NeuSEditor: From Multi-View Images to Text-Guided Neural Surface Edits

Supplementary Material

Figure 7. Our network architecture retains information about the identity of the original scene, including background and foreground details, while simultaneously learning the edited render and geometry. Text prompt: *put him into a suit*

7. Network Architecture Details

As shown in Fig. 7, our network architecture simultaneously captures three critical components: (1) the background scene context, (2) the source input's foreground elements, and (3) the target edited scene composition. The architecture comprises three specialized renderers: a background renderer, a source foreground renderer, and a target foreground renderer, as depicted in Fig. 8. In this section, we elaborate on the details of each component and their implementation. Identity in this work refers to the composite information captured by the source renderer and background renderer during stage 1 (see Sec. 4).

Background Renderer For unbounded scenes, computing a Signed Distance Field (SDF) for the background is impractical; thus, we employ density fields instead. The background renderer operates with its own hash grids and includes subnetworks for implicit geometry and color rendering. The implicit geometry subnetwork processes hash encodings of a spatial point to produce a density value σ and a feature vector $\mathcal{F}$. This feature vector, combined with directional information, is then input to the renderer subnetwork to generate color values. To effectively model the unbounded background, we utilize the Inverted Sphere Parameterization [69] for volume rendering, yielding the background image $\mathcal{I}_{\text{bg}}$.

Source Renderer The source renderer is tasked with learning the foreground of the input scene. It leverages hash grids and subnetworks for implicit geometry and rendering. The implicit geometry subnetwork outputs an SDF value δ^{src} and a feature vector $\mathcal{F}$ for a given point encoding, which, along with directional information, are processed by the renderer subnetwork to determine the color. Using the NeuS volume rendering equations [59], we compute the foreground color information $\mathcal{I}_{\text{fg}}^{\text{src}}$ and the foreground mask $\mathcal{M}_{\text{fg}}^{\text{src}}$. The full source image $\mathcal{I}^{\text{src}}$ is then synthesized by combining the foreground and background images with the mask: $\mathcal{I}^{\text{src}} = \mathcal{M}_{\text{fg}}^{\text{src}} \cdot \mathcal{I}_{\text{fg}}^{\text{src}} + (1 - \mathcal{M}_{\text{fg}}^{\text{src}}) \cdot \mathcal{I}_{\text{bg}}$.

Target Renderer The target renderer, designed for the edited scene, mirrors the source renderer's structure, employing hash grids, an SDF-based implicit geometry subnetwork, and a color renderer subnetwork. It also adopts the NeuS volume rendering equations to derive the foreground image $\mathcal{I}_{\text{fg}}^{\text{tgt}}$ and mask $\mathcal{M}_{\text{fg}}^{\text{tgt}}$ for the target scene. The full target image is composed as $\mathcal{I}^{\text{tgt}} = \mathcal{M}_{\text{fg}}^{\text{tgt}} \cdot \mathcal{I}_{\text{fg}}^{\text{tgt}} + (1 - \mathcal{M}_{\text{fg}}^{\text{tgt}}) \cdot \mathcal{I}_{\text{bg}}$. To preserve recognizable features from the original scene during the editing (see Sec. 4), the target renderer is initialized to approximate the source rendering and geometry. The target geometry network is conditioned on the source geometry network, guiding it to generate a geometry (semantically) proximate to the source. Meanwhile, the color network focuses on tuning and editing the source scene's colors while retaining its access to source colors.

By integrating these three renderers, our architecture facilitates the cohesive modeling of the background, the original foreground, and the edited foreground, enabling seamless scene editing.

