ConsistDreamer: 3D-Consistent 2D Diffusion for High-Fidelity Scene Editing

Jun-Kun Chen[1†] Samuel Rota Bulò[2] Norman Müller[2] Lorenzo Porzi[2]
Peter Kontschieder[2] Yu-Xiong Wang[1]
[1]University of Illinois Urbana-Champaign [2]Meta
{junkun3,yxw}@illinois.edu
{rotabulo,normanm,porzi,pkontschieder}@meta.com

Supplementary Material

This document contains additional analysis and extra experiments. The content of this document is summarized as below:

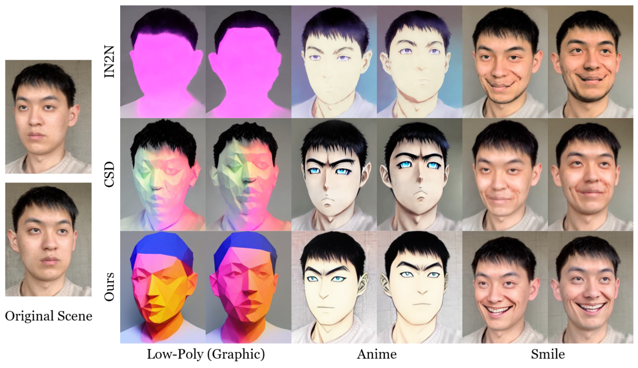

Figure B.1. Qualitative comparisons with baseline CSD on three tasks show that our ConsistDreamer achieves high-quality editing, outperforming both IN2N and CSD with more successful editing.

A. Supplementary Video (SV)

To better visualize our results and compare with baselines beyond static 2D images, we provide a supplementary video (SV) on our project page at *immortalco.github.io/ConsistDreamer*. We also include a short demo in this video, to enhance the understanding of 3D-consistent structured noise. The original size of the video is around 1.25GB, therefore we have to compress it to fit it in the upload size limitation of 200MB on OpenReview.

In the following sections, we use SV to refer to this supplementary video.

B. Comparisons with Additional Baselines

In the main paper, we compare our ConsistDreamer with IN2N [4] and ViCA [3]. In this section, we compare

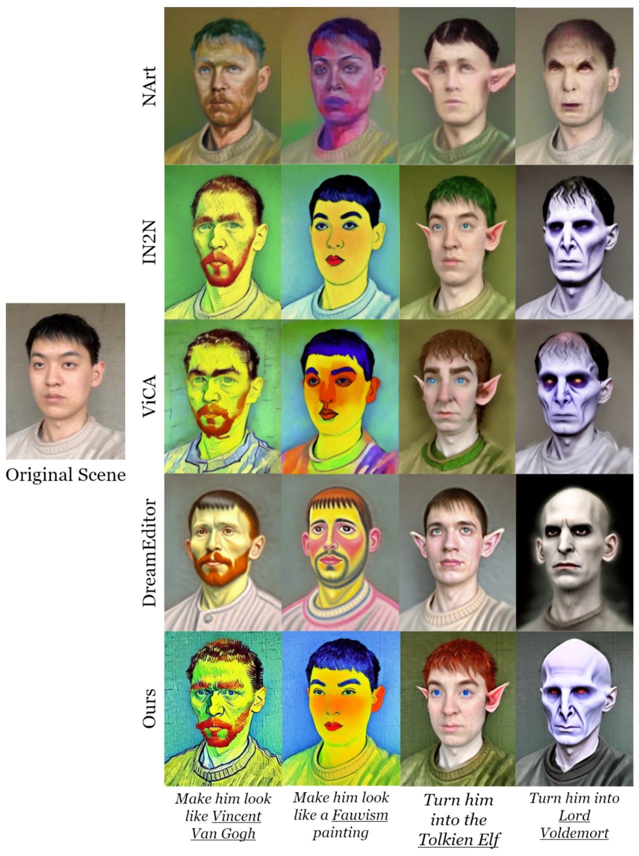
Original Scene
NArt
IN2N
ViCA
DreamEditor
Ours
Make him look like Vincent Van Gogh
Make him look like a Fauvism painting
Turn him into the Tolkien Elf
Turn him into Lord Voldemort

Figure B.2. Compared with DreamEditor, our ConsistDreamer achieves better editing, which not only follows and satisfies the given instructions, but also preserves as much content of the original scene as possible. On the contrary, DreamEditor completely edits the original person to another in all the tasks.

our ConsistDreamer with other baselines and provide some analysis. These methods either do not have publicly available code, or evaluate on the scenes which are not supported by NeRFStudio. Therefore, we could only compare our ConsistDreamer under the tasks used by them, with the provided visualizations from their papers or websites.

