOM	-	<b>7.84</b>
100	-	OOM	OOM	-	<b>8.42</b>
150	-	OOM	OOM	-	<b>9.13</b>
200	-	OOM	OOM	-	<b>9.85</b>
250	-	OOM	OOM	-	<b>10.56</b>
300	-	OOM	OOM	-	<b>11.27</b>

leading to artifacts and blurring in the final rendered images. This result underscores the critical role of the global token in preserving multi-view consistency.

Fig. 28 provides complementary quantitative evidence for the effectiveness of Token Merging. We evaluate this ablation across seven scenes and report the average CLIP-based editing scores, with the scene details summarized in Table 6. Compared with the variant without Token Merging, our full model achieves higher CLIP-based scores, indicating better semantic alignment with the target instructions after 3D optimization.

As shown in Fig. 31, we conducted ablation studies on the segmentation model. The comparison between these scenarios reveals that without leveraging a segmentation model, the 3D editing process undesirably affects irrelevant background regions. While the “Lang SAM” model, which relies on text-based prompts, can achieve reasonably accurate segmentation in most scenarios, it fails to execute fine-grained segmentation tasks; for instance, an input prompt for “left arm” results in the segmentation of the entire person. In contrast, our approach, which employs the SAM2 model, successfully facilitates fine-grained editing operations such as “turning the left arm black” and “making the eyes blue”. This consequently demonstrates that our proposed point-prompted interactive segmentation method achieves superior precision in segmentation operations.

Fig. 32 reports the critical impact of our segmentation strategy on the efficacy of local editing. With the editing instruction “Turn the truck into green”, removing either the 3D Segmentation module or the blending process results in a degradation of editing quality. Specifically, the absence of 3D Segmentation leads to noisy color artifacts in background regions outside the intended target. Conversely, removing blending introduces numerous irrelevant green floaters in the surrounding space. Therefore, within our interactive 3D segmentation module, the integration of 3D Segmentation and blending is essential for suppressing artifacts and achieving precise local edits with sharp bound-



Figure 21. **Complex Editing Cases.** Results of performing more complex editing operations in more challenging scenes.

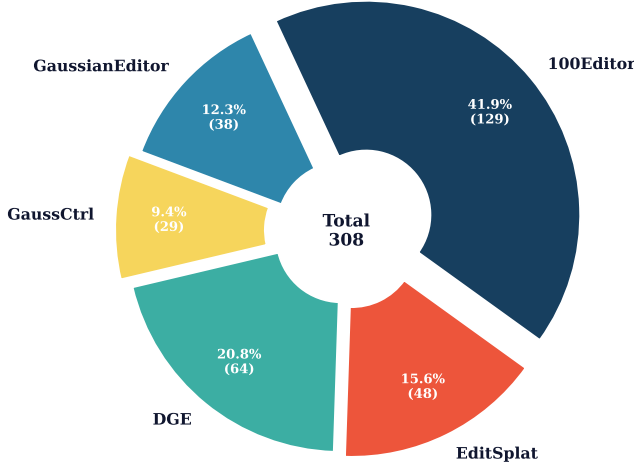


Figure 22. **Overall Results of the User Study.** In the user study, 100Editor garnered the highest percentage of votes, demonstrating its ability to produce editing results that are more publicly acceptable.

aries.

## D. 3D Editing Software

In this section, we present our designed 3D editing software by reporting its key functions, including data preprocessing, 3DGS visualization, 3D reconstruction, 3D editing, and image fitting.

### D.1. Data Preprocessing

When users click on the “Init” option in the top tab, they can enter the data preprocessing module, which specifically includes the functions of video processing, sparse point cloud reconstruction, and sparse point cloud visualization.

**Video Processing** After clicking the “processing” button, users can perform video processing operations, as shown in Fig. 33. Users can click the “Video Path” button in the menu bar to select the video file that needs to be processed. Then, users can click the “Output Path” button to select the location for the images after frame extraction. Click the

“Process” button to perform the video frame extraction operation.

**Sparse Point Cloud Generation** After clicking the “convert” button, users can perform the operation to obtain a sparse point cloud, as shown in Fig. 36. When the user clicks the “Source” button, they can select the relevant folder, which must contain an “input” folder with images from various perspectives. In addition to the traditional “SfM” operation, this software also implements the “vggt” operation. Users can choose the acquisition method by clicking the “SfM” or “VGGT” option. For the “SfM” operation, users can click the “Browse” button to select the corresponding colmap path; checking the “Use GPU” option will enable GPU for processing. For both methods, users only need to click the “colmap” button to obtain the sparse point cloud.

**Sparse Point Cloud Visualization** After performing the “showing colmap” operation, users can visualize the sparse point cloud, as shown in Fig. 37. Users can click the “Browse Sparse” button to select the folder containing the sparse point cloud to be visualized. Clicking the “Show” button will display the sparse point cloud in a pop-up window via Open3D, and clicking the “Stop” button will end the visualization of the sparse point cloud.

### D.2. 3DGS Visualization

When users click the “load” option in the top tab, they can enter the “Load 3DGS Model” module, which mainly includes the functions of “change camera pose”, “edit 3DGS model Properties”, and “render video”.

**Loading 3DGS** Users can click the “Load” button to load 3DGS models. Specifically, local 3DGS model files can be selected for loading by clicking the “Browse” button. For incorporating additional scenes into the display, users click the “Add Scene” button to select and load the target scenes, as illustrated in Fig. 39. To remove a specific scene, the “Remove” option is triggered to delete the corresponding scene instance. Enabling the “SplitScreen” and

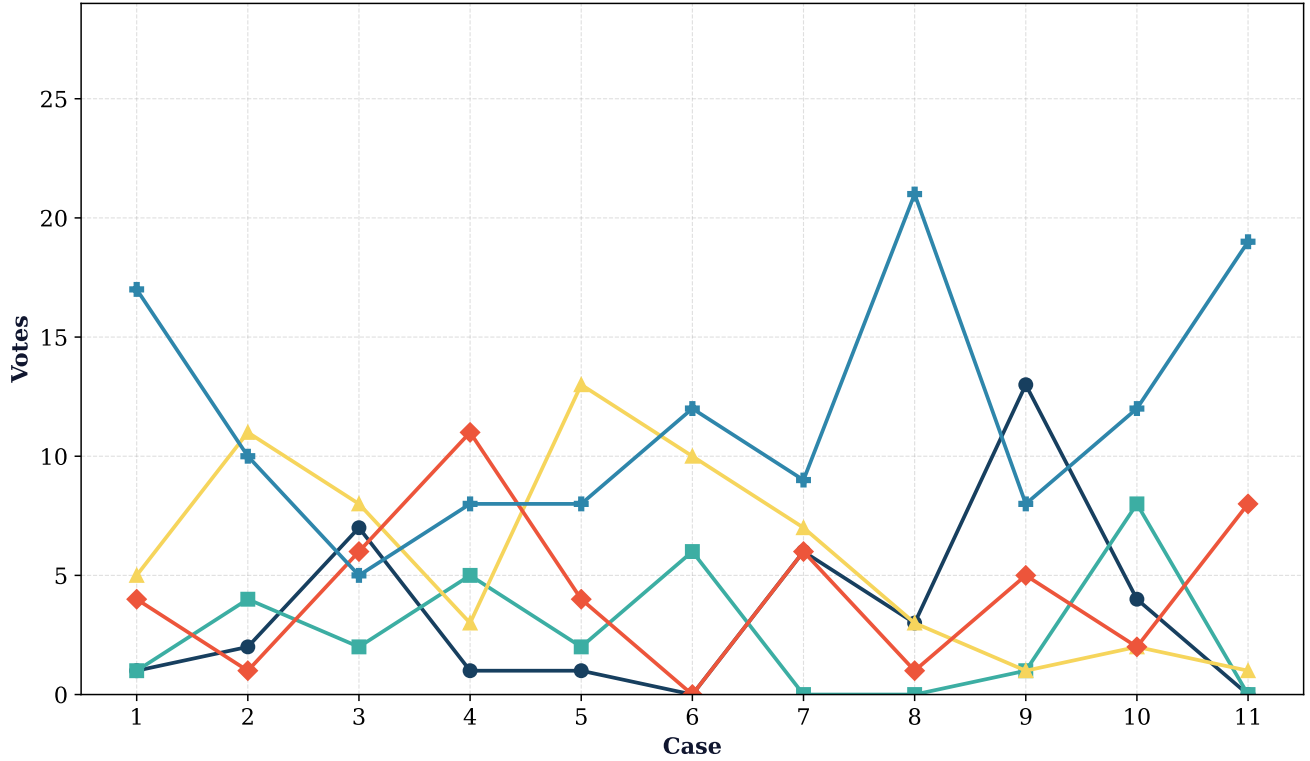


Figure 23. **The Vote Proportions for Each Example in the User Study.** Out of all eleven examples, 100Editor achieved the highest vote count in six instances, which demonstrates that 100Editor is capable of delivering higher quality editing results.