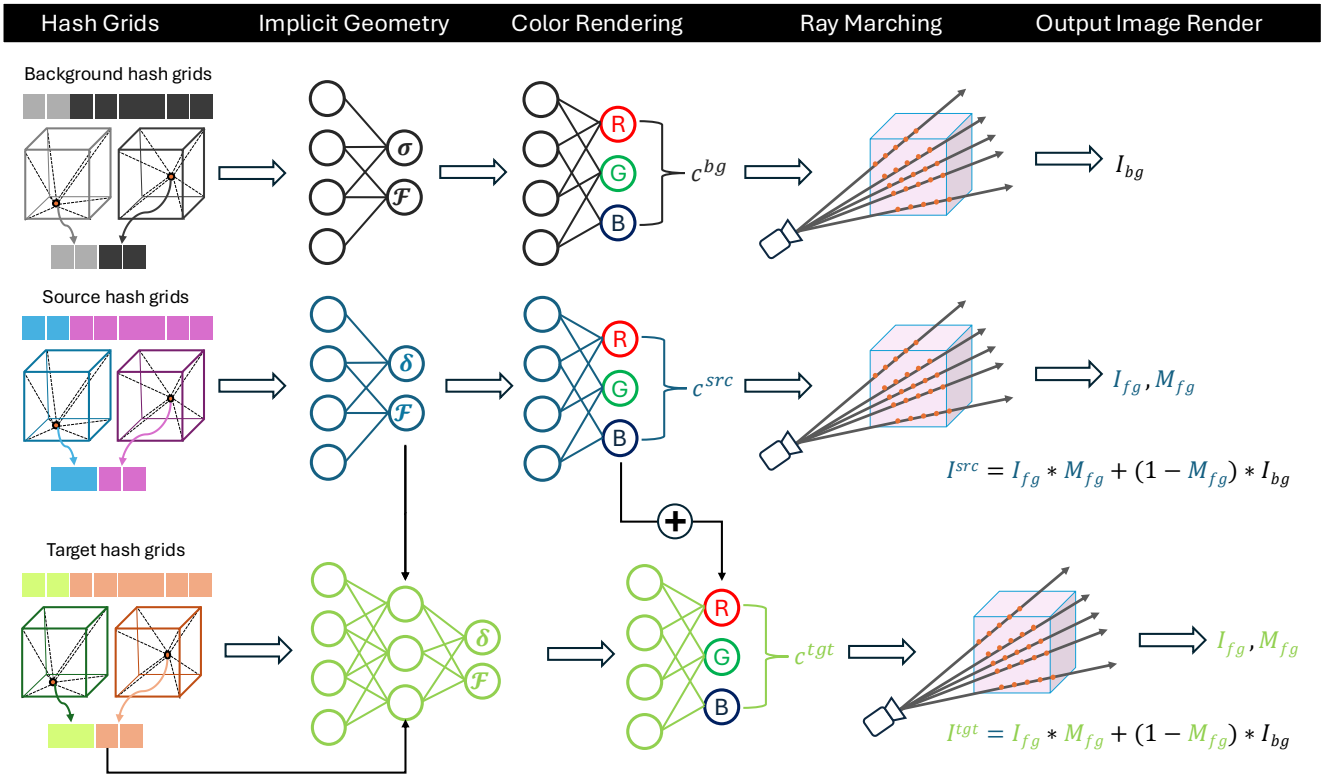

Figure 8. NeuSEditor integrates three dedicated renderers: a background renderer utilizing density fields, source and target foreground renderers employing SDF-based geometry with NeuS volume rendering. The target renderer is initialized from and conditioned on the source to preserve original scene features during editing.

8. Background editing

As discussed in Sec. 7, the background is modeled using radiance fields, with a separate set of hash grids as positional encodings. This background model utilizes its own geometry and renderer subnetworks. Our method also supports background editing. Similar to foreground editing, we still need to learn the input scene first to decompose it into foreground and background elements.

Unlike foreground editing, the use of "additive learning" techniques is less crucial for background editing. This is because background modifications are generally more generative in nature, aiming to completely change the scene rather than adjusting specific elements. Thus, to reduce computational demands, we directly edit the parameters of the background renderer instead of relying on "additive learning".

Fig. 9 shows the background editing capabilities of our approach. The top section displays two rows of images: the upper row shows the complete model render, while the lower row presents the isolated background render under different text prompts. These results demonstrate that our method can effectively modify the background while preserving the integrity of the foreground. The lower section compares our method with three recent approaches. Notably, despite explicit background specifications and various prompt engineering attempts, competing methods persistently modify foreground elements. In contrast, our approach preserves the foreground by exclusively optimizing background parameters. Thus, the explicit separation of background and foreground renderers further enhances our control over edits.