We also provide some comparisons in the video format of the baselines in SV.

B.1. CSD [6]

CSD is a method focusing on general consistent generation, including large image editing, scene editing, and scene generation. We compare our ConsistDreamer with CSD under three tasks shown on the website of CSD[1]: Low-Poly (Graphic), Anime, and Smile.

As shown in Fig. B.1 and SV, our ConsistDreamer significantly outperforms IN2N, which fails in the Low-Poly and Anime tasks, and has the side effects of adding beards

[1]https://subin-kim-cv.github.io/CSD/

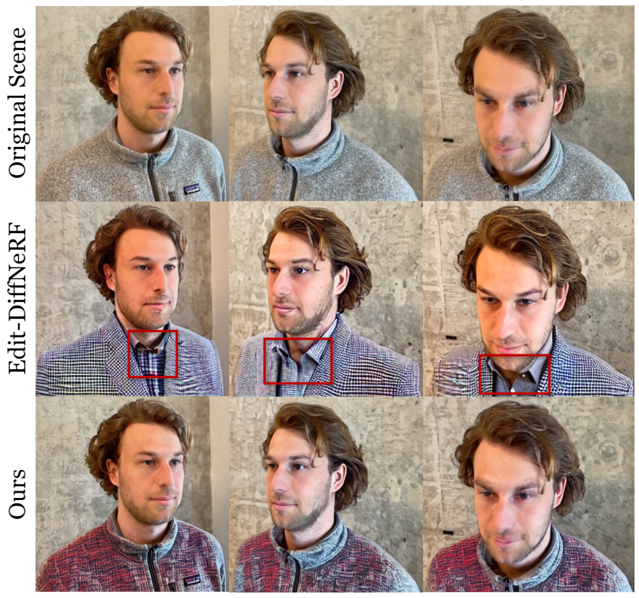

Figure B.3. Our ConsistDreamer achieves consistent editing in the checkered/plaid pattern (also visualized as smooth video in SV), while Edit-DiffNeRF has obvious inconsistency in the shape and texture of the collar.

in the Smile task. Compared with CSD, our editing in the Low-Poly task is more noticeable, with a successfully edited hair part. Our edited scene in the Smile task is the only one among all three to successfully show the teeth when smiling, while CSD's result contains strange muscles as if the person is keeping a straight face. In conclusion, our ConsistDreamer achieves more successful editing than CSD.

B.2. DreamEditor [18]

DreamEditor is another method focusing on scene editing, but with another diffusion model [11] instead of [1]. As NeRFStudio does not support the other scenes, we compare our ConsistDreamer with DreamEditor by comparing Fig. 3 in our main paper with Fig. 8 in [18].

Fig. B.2 presents the results in these tasks, along with other baselines in Fig. 3 in our main paper. It shows that our ConsistDreamer preserves most of the contents in the original scene while editing, *e.g.*, the shape of the head and face, and the shape and type of the clothes, minimizing the side effects of editing. DreamEditor, however, completely edits the person to another person, even in the Fauvism task, which is supposed to be only style transfer. This demonstrates that our ConsistDreamer achieves more reasonable editing than DreamEditor.

B.3. Edit-DiffNeRF [16]

Edit-DiffNeRF is another paper that also claims to successfully complete the checkered/plaid pattern. As they did not provide any code, we compare our ConsistDreamer with the images provided in their paper. As shown in Fig. B.3, our ConsistDreamer achieves consistent editing among all three views, while Edit-DiffNeRF's results are multi-view inconsistent, obviously shown in the collar part. The smooth video of our rendering result in SV also shows the consistency of our ConsistDreamer. These results validate that our ConsistDreamer archives significantly better consistency in checkered/plaid patterns, while Edit-DiffNeRF fails to achieve such consistency.

B.4. Instruct 3D-to-3D [5]

Instruct 3D-to-3D is a method focusing on style transfer of scenes. It uses LLFF and NeRF Synthetic (NS) scenes as editing tasks instead of the widely-used IN2N dataset. In contrast, we focus on editing more challenging and realistic scenes. In addition, as NeRFStudio and NeRFacto do not support LLFF and NS datasets well (more specifically, NeRFStudio does not support the LLFF dataset, and NeRFacto works well in real scenes but not in synthetic scenes like NS), we cannot compare with Instruct 3D-to-3D on these two datasets. Moreover, the code of Instruct 3D-to-3D is not publicly available. Therefore, we are unable to compare with Instruct 3D-to-3D.