“Highlight Border” checkboxes enhances the clarity of visual feedback when multiple scenes are presented simultaneously, as shown in Fig. 40.

**Camera Pose Control** The software provides two distinct modes for changing camera pose using both mouse and keyboard inputs: “Orbit” and “First-Person Shooter (FPS),” as shown in Fig. 35. The “Orbit” mode offers an object-centric interaction paradigm, allowing the user to rotate the camera’s viewpoint around a central target. The rotation center can be reset by right-clicking on any point in the viewport; the new center is determined by the depth value at that pixel. The mouse wheel controls the orbital radius (zooming), while holding the wheel and moving the mouse pans the camera. In contrast, the “FPS” mode provides egocentric navigation. In this mode, the mouse controls the camera’s orientation, and the keyboard controls its translational movement. Furthermore, the user interface provides direct controls to modify camera parameters.

**3DGS Property Modification** By clicking the “edit” button, users can change the properties of the 3DGS model. The operation interface is shown in Fig. 38. Users can adjust the value of each variable via the slider, where the ad-

justed values serve as input variations for subsequent editing of the code block. Within the code block below, users are allowed to modify the properties of the current 3DGS representation, including its position, opacity, rotation parameters, and scaling factors. Furthermore, users can input text in the “Preset Name” field and trigger the “Save as Preset” button to store the current editing state of the 3DGS model.

**Video Generation** There are two ways of generating videos. One is to render by customizing the camera movement trajectory, and the other is to shoot by rotating 360 degrees around the scene, as shown in Fig. 34. Users can interactively define a camera path by specifying a series of keyframe camera poses at desired locations within the scene via an “Add Camera” function. Our software provides tools for camera management, including previewing the view from any selected camera (“See”), deleting a specific camera (“Remove”), and clearing the whole camera (“Clean Cameras”). The temporal smoothness of the camera motion is controlled by an “Interpolations” parameter, which dictates the number of intermediate frames to be generated between consecutive keyframes. Upon finalization of the path and parameters, the “Create Video” function initi-

## User Study for 3D Editing

The top row presents images from the source material. The subsequent rows showcase edited 3D scenes generated according to the specified editing instructions. Please select the best edited scene based on the following questions.

Assessment Criteria for Editing Quality:

1. **The target object is sufficiently edited;**
2. **Non-target regions remain unaffected.**

\*



Turn the **small sofa** into **red color**

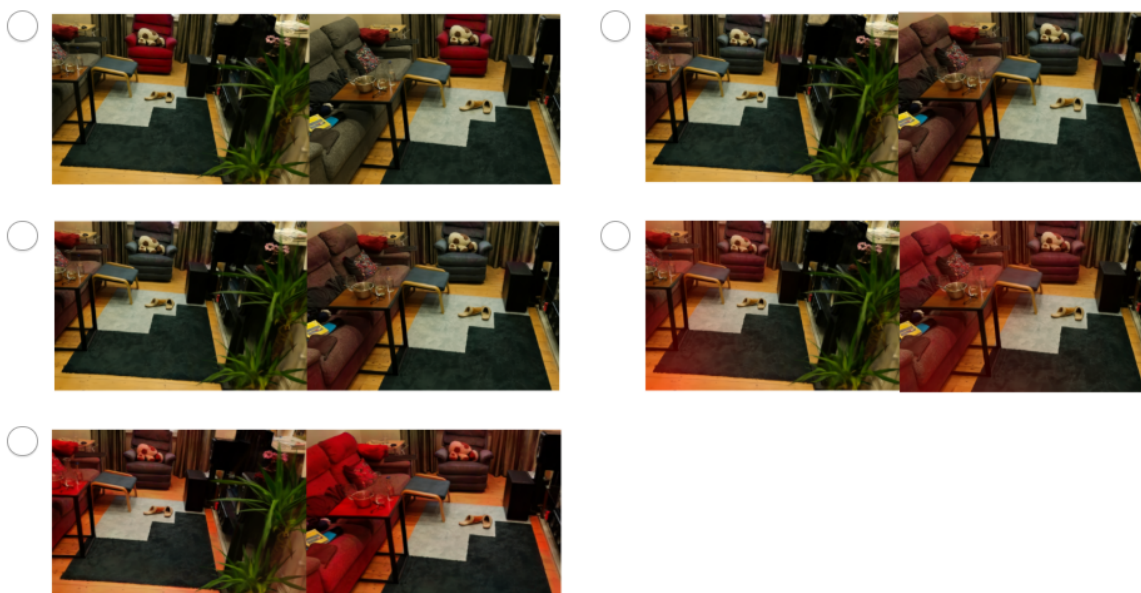


Figure 24. **User Study Questionnaire.** For each item in the questionnaire, the original multi-view image and the corresponding editing results from five models were presented. The model options were randomized for every question, and participants were required to select the option that exhibited the highest editing quality.

ates a generating process that synthesizes the final video sequence by traversing the interpolated camera trajectory. For the generation of 360-degree surround videos, users click

the “Video” button to adjust the relevant parameters, and then click the “Render” option to complete the generation.



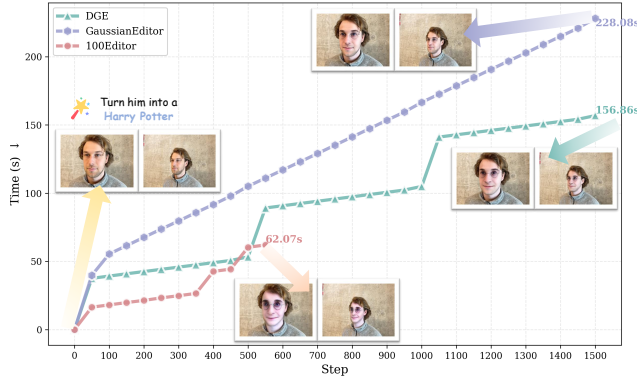


Figure 25. **Comparison of Time Across Steps.** 100Editor enables scene editing to be accomplished with the minimal number of editing steps and computation time, while achieving the high-fidelity editing result.

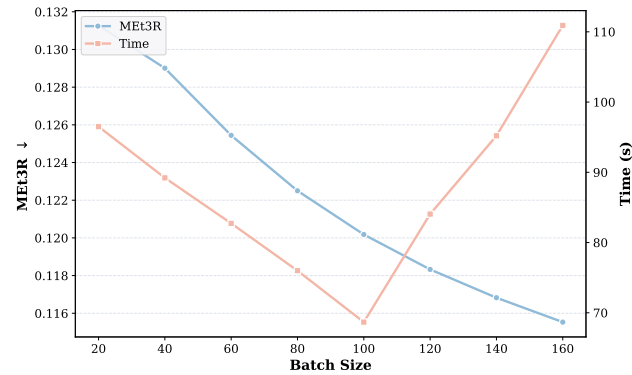


Figure 26. **Effect of batch size on multi-view consistency and runtime.** On the *garden* scene, larger batch sizes improve multi-view consistency, as indicated by lower MET3R, but increase runtime when the batch size exceeds 100.

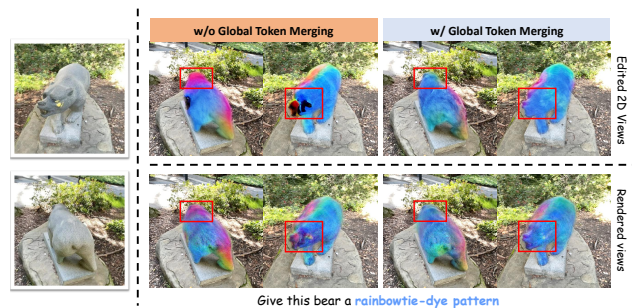


Figure 27. **Ablation Study of Global Token Merging.** Global Token Merging plays a crucial role in preserving multi-view consistency during scene editing.