Metric	Backbone	IN2N	PDS_{NERF}	PDS_{GS}	Ours
clip ↑	ViT/16	0.317	0.318	0.336	0.334
	ViT/32	0.318	0.312	0.341	0.319
lpips ↓	Alex	0.748	0.668	0.737	0.533
	VGG	0.697	0.615	0.658	0.500

Table 5. The top row shows text-image similarity, and the bottom shows perceptual similarity between the input and edited scenes.

To quantitatively evaluate our background editing, we utilize both CLIP and LPIPS metrics. The CLIP score measures text-render alignment, while the LPIPS distance assesses the preservation of foreground details. To ensure an unbiased evaluation of the LPIPS distance, we trained a separate model solely to render the foreground components of the scene. Ideally, a high CLIP score combined with

Figure 9. Background editing results. Top row shows the full model render, while the bottom row displays the edited background under various text prompts. Our method effectively modifies the background while preserving the foreground integrity, outperforming competing methods that alter foreground details.

a low LPIPS distance indicates that modifications are restricted to the background. As shown in Tab. 5, our method achieves competitive text-image alignment while maintaining a lower LPIPS distance compared to other approaches. These results confirm that our edits are effectively confined to the background, contributing positively to the overall text-image alignment.

9. Avoiding mode collapse

Mode collapse is a common issue in generative AI pipelines (e.g. GANs, GPTs, text-to-3Ds) where the generator learns to produce a limited set of (or similar) outputs, ignoring the full diversity of the target data distribution. Attentive readers may have observed from Figs. 1 and 5 that our method is capable of generating diverse edits using the same prompt ("*make it a church*"). However, like most generative tech-

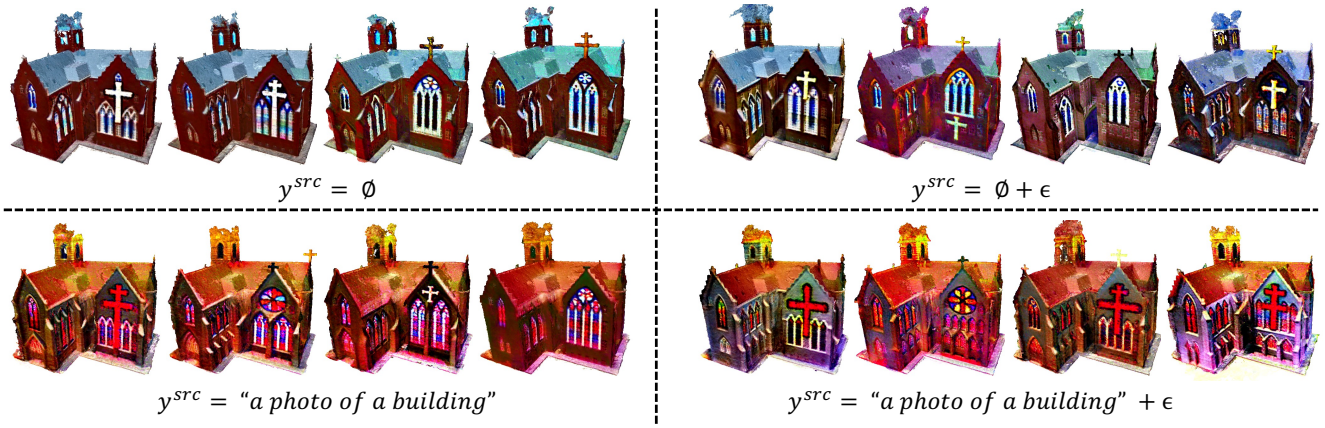

Figure 10. This figure shows textured mesh reconstruction results under varying text prompts. The top-left quadrant depicts results without a source prompt (*null* prompt), the top-right quadrant illustrates results with a *null* prompt and slight perturbations to the source text embeddings. The bottom-left quadrant demonstrates results with a descriptive source prompt, while the bottom-right quadrant presents results where small perturbations are applied to the same descriptive source text embeddings.

niques, text-guided editing approaches are prone to mode collapse. To mitigate this issue, we propose two simple yet effective techniques that leverage source prompting.