B.5. Concurrent Works: GE [2], EN2N [12], And PDS [7]

[2, 7, 12] are three concurrent works. GE [2] and EN2N [12] achieve 3D editing through the same 2D diffusion model [1] and have some modifications in the pipeline or scene representation, while PDS [7] proposes another distillation formula and uses DreamBooth [11] for editing.

The comparisons against them are in Fig. B.4. Our ConsistDreamer generates high-quality editing results with brighter color and clearer textures, while all these concurrent works generate blurred textures, gloomy colors, and/or unsuccessful or unreasonable editing.

C. CLIP [9] Metrics In IN2N [4]

We provide the quantitative comparison with CLIP [9] metrics introduced in IN2N [4] in Tab. C.1. In all four ablation scenes, ours significantly and consistently outperforms IN2N in both metrics.

D. Implementation Details

D.1. Hyperparameters and Settings

In our experiments, we use the multi-GPU pipeline with $n = 3$ (4 GPUs in total), and surrounding views of $k = 5$ (1 main view and 12 reference views).

The learning rate of each component is shown below:

- NeRFacto [13]: 5×10^{-3} for field part, and 10^{-2} for proposal network part.
- LoRA-augmented diffusion model [1]: consistent with their original implementation (10^{-4}).
- Learnable 3D positional embedding: 2×10^{-3}.

All the views are resized to $3 : 4$ or $4 : 3$ according to their orientations. For landscape images (portrait images use the same setting with a flipped height and width), the diffusion model takes a surrounding view image input at 1152×864 (also $4 : 3$), with horizontal splitters at heights 6pix, and vertical splitters at widths 8pix. The sizes of the main view and reference views are 688×516 and 224×168, respectively. This setting is consistent with the original usage of diffusion models [1, 10] trained at 512×512, as our main view has a height close to it.

Consistent with IN2N [4], both MSE and LPIPS losses are used to train NeRF.

D.2. Viewpoints And Camera Trajectory

During the distillation process, we directly use the viewpoints provided in the original scene dataset, which is sufficient to cover the whole scene. In visualization, we use the provided camera trajectory for IN2N [4] dataset, and manually construct another camera trajectory for a smooth visualization for ScanNet++ [15] dataset.

D.3. Training Schedule

One standard full training contains $1,600$ epochs across multiple sub-stages. All these stages are explained below:

- Initialization Stage (Epoch $1 \sim 200$): Train diffusion [1] before NeRF fitting. We perform one diffusion training step in one epoch.
  - Early Bootstrap (Epoch $1 \sim 50$): Train the LoRA-augmented diffusion model to mimic the behavior of the original model with the augmented input of 3D positional embedding. The weight regularization loss of maintaining original behavior (detailed in D.8) is significantly higher. NeRF training has not started.
  - Bootstrap (Epoch $51 \sim 150$): Train the consistency-awareness of the LoRA-augmented diffusion model while keeping original behavior, at a similar importance with balanced weights.
  - Warming Up (Epoch $151 \sim 200$): Use the standard weights to balance the consistency loss and regularization, focusing more on consistency. This epoch generates sufficient images for the edited view buffer for NeRF fitting.
- Distillation Stage (Epoch $201 \sim 1600$): Train diffusion while fitting NeRF. In each of 4 epochs, we do 3 diffusion generation steps without training (to fill the edited view buffer), and only one diffusion training step. Here, the "noise level" means the mixture rate of the current NeRF (being edited) rendered image and the noise as the diffusion's input: full noise level means using only noise for generation (standard generation), while a 30% noise