### D.3. 3D Reconstruction

When users click on the “train” tab at the top of the page, they can enter the “3D Reconstruction” module, which is

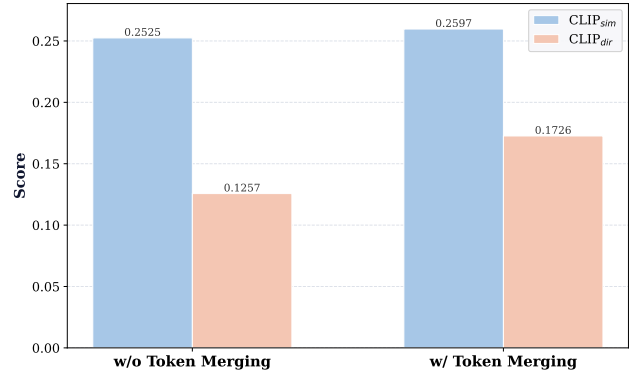


Figure 28. **Quantitative Ablation of Token Merging.** Enabling Token Merging improves the CLIP-based editing scores, indicating better text-aligned and higher-quality 3D editing results.

mainly used to visualize the training process of the 3DGS model, as shown in Fig. 44. Users can enter the training interface by clicking the “Training” button to achieve 3D reconstruction. First, users can choose a training framework between the “Origin” and “gsplat” options. Configuration then requires specifying the directories for training data input and result output, alongside the adjustment of various training hyperparameters. The process is commenced by clicking “Start Training”. Throughout the training, a monitoring panel displays real-time metrics, such as the loss function value, spherical harmonic (SH) coefficients, and the total number of Gaussians, allowing for observation of the reconstruction progress. The camera operation within this interface is identical to that found in the 3DGS model visualization module.

### D.4. 3D Editing

When users click on the “edit” tab at the top of the page, they can enter the 3D editing module. Users first need to click on the “Load” tab to enter the editing interface. Within this interface, the 3DGS model is imported using the “Browse ply” button, and the path for the training camera data is specified via the “Browse data” button. The module’s function is to support a range of editing tasks, such as non-rigid editing, additive editing, subtractive editing, and rigid editing.

**Parameter Settings** Users can select “Option” in the “Editor” menu to set editing parameters, as shown in Fig. 41. The software offers several editing modes, including “Editing”, “BatchEditing”, “Adding”, “BatchAdding”, and “Deleting”. Additionally, users can manually configure the operational parameters associated with each mode through the settings interface.

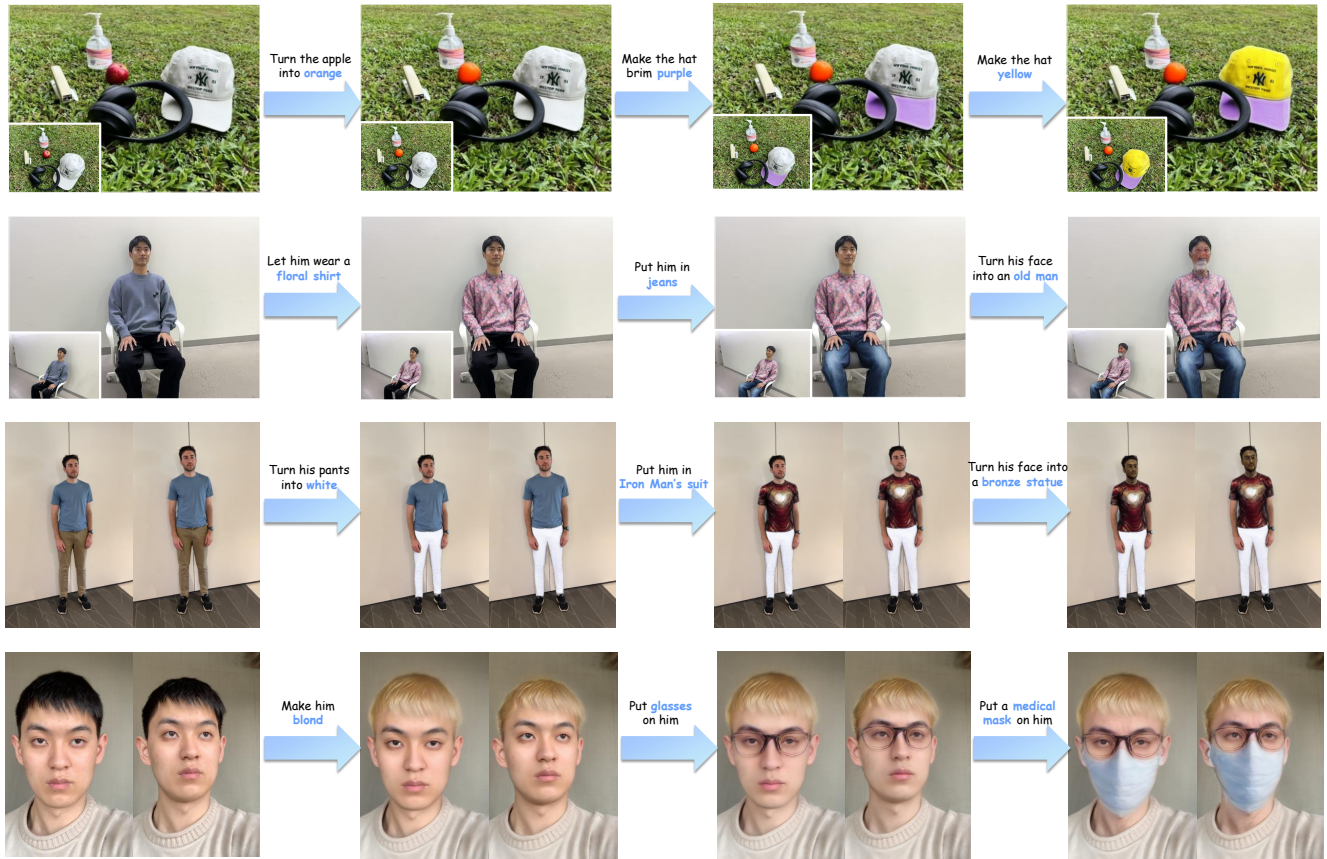


Figure 29. **Results of Additional Iterative Scene Editing.** 100Editor supports iterative editing, thereby achieving editing results that would originally require a complex instruction to define.

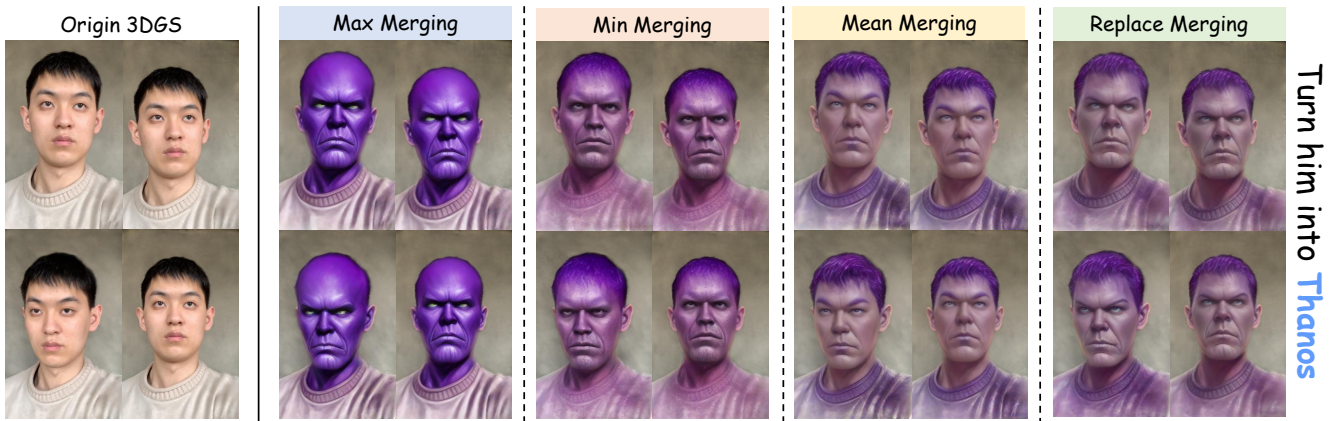


Figure 30. **The Impact of Different Views Token Merging Types on Editing Results.** The Max Merging results in over-editing, while the Min Merging leads to the generation of numerous artifacts post-editing. Furthermore, the Replace Merging we adopted achieves superior visual editing results compared to the Mean Merging.