Eq. (11) shows that our model can consistently apply edits with or without the source prompt. However, including the source prompt allows users to guide the editing process towards exploring different modes of the distribution. Additionally, to improve distribution coverage, we introduce a straightforward technique that involves injecting a small amount of noise into the source prompt embedding. By utilizing this noise, users can steer the network to generate varied edits that align closely with the target text.

Fig. 10 provides qualitative evidence supporting the effectiveness of our proposed strategies for improving distribution coverage. The figure demonstrates that appending a descriptive source prompt effectively shifts the mode of the distribution. Furthermore, introducing a slight perturbation to the embedding of the source prompt encourages greater diversity in the reconstructed outputs.

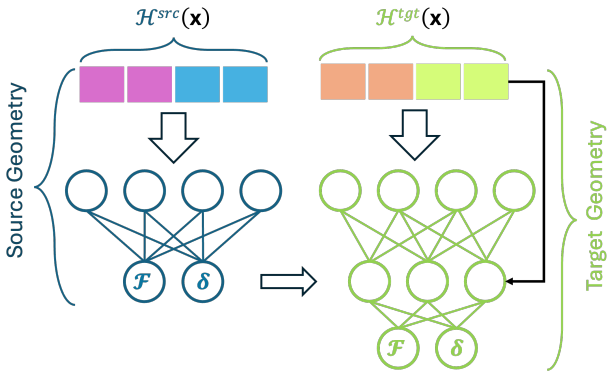

Figure 11. Source and target geometry networks.

10. SDF gradient computation

We have observed that, during editing, the numerical computation of the SDF gradient (via *finite differences*) results in cleaner and smoother geometry compared to using the analytical gradient via *torch.autograd.grad*. Fig. 11 shows the geometry networks for both the source and target. These networks process 3D points using hash encodings combined with MLPs, outputting the SDFs and corresponding geometric feature vectors for the source and target scenes. The *true* gradient for the source (identity) SDF can be computed using the following equations:

$$\delta^{\text{src}}(\mathbf{x}), \mathcal{F}^{\text{src}}(\mathbf{x}) = \mathbf{G}^{\text{src}}(\mathbf{x}) = \text{MLP}^{\text{src}}(\mathcal{H}^{\text{src}}(\mathbf{x})) \tag{20}$$

$$\nabla\delta^{\text{src}}(\mathbf{x}) = \lim_{h\to 0} \frac{\delta^{\text{src}}(\mathbf{x}+h) - \delta^{\text{src}}(\mathbf{x}-h)}{2h} \tag{21}$$

where $\mathbf{x}$ is the 3D query point position, $\mathcal{H}^{\text{src}}$ represents the hash encodings of the query point, $\mathbf{G}^{\text{src}}$ denotes the MLP-based geometry module for the source scene, and δ^{src} and $\mathcal{F}^{\text{src}}$ are the respective output SDF and features from the geometry module. As depicted in Fig. 11, the target geometry is conditioned on the outputs of the source geometry network. Thus, the solution for the *true* target (edited) SDF gradient can be computed as follows:

$$\delta^{\text{tgt}}(\mathbf{x}), \mathcal{F}^{\text{tgt}}(\mathbf{x}) = \text{MLP}^{\text{tgt}}(\mathbf{G}^{\text{src}}(\mathbf{x}), \mathcal{H}^{\text{tgt}}(\mathbf{x})) \tag{22}$$

$$\nabla\delta^{\text{tgt}}(\mathbf{x}) = \lim_{h\to 0} \frac{\delta^{\text{tgt}}(\mathbf{x}+h) - \delta^{\text{tgt}}(\mathbf{x}-h)}{2h} \tag{23}$$

where $\mathbf{x}$ is the 3D query point position, $\mathcal{H}^{\text{tgt}}$ represents the target hash encodings of the query point, MLP^{tgt} is the MLP-based feature and geometry extraction module for the target scene, which utilizes knowledge from the source geometry. The terms δ^{tgt} and $\mathcal{F}^{\text{tgt}}$ represent the respective output SDF and features of the target scene. For both the source