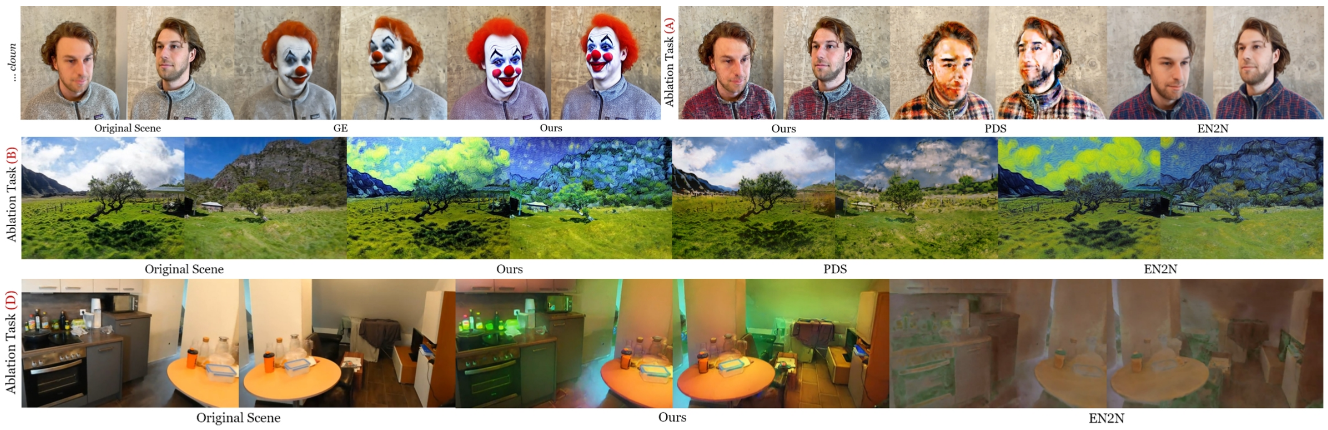

Figure B.4. Our ConsistDreamer outperforms all three concurrent works with brighter color, clearer textures, and better editing results matched with the editing instruction.

Method	Text-Image Direction Similarity (CTIDS) ↑				Direction Consistency (CDC) ↑			
	A	B	C	D	A	B	C	D
Ours	**0.0259**	**0.1679**	**0.1204**	**0.1268**	**0.5785**	**0.1735**	**0.3077**	**0.2878**
IN2N [4]	0.0099	0.1252	0.1163	0.1055	0.5106	0.1634	0.2900	0.1772

Table C.1. Our ConsistDreamer significantly and consistently outperforms baseline IN2N in CLIP [9] metrics over all four ablation scenes.

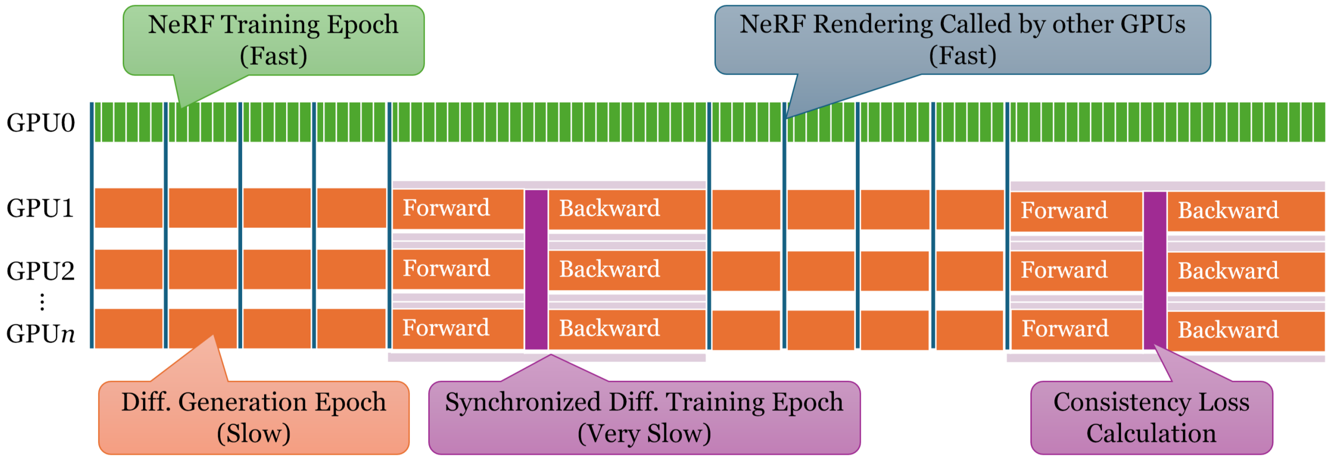
NeRF Training Epoch (Fast)
NeRF Rendering Called by other GPUs (Fast)
GPU0
GPU1
GPU2
⋮
GPUn
Forward
Backward
Forward
Backward
Forward
Backward
Forward
Backward
Forward
Backward
Forward
Backward
Diff. Generation Epoch (Slow)
Synchronized Diff. Training Epoch (Very Slow)
Consistency Loss Calculation

Figure D.1. Our multi-GPU training pipeline uses $n+1$ GPUs, one dedicated for NeRF fitting, and the others for diffusion training and generation.

level means the input image is the mixture of 30% noise and 70% rendered image.