**Semantic Editing** When users click the “text” button, users can edit specific areas of the 3D scene through input instructions in the interface, as shown in Fig. 43. The “SAM Option” section allows for the selection of different

Segment Anything Models (SAM). When the “Lang-SAM” model is chosen, users specify the target object for segmentation via a prompt in the “Seg prompt” input field. Alternatively, selecting the “SAM2 (image)” option enables



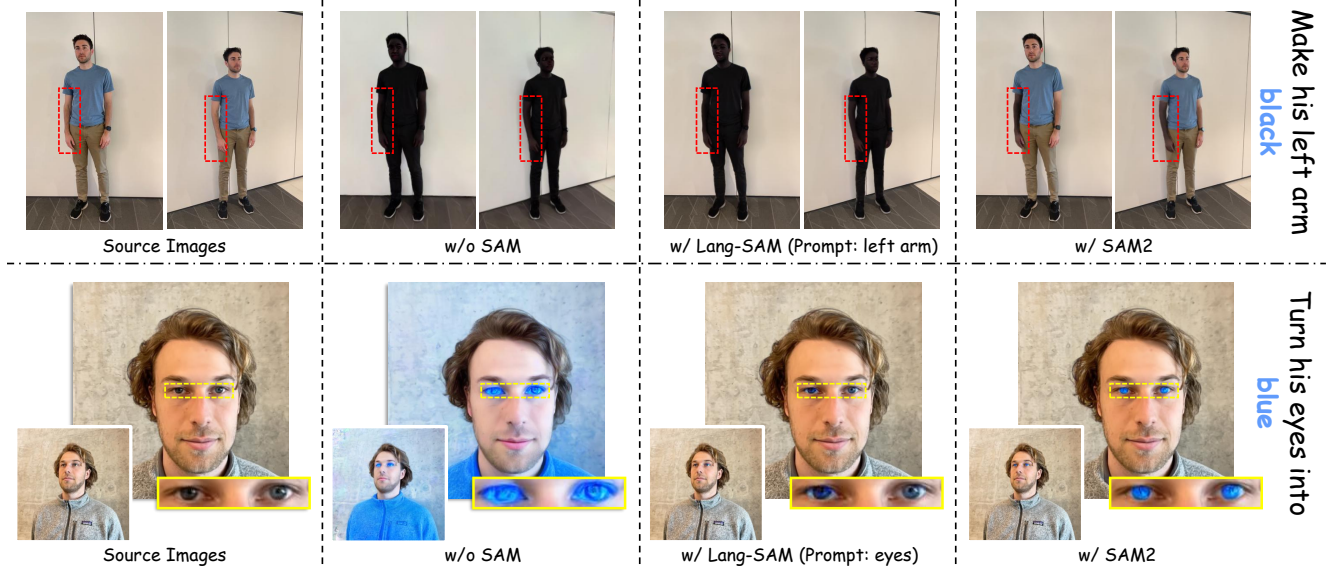


Figure 31. **Ablation of Segmentation Model.** The absence of a segmentation model results in the unintended alteration of irrelevant background regions in the edited output. Furthermore, utilizing the SAM2 model facilitates finer-grained editing compared to the Lang-SAM model.

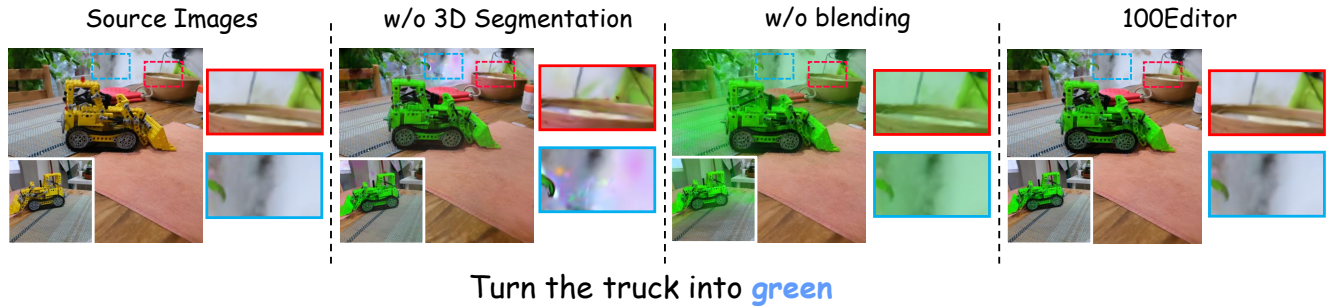


Figure 32. **Ablation Study of Local Editing.** Ablating the 3D segmentation module and the blending operation both result in notable artifacts in the final edited scene, which highlights their crucial role in local editing.

interactive segmentation, where users can segment the 3D object by setting positive and negative points directly on the current perspective image. Furthermore, users can select the “SAM(video)” option to perform segmentation on the entire 3D model using video segmentation. Finally, users click the “Save” button to select the path where the edited 3D model needs to be saved, and click the “Edit” button to start the 3D editing.

**Additive Editing** Users can enter the additive editing interface by clicking the “add” button and selecting the “Adding” option, as shown in Fig. 45. Within this interface, users first define a target area using either a “Rectangular Mask” or a “Sketch Mask,” as depicted in Fig. 42. After applying the mask in the right-hand panel, the user specifies an additive prompt in the “Edit one image” section on the left to modify the current view. Subsequently, in the

“Coarse Adding” module, users can generate a corresponding 3D asset and integrate it into the loaded 3DGS scene through depth adjustment. In the “Fine Adding” section, users input a final prompt, designate a save path, and click the “Add” button to complete the additive edit.

**Subtractive Editing** When the user clicks the “delete” button, they can remove the object in the 3D scene on the interface, as shown in Fig. 46. First, users input the prompt in the “Remove in the single image” field and click the “remove” button to preview the result of deleting the object from the current perspective. After stopping the preview with the “stop” button, users segment the object that needs to be removed in the “SAM Option” section. Finally, users select the relevant output path and click the “Delete” button to perform subtractive editing.

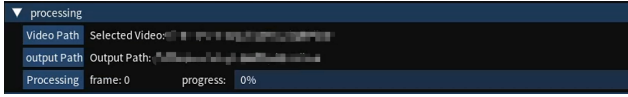


Figure 33. **Video Processing.** The user is enabled to perform frame extraction on the acquired scene video, which is a prerequisite for the subsequent reconstruction procedure.



Figure 34. **Generate the Video.** The user is enabled to generate videos either by orbiting the camera’s center of rotation or by following a user-defined camera trajectory.

**Non-rigid Editing** When the user clicks the “drag” button, they can drag the object in the 3D scene on the interface, as shown in Fig. 47. First, users need to segment the object to be dragged in the “SAM Option” area, and then click the “init graph” button to initialize. After the initialization is completed, users can perform a series of operations on key points by holding down the corresponding keys and using the left mouse button: hold down the “q” key to select key points, hold down the “e” key to expand key points, hold down the “z” key to translate and rotate key points, and hold down the “c” key to translate key points. The precise adjustment of these key points together achieves the final dragging effect of the object.

## D.5. Image Fitting

The “Other” menu provides functionality for fitting 2D images using the 3D Gaussian Splatting (3DGS) model. To begin, users select the “Fitting” option, specify the target image via the “Image Path” button, and configure the associated fitting parameters. Initiating the process by clicking “Start Training” visualizes the fitting procedure in the right-hand panel.

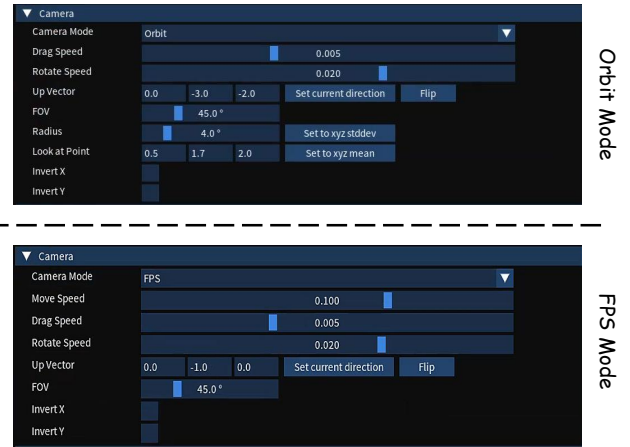


Figure 35. **Camera Mode Selection.** The software facilitates diverse interactive experiences by enabling users to select between two distinct camera modes.