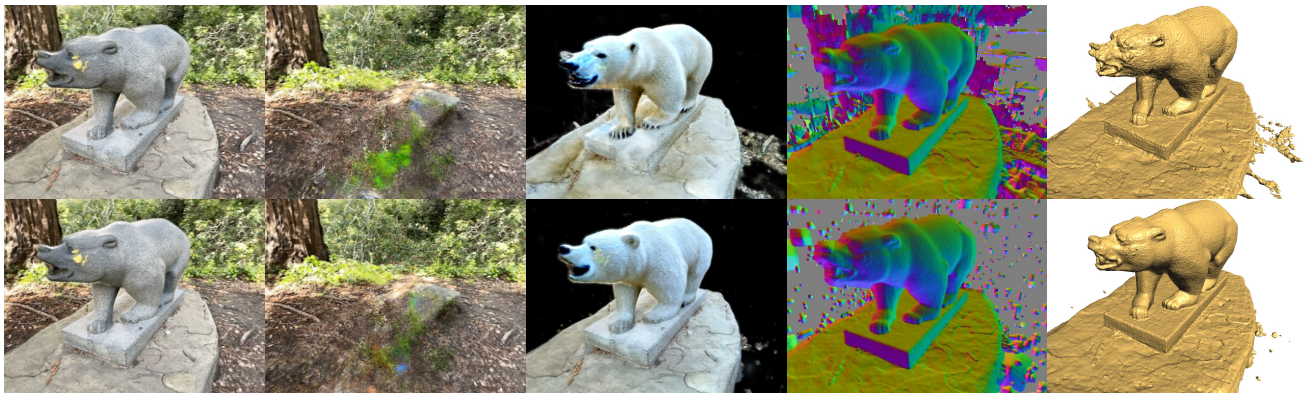

Figure 12. Experiments conducted on the IN2N dataset (bear scene) demonstrate both analytical and numerical solutions for SDF gradient computations. From left to right, the figure demonstrates the full identity rendering, background rendering, edited foreground rendering, normalized SDF gradient output from the network represented as a normal map, and the corresponding mesh rendering. The top row shows SDF gradients computed using the analytical solution, while the bottom row utilizes the numerical solution.

and target scenes, we *numerically* approximate the *true* gradient by setting h to a very small positive value.

Metric	Backbone	Analytical	Numerical
clip ↑	ViT/16	**0.298**	0.294
	ViT/32	**0.290**	0.285
lpips ↓	Alex	0.292	**0.201**
	VGG	0.353	**0.262**

Table 6. Gradient type evaluation using CLIP and LPIPS.

Our experiments show that numerical SDF gradients outperform analytical gradients in capturing target geometry. Fig. 12 illustrates experiments with a bear scene prompted with "*turn the bear into a polar bear*". The numerical SDF gradient computation (bottom row), representing the output of the fusion module in Fig. 11, significantly enhances the network's ability to capture foreground details compared to the analytical gradient (top row) during the editing process. The normal map in the Fig. 12 is computed using the SDF gradient output of the target geometry. For mesh extraction, we utilized the marching cubes algorithm, and our experiments revealed that numerical gradients also contribute to cleaner surface reconstructions.

We quantitatively evaluate the editing quality of the experiment. Tab. 6 presents the evaluation of different gradient types for this 3D editing task. Our results indicate that both gradient types yield similar CLIP metrics, suggesting that the choice of gradient type does not significantly affect text image (target rendering) alignment. In contrast, the LPIPS distance is notably reduced when using numerical gradients compared to analytical gradients. As qualitative observations also suggest, we believe that this improvement is due to numerical gradients generating fewer floaters and yielding smoother geometric surfaces.

11. Dataset and benchmark details

Algorithm 1 Generate DTU Spherical Camera Poses

function DTUSPHERICPOSES(cams, n_steps)
  center ← [0, 0, 0]
  cam_center ← mean(cams)
  eigvecs ← eigenvectors(cams^T × cams)
  up ← eigvecs[:, 1]
  rot_dir ← cross(up, cam_center)
  max_angle ← max(arccos(cams, cam_center))
  poses ← [ ]
  for θ in linspace(−max_angle, max_angle, n_steps) **do**
    cam_pos ← cam_center · cos(θ) + rot_dir · sin(θ)
    look_dir ← (center - cam_pos).$norm$()
    side ← (look_dir × up).$norm$()
    up_vec ← (side × look_dir).$norm$()
    pose ← [side, up_vec, −look_dir, cam_pos] #SE(3)
    Append pose to poses
  end for
  return poses
end function