- Full Noise Generation (Epoch 201 $\sim$ 500): The diffusion model is trained and used for generation at a full noise level to edit the views sufficiently regardless of NeRF.
- Pre-Annealing (Epoch 501 $\sim$ 600): The diffusion model is trained and used for generation with a noise level sampled from $[70\%, 100\%]$. It edits the views with a few references to the current NeRF, starting to refine the current NeRF.
- Annealing (Epoch 601 $\sim$ 1500): Following the idea of HiFA [17], the range of the noise level linearly anneals from $[70\%, 100\%]$ to $[10\%, 40\%]$. The NeRF will gradually converge to a fine-grained edited version.
- Ending (Epoch 1501 $\sim$ 1600): The diffusion model is trained and used for generation with a noise level sampled from the annealed range $[10\%, 40\%]$, to further refine the edited NeRF.

If the editing task requires editing the geometry or shape of the scene (“shape editing”), the depth-based warping using the depth of the original scene will be inaccurate. Therefore, in the Initialization Stage, we put the original diffusion model's output to the edited view buffer for NeRF fitting, equivalently using IN2N in this stage. In the distillation stage, the shape of the NeRF will be adjusted to the edited shape in a short time, and then we will start to use the trained diffusion model's output for NeRF fitting.

In our experiments, most IN2N scenes converge to a fine-grained edited scene at $600 \sim 700$ epochs, while ScanNet++ [15] scenes take around 1000 epochs.

D.4. Structured Noise Implementation

In the main paper, the structured noise is implemented by constructing "a dense point cloud of the scene by unprojecting all the pixels in all the views", and rendering/projecting such a point cloud at a view to generate the structured noise. Directly implementing this literal description is complicated and inefficient.

Therefore, we use an equivalent implementation.

- Instead of explicitly generating this dense point cloud, we just put the weighted noises on each pixel of all views.
- For the view we query for structured noise, we warp the noise from all other views to it. This is equivalent to projecting the sub-point cloud generated by each view to the querying view; therefore, it is equivalent to the original design.

With this implementation, explicitly generating, maintaining, and projecting a point cloud with billions of points (number of views $\times$ height $\times$ width) is unnecessary, and a query can be completed in less than one second.

D.5. Surrounding Views - Reference View Selection

We construct the surrounding view with one large main view and several small reference views. The purpose of the reference views is two-folded: (1) to provide enough context about the whole scene, and (2) to have enough overlapped parts of the main view to facilitate consistency-enforcing training. Therefore, we select 40% of the views to be a random view of the scene, and the rest 60% of the views to be a view with at least 20% overlap of the main view (quantified by the area of matched pixels through warping). The order of the views is randomly shuffled. We observed that none of these randomnesses highly alter the editing result – after consistency-enforcing training, any choice of reference views and their order will lead to a consistent edited result of the main view.

D.6. Training - Pixel Weights

In consistency-enforcing training, we apply warping and weighted averages to compute the training reference views $\{v'\}$, so that all the views in $\{v'\}$ are 3D consistent. Using identical weights for all pixels will result in blurred images: In a scene of a person, one view only contains their face, and another view contains the whole body. Warping the latter to the former indicates an upsampling of the face part, which will be blurred. Merging the blurred, warped view with the former view at the same weight results in blurred overall results.

We propose a better pixel-weighting strategy based on a further analysis of this situation. If we warp pixel a to pixel b, where a has a larger "scope" and contains more scene objects, then we need to upsample the b part of the view from a, resulting in blurry. Therefore, the weight should be related to the scope of the pixel. Following this, we define the pixel area to quantify this scope. For a pixel p in a view from camera position o, the four vertices of the pixel grids correspond to the rays $\{o+td_i\}_{i=1}^4$. We use NeRF to predict each of their depth $\{t_i\}_{i=1}^4$, and calculate their corresponding points $P_i = o + t_i d_i$. The pixel area $S(p)$ of this pixel is defined as the area of a square with vertices P_1, P_2, P_3, P_4 in the 3D space, which can be regarded as an approximation of the surface area the pixel represents. As we need a lower weight for a pixel with a larger scope, *i.e.,* larger $S(p)$, we define the weight as $1/S(p)$, which satisfies all our needs.