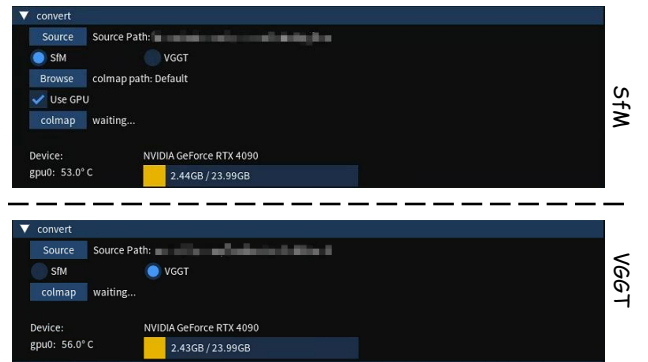


Figure 36. **Reconstruction of Point Clouds.** The corresponding point cloud can be derived by the user utilizing either COLMAP or VGGT processing.



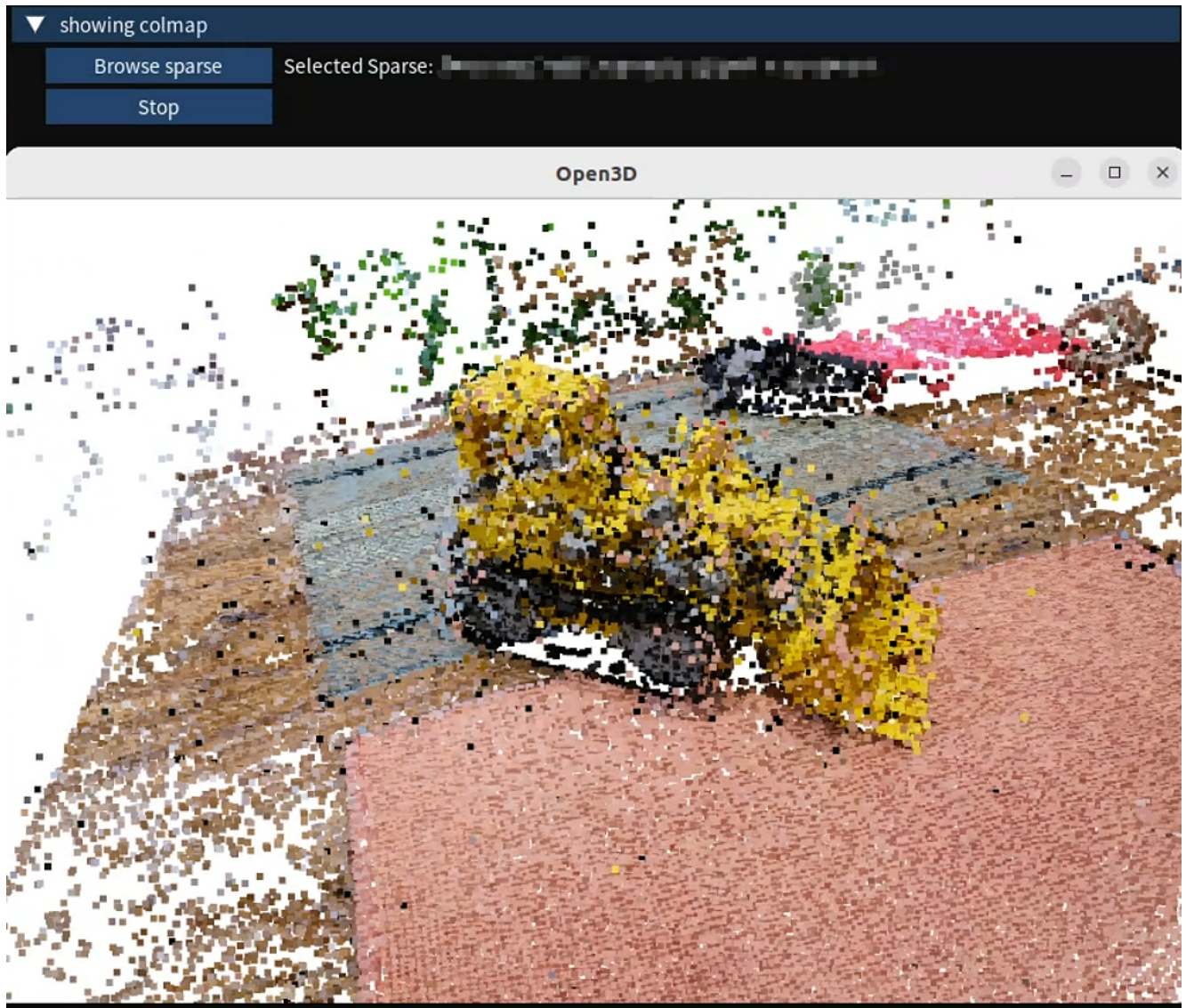


Figure 37. **Sparse Point Cloud Visualization.** The software enables users to engage in interactive visualization of the corresponding sparse point cloud.

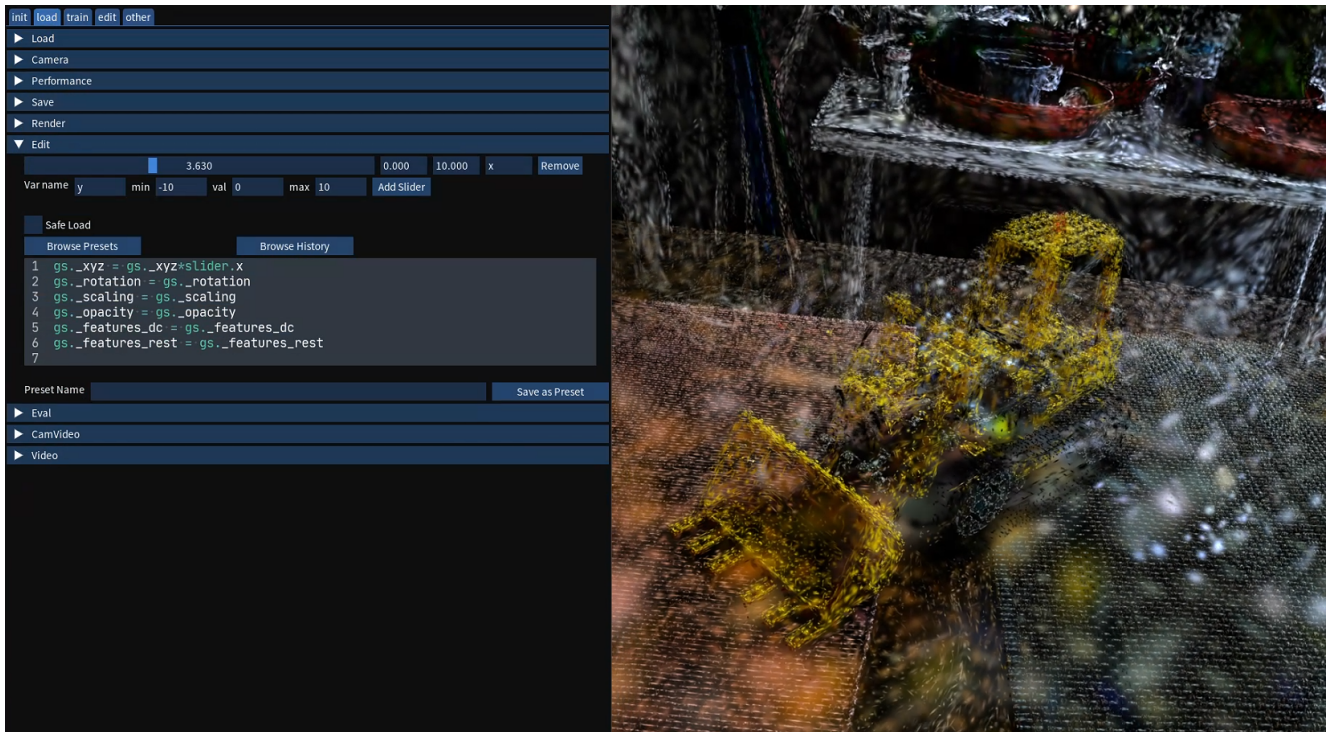


Figure 38. **Change 3DGS Properties.** The user can modify the 3DGS Properties and simultaneously observe the resulting changes within the interface.