Spherical rendering setup. As discussed in Sec. 5.3, we rendered videos using spherical poses for the DTU scenes and the IN2N *bear* scene to ensure a fair evaluation. The DTU dataset contains scenes with 49 or 64 images, where the first 49 images are distributed on the same upper hemisphere. To compute the spherical rendering path of the DTU scene, we calculate the average distance from the camera poses to the center $(0., 0., 0.)$ and set it as the radius. Additionally, we assumed that the second eigenvector of the

camera positions aligns with the vertical "up" vector, given the predominant horizontal and (then) vertical distribution of the cameras. Full details are provided in Algorithm 1. For the *bear* scene, we followed a similar approach, leveraging colmap SfM [48] data. Here, we assumed the "up" vector to be normal of the dominant ground plane, with the scene center defined as the closest "intersection" point of the cameras' "look-at" rays.

Hyperparameter tuning of IN2N and PDS methods. We devoted significant effort to tune the hyperparameters of the IN2N and PDS methods to improve their results. Since these methods rely on nerfstudio [57], we ensured that camera optimizers were disabled, as we primarily used groundtruth poses. Additionally, all poses were included in the training set, with no views excluded for validation.

IN2N was found to be sensitive to hyperparameters and prone to catastrophic forgetting. To address this in Blender and DTU scenes, we trained each scene using three hyperparameter configurations: (1) default settings with *text guidance* of 7.5 and *image guidance* of 1.5; (2) increased *image guidance* of 2.5; and (3) reduced *text guidance* of 5.0. The best results were selected, with default settings used as a fallback in case all of them leading to degenerate cases. For IN2N-data, we used the hyperparameter values specified in its supplementary material and followed recommendations from GitHub discussions (1, 2).

For PDS methods, we converted all datasets to the *nerfstudio-data* format, as their dataparser requires this. Each experiment was run multiple times, and the best result was selected. PDS_{NeRF} was more prone to degenerate cases with monotonic colors. In cases where PDS_{NeRF} led to degenerate outputs, early stopping was considered.

For our method, we utilized analytical SDF gradients and a shorter identity learning phase (8K iterations at stage 1) in DTU and Blender scenes to better demonstrate our architecture's identity-preserving capabilities, whereas other methods were trained 30K iterations for identity learning.

User study results details. Tab. 7 shows the user study results for each experiment. For our results, we set $\lambda_{\text{PDS}} = 1$ and $\lambda_{\text{PE}} = 0.2$, and we use empty source prompts in all experiments. In the DTU scenes, our method and $\text{PDS}_{\text{Splat}}$ clearly outperform the other methods, while IN2N struggles with optimization, often leading to degenerate cases. PDS_{NeRF} produces excessive floaters, likely leading users to prefer our method and $\text{PDS}_{\text{Splat}}$.

In the Blender dataset, particularly in the *hotdog* and *mic* scenes, our method and $\text{PDS}_{\text{Splat}}$ again achieve superior performance, whereas PDS_{NeRF} struggles due to excessive floater generation. However, IN2N performs better than PDS_{NeRF} in these cases, likely because of its lower floater count. In the *ficus* scene, $\text{PDS}_{\text{Splat}}$ fails due to multi-view inconsistencies, IN2N struggles with producing the expected edits, and PDS_{NeRF} performs better aligning more closely with the input prompt.

In experiments using IN2N-data, IN2N achieves the best overall results, as expected, since it utilizes the same prompts, datasets, and hyperparameters found effective in the original Supp. materials. In the *person* scene, all methods generate clean renders compared to previous scenes. However, users prefer the results from $\text{PDS}_{\text{Splat}}$ and IN2N, likely due to their cleaner outputs and better preservation of human identity. In the *bear* scene, our method and IN2N produce comparable results, with users slightly favoring our method, likely because it is more effective at avoiding the "Janus artifact". $\text{PDS}_{\text{Splat}}$ suffers from multi-view inconsistencies, while PDS_{NeRF} produces monotonic colors.