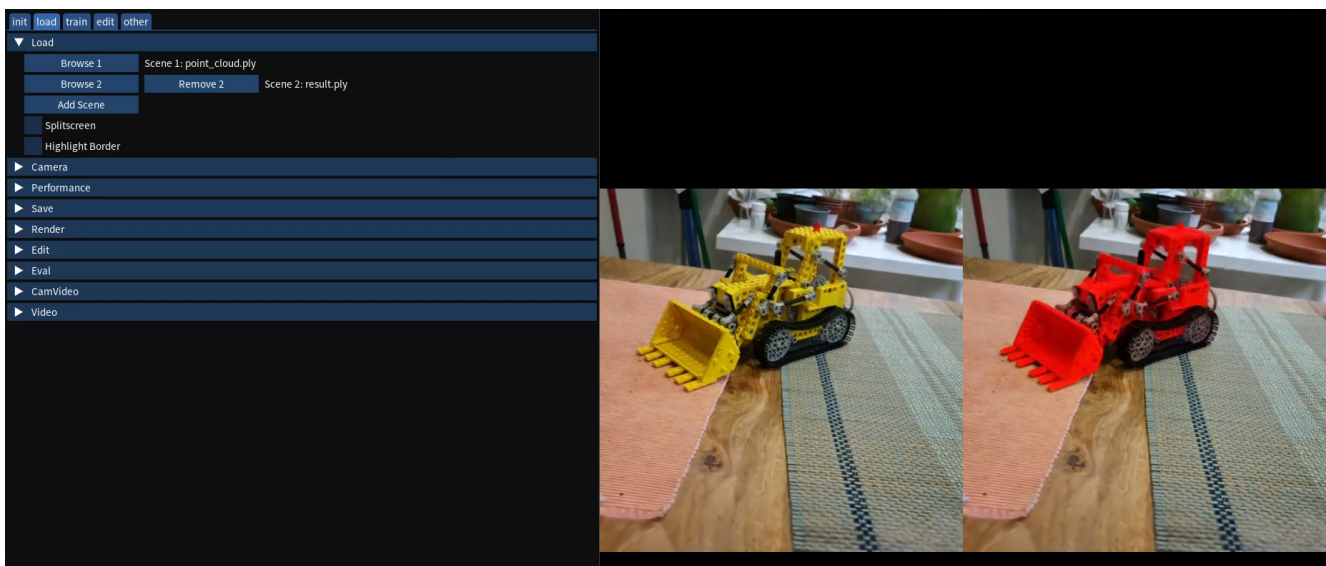


Figure 39. **The First Presentation of 3DGS in Software.** The user is enabled to present two 3DGS models concurrently via a split-screen display, thereby facilitating the observation of inter-scene differences.



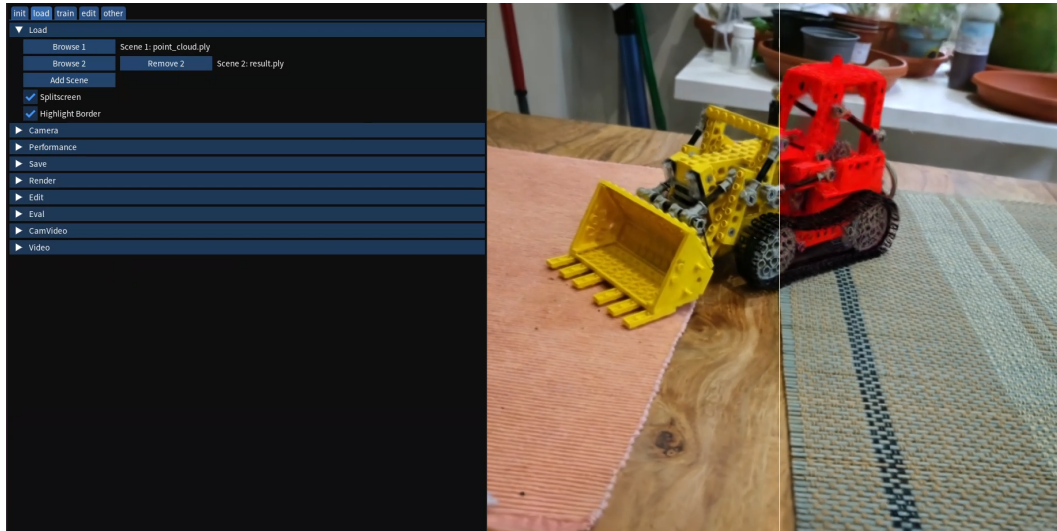


Figure 40. **The First Presentation of 3DGS in Software.** The user can merge the two rendered images presented in a split-view, which enables a more effective comparison of the disparities between the two outputs.



Figure 41. **Parameter settings.** The user is enabled to tune the relevant parameters throughout the editing procedure via the graphical user interface (GUI).

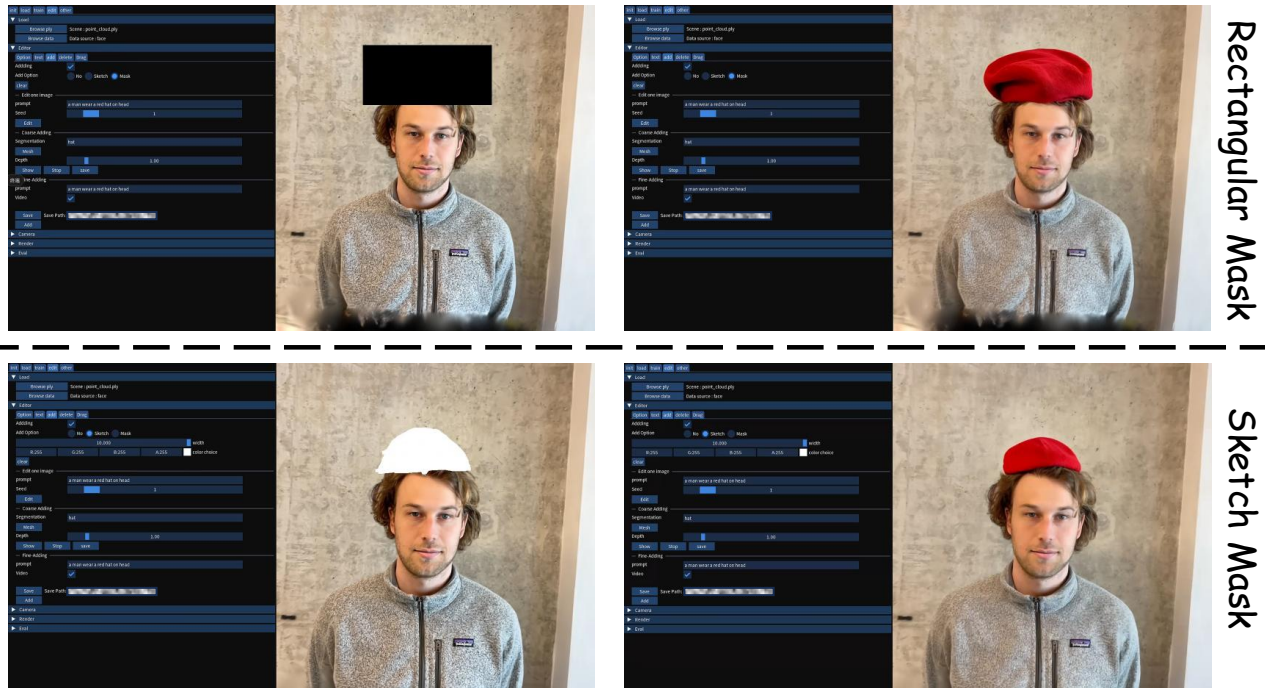


Figure 42. **Single View Adding.** The user is enabled to introduce relevant objects into the corresponding rendered image either through a rectangular mask or via a freehand sketch input.

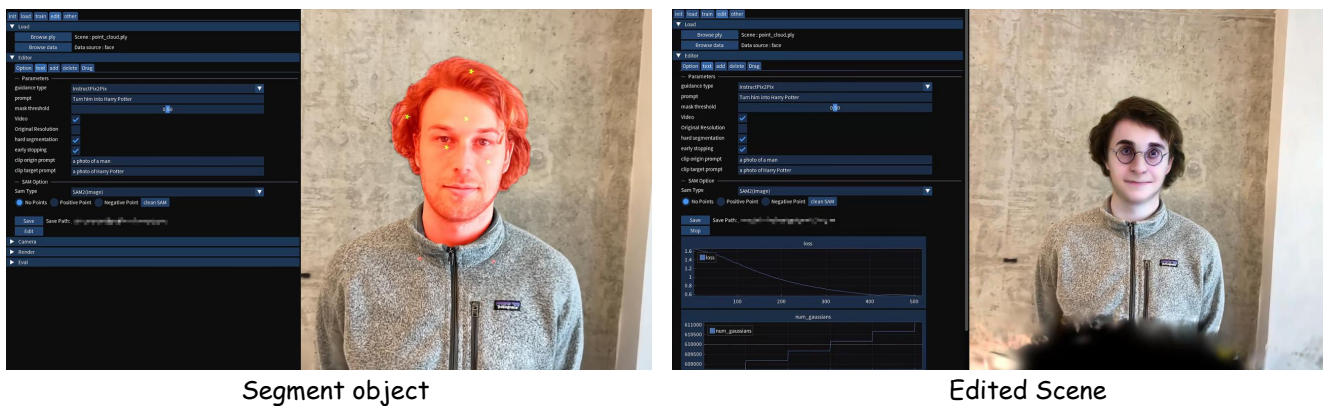


Figure 43. **3D Editing in software.** The procedure begins with the user selecting the target region for editing and specifying whether to employ the CPS algorithm. Subsequently, during the 3D editing process, the user can monitor the editing progression and the evolution of the corresponding 3DGS parameters.



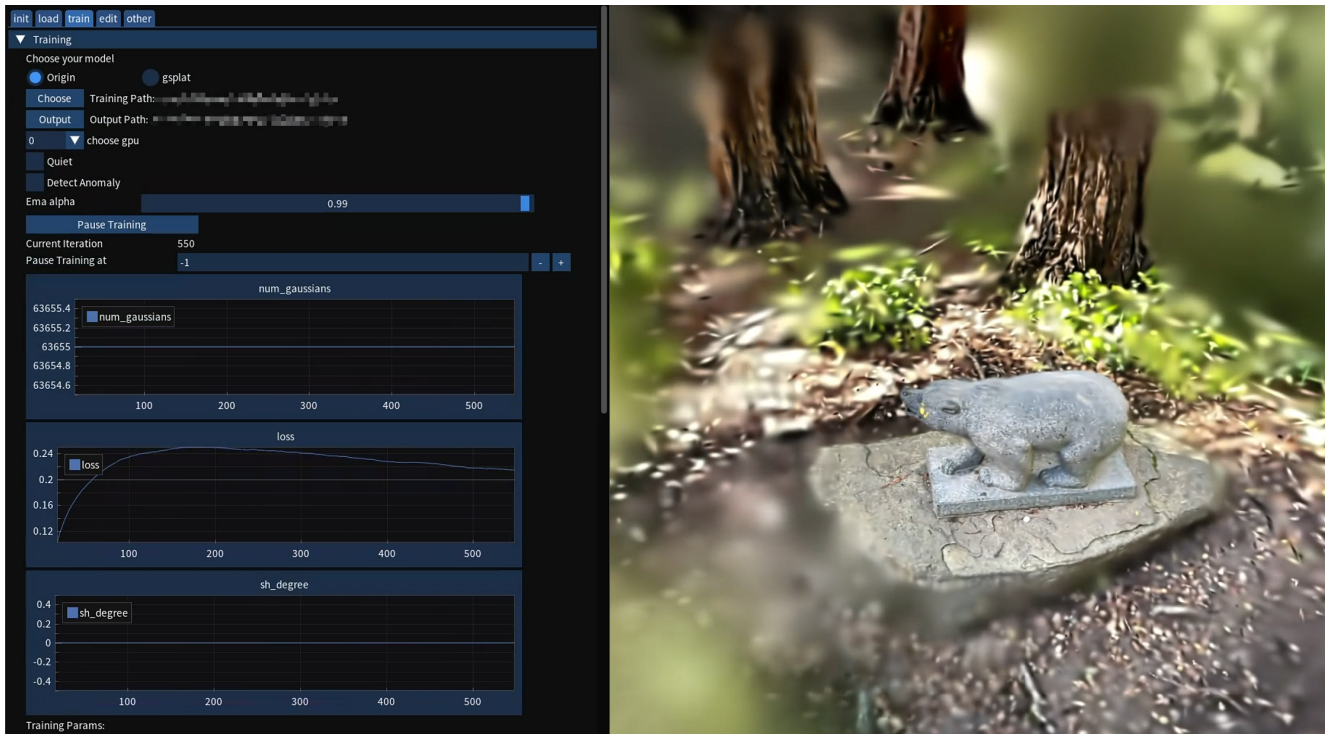
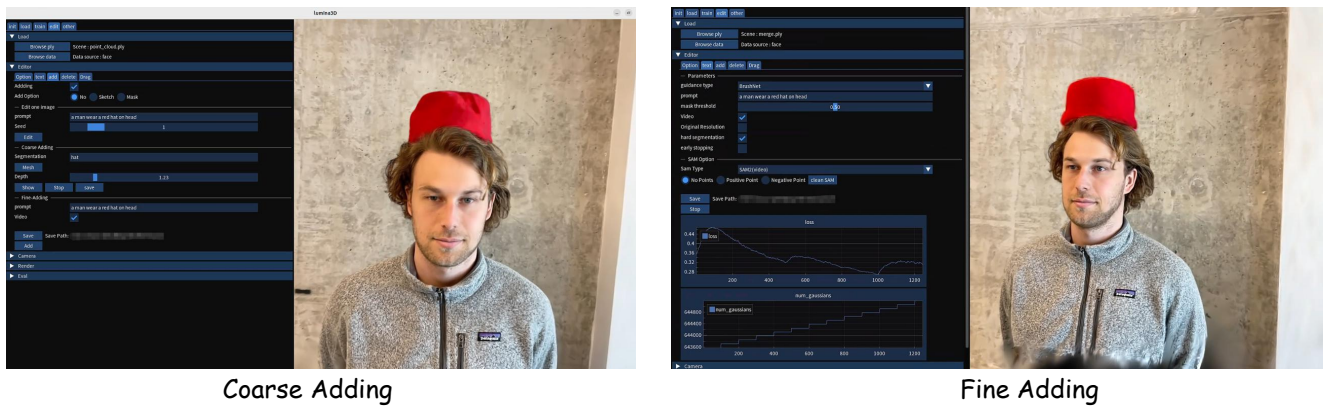


Figure 44. **3DGS Training in Software.** 3DGS training is initiated by the user selecting the designated directory containing the scene data. During this training process, the user can monitor the progression of the training and the evolution of relevant parameters.



Coarse Adding

Fine Adding

Figure 45. **Additive Editing in Software.** The user can first integrate the generated coarse object representation into the source scene, followed by a subsequent optimization process applied to the scene.

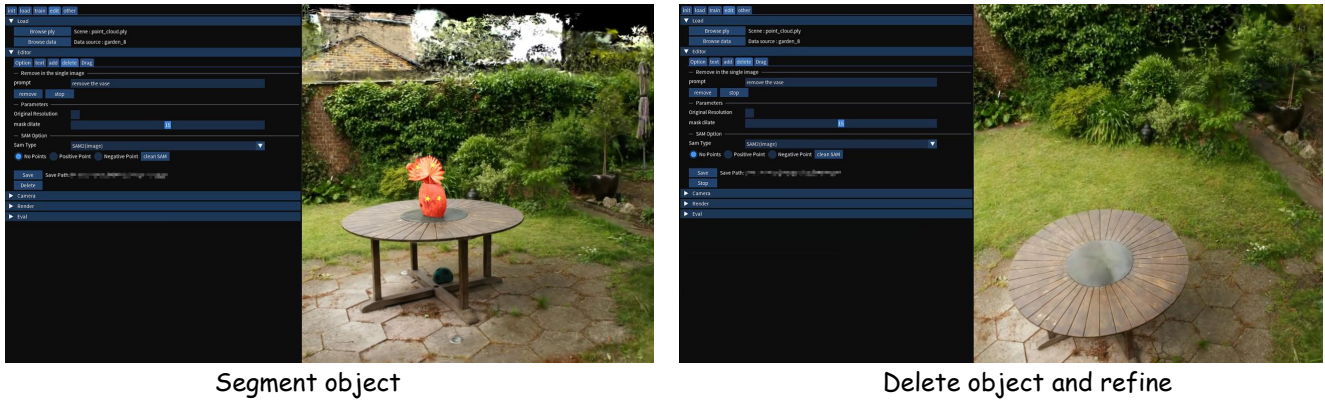


Figure 46. **Subtractive Editing in Software.** The user first selects the object to be deleted, completes the preliminary scene object removal, and subsequently subjects the affected region to inpainting processing.