12. Representation Choice

Implicit neural rendering approaches have achieved SOTA results in geometry reconstruction, as demonstrated by Neuralangelo [29], and in visual rendering quality, as shown by ZipNeRF [3]. Motivated by these successes, we adopt an SDF-based neural rendering approach. While our method draws inspiration from Neuralangelo's encoding strategy [29], the original implementation requires substantial GPU memory and 1–2 days of training, as it lacks the key CUDA optimizations introduced in Instant-NGP [36]. In contrast, our pipeline is optimized: the first stage (source learning) completes training in just 5–10 minutes. We also avoid blindly adopting all Neuralangelo's loss functions.

Although Gaussian Splatting (GS) [25] offers faster training, PDS_{GS} [26] shows suboptimal performance, with increased floaters and multi-view inconsistencies (main Fig. 6). Additionally, applying additive learning techniques [20, 70] to GS is non-intuitive. As noted in the main part, our identity representation requires only 28M floats for the source scene and 14M per edit, substantially less than typical GS parameter counts and consistent across scenes.

13. Choice of Phong vs. Normals/Depth Maps

While depth maps encode geometry, they are often overly smooth, failing to capture fine surface details critical for precise editing. Normal maps offer greater detail, but in our empirical observations, Phong shading yielded superior results. We believe the reason is that Phong-shaded images, unlike raw normal maps, resemble natural grayscale images, which better align with the training domain of the pretrained diffusion model [45]. This enables more effective supervision, enhancing edit fidelity and reducing artifacts like floaters (Fig. 5), as validated in ablations (Sec. 5.2).

#	Dataset-Scene	Keyword	IN2N	PDS_{NeRF}	PDS_{Splat}	Ours
1	DTU - Scan24	Church	0.20	1.37	1.68	2.76
2		Mosque	0.07	1.22	2.10	2.61
3		Castle	0.20	1.54	2.22	2.05
4		Disney castle	0.39	0.76	2.17	2.68
5		Lego	0.15	1.37	1.90	2.59
6		Barn	0.07	1.34	1.73	2.85
7	DTU - Scan65	Moustache*	1.73	0.71	0.90	2.66
8		Horned skull	0.17	1.00	2.02	2.80
9		Alien	0.63	0.71	1.98	2.68
10		Buddha	0.10	1.22	1.90	2.78
11		Clown*	1.98	0.39	1.10	2.54
12	DTU - Scan83	Suit	0.07	1.88	1.66	2.39
13		Bowtie*	0.07	2.02	1.85	2.05
14	DTU - Scan105	Suit	0.46	0.66	2.12	2.76
15		Tiger	0.63	1.83	1.59	1.95
16		Bowtie*	0.56	1.20	1.71	2.54
17	DTU - Scan106	Chickens	0.98	0.88	1.59	2.56
18		Crows	0.07	1.49	1.76	2.68
19	DTU - Scan110	Monk	0.88	0.22	2.34	2.56
20		Buddha	0.98	0.12	2.68	2.22
21		Snoop Dogg*	0.63	0.56	2.20	2.61
22	Blender - Hotdog	Bananas	1.27	0.22	1.85	2.66
23		Corns	0.90	0.56	1.68	2.85
24	Blender - Mic	Hair dryer	1.34	0.27	1.93	2.46
25		Pistol	1.22	0.07	2.22	2.49
26	Blender - Ficus	Cactus	0.24	0.98	1.85	2.93
27		Apple tree	1.98	1.37	0.07	2.59
28		Rose bush	0.59	1.51	1.07	2.83
29	IN2N - Person	Clown	2.05	0.15	2.46	1.34
30		Suit	2.76	0.73	1.95	0.56
31		Firefighter	2.29	1.07	2.17	0.46
32	IN2N - Bear	Grizzly bear	2.71	0.80	0.41	2.07
33		Panda	2.22	0.56	0.66	2.56
34		Polar bear	2.24	0.44	0.66	2.66

Table 7. User survey results across 34 experiments. NeuSEditor used a guidance scale of 350 for all experiments, except those marked with an asterisk (*), where a lower scale (100) was applied to encourage minimal changes (e.g. add a moustache/bowtie). We invite readers to compare these quantitative user survey results with the qualitative results shared on the survey page replica.