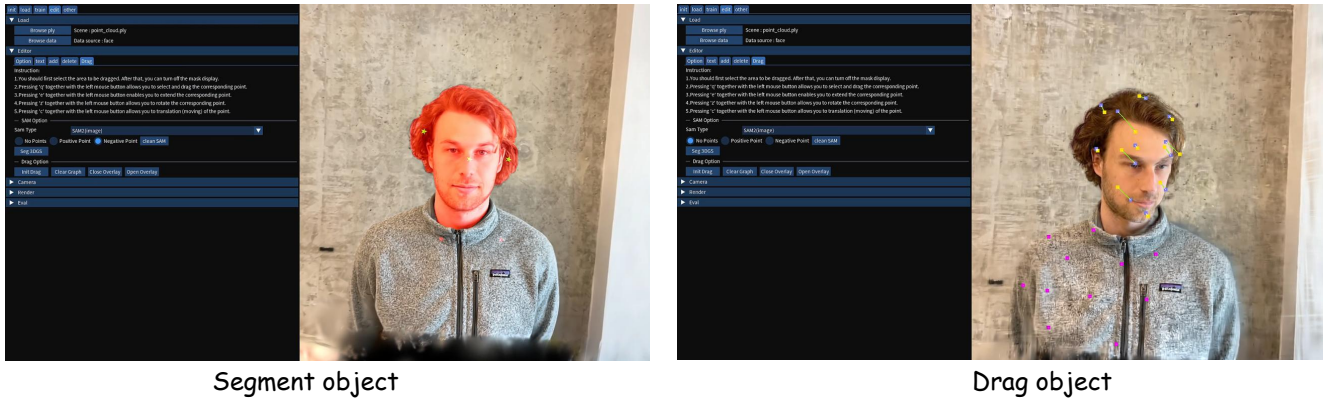


Figure 47. **Non-rigid Editing in Software.** The interaction begins with the user selecting the target region for dragging. Subsequently, control points are selected and manipulated via mouse movement to perform the dragging maneuver on the corresponding object.

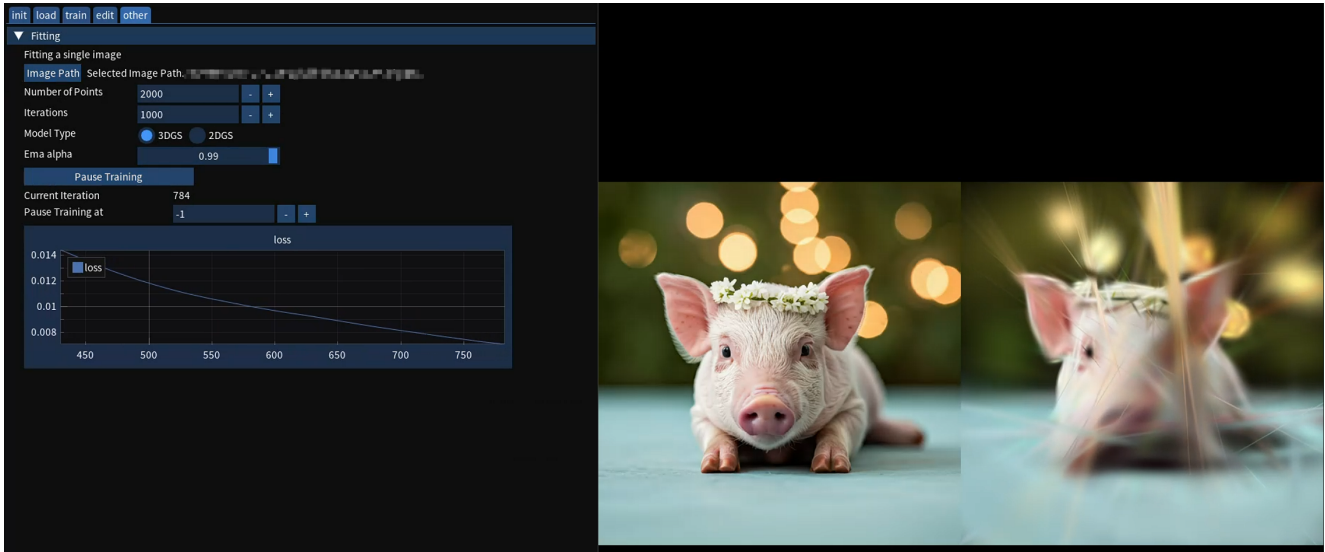


Figure 48. **Fitting one image.** The user can use either 3DGS or 2DGS to fit a single input image.

Table 6. **Details of the descriptions used for the CLIP Score.** For each editing instruction, we designed corresponding source and target descriptions to measure the associated CLIP score.

Scene	Source Description	Target Description	Editing Instruction
face	a photo of a face of a man	a photo of a face of a Kevin Durant	Turn his face into Kevin Durant
face	a photo of a face of a man	a photo of a face of a LeBron James	Turn his face into LeBron James
person-small	a photo of a man	a photo of a man wearing a suit	Make him wear a suit
person-small	a photo of a man	a photo of a iron man	Turn him into iron man
fangzhou-small	a photo of a face of a man	a photo of a face of a Tim Cook	Turn him into Tim Cook
fangzhou-small	a photo of a face of a man	a photo of a face of a Zombie	Turn him into Zombie
bear	a photo of a bear statue in the forest	a photo of a grizzly bear in the forest	Turn the bear into a grizzly bear
bear	a photo of a bear statue in the forest	a photo of a brown bear in the forest	Turn the bear into a brown bear
garden	a photo of an outdoor garden	a photo of an outdoor garden in autumn	Make it autumn
garden	a photo of an outdoor garden	a photo of an outdoor garden in winter	Make it winter
stump	a photo of a stump	a photo of a stump in the desert	stump in the desert
stump	a photo of a stump	a photo of a stump in fire	Make it fire
kitchen	a photo of a truck on a table	a photo of a red truck on the table	Turn the truck into red
kitchen	a photo of the truck on a table	a photo of a wooden truck on the table	Make this truck wooden



Table 7. **Detailed Instructions used for 3D scene editing.** For each editing instance, we design a corresponding editing instruction. However, for methods leveraging inversion-based 2D image editors, such as GaussCtrl[63], we explicitly formulate both source descriptions and target descriptions for every instruction to ensure fairness.

Scene	Editing Instruction	Source Description	Target Description
face	Turn his face into Kevin Durant	a photo of a face of a man	a photo of a face of a Kevin Durant
face	Turn his face into LeBron James	a photo of a face of a man	a photo of a face of a LeBron James
person-small	Make him wear a suit	a photo of a man	a photo of a man wearing a suit
person-small	Turn him into iron man	a photo of a man	a photo of a iron man
fangzhou-small	Turn him into Tim Cook	a photo of a face of a man	a photo of a face of a Tim Cook
fangzhou-small	Turn him into Zombie	a photo of a face of a man	a photo of a face of a Zombie
bear	Turn the bear into a grizzly bear	a photo of a bear statue in the forest	a photo of a grizzly bear in the forest
bear	Turn the bear into a brown bear	a photo of a bear statue in the forest	a photo of a brown bear in the forest
garden	Make it autumn	a photo of an outdoor garden	a photo of an outdoor garden in autumn
garden	Make it winter	a photo of an outdoor garden	a photo of an outdoor garden in winter
stump	stump in the desert	a photo of a stump	a photo of a stump in the desert
stump	Make it fire	a photo of a stump	a photo of a stump in fire
kitchen	Turn the truck into red	a photo of a truck on a table	a photo of a red truck on the table
kitchen	Make this truck wooden	a photo of the truck on a table	a photo of a wooden truck on the table