D.7. Training - Multi-GPU Pipeline

An illustration of our multi-GPU training pipeline is in Fig. D.1. By implementing such a parallelization pipeline by ourselves, we decouple NeRF training with diffusion generation and training in the most asynchronized way, waiving the necessity of trade-offs between NeRF training and diffusion, achieving considerable speed up.

D.8. Training - Regularizations

We use the consistency loss as the main loss in the consistency-enforcing training. However, this loss only enforces several equalities (required by consistency), leading to trivial results of a pure-color image without regularization losses – this is also reasonable as all pure-color images of the same color are perfectly consistent. Also, there is no encouragement or enforcement to use the 3D information in the 3D positional embedding. To avoid these, we propose several regularization losses, as shown below:

- Maintain Original Behavior. We expect that the trained diffusion model will generate images that are very similar to the original model when all the inputs (image, noises, and 3D positional embeddings) are identical. Therefore, we use MSE and VGG perceptual and stylization losses, to regularize both the generated images and the constructed referenced images (with gradient, generated by warping and averaging) of the trained diffusion, with the original model's output. We further expect that the UNet in the trained diffusion model predicts similar noises at each denoising step as the original UNet, so we also use this to regularize during each denoising step.
- Encourage 3D Information Utilization. The original [1]

takes the original image of the scene as another part of input, using it as a condition to generate the edited image. To encourage 3D information utilization, we design a regularization loss, to enforce the diffusion model *without* the original image input to generate very similar results to the one *with* the original image input (both with 3D positional embedding input). With the lack of the original image, the only way for the diffusion model to perceive the original view is the 3D positional embedding. Therefore, the diffusion is trained to use the 3D positional embedding at least for novel view synthesis to recover the original image, encouraging the utilization of 3D information. This regularization loss is also applied on the UNet in each denoising step.
- Encourage Consistent Editing Style. The diffusion model has some diversity in editing. However, we need to converge to one specific style in one editing procedure, otherwise, the NeRF may use view-dependency to overfit different styles at different views. Therefore, in the Pre-Annealing step (Sec. D.3), we use the NeRF's rendering result to supervise the diffusion model, to make it converge to the style NeRF converges to.

D.9. Variant "IN2N" And IN2N [4]

In our ablation study in the main paper, we have a variant "IN2N" being our full ConsistDreamer with all three major components removed. In this section, we discuss how it is equivalent to an implementation of IN2N, and the major differences between them.

IN2N is a method that (1) gradually generates newly edited images with a noise level (detailed in Sec. D.3) sampled from $[70\%, 98\%]$, and (2) uses the newly generated images to fit the NeRF, while the fitting NeRF's rendering results can affect the following editing (through the input of diffusion model as a mixture with noise). This matches our pre-annealing sub-stage. Therefore, "IN2N" includes vanilla IN2N as a sub-procedure. Additionally, "IN2N" has the following improvements beyond IN2N:

- IN2N only samples noise levels from $[70\%, 98\%]$. This makes IN2N (1) sometimes unable to sufficiently edit the scene due to the absence of 100% noise level editing (*e.g.*, unable to achieve a Lord Voldemort editing with no hair in Fig. B.2), and (2) cannot refine the editing results based on a converged style, and sometimes even deviates from a converged style to another, as the noise level is always as high as 70%. The variant "IN2N" starts at a full noise before the pre-annealing sub-stage, guaranteeing sufficient editing. After the pre-annealing sub-stage, "IN2N" anneals the noise level range to refine the results, leading to a more fine-grained editing.
- IN2N adds the newly edited image to the dataset by replacing a subset of pixels, which may negatively affect the LPIPS/perceptual loss. "IN2N" uses an edited view buffer to fit NeRF containing only full, edited views, on which the perceptual loss can perform well.

In conclusion, our variant, "IN2N," is an equivalent and improved implementation of IN2N. As shown in SV, "IN2N" generates noticeably better results than IN2N.

E. Supporting Evidence for Claims

E.1. Diffusion Models Perform Well with Composed Images

As shown in Fig. E.1, the pre-trained diffusion model [1], though not directly trained in this pattern, still works as expected in surrounding views. It generates editing results for each sub-view individually while all of them also share a similar style, across various scenes, including indoor, outdoor, and face-forwarding scenes.

Notably, as shown in the last row, when editing a view with little context, directly editing the single view fails. Constructing a surrounding view using it as the main view, however, helps the diffusion model [1] to achieve successful editing. This shows the effects of surrounding views in achieving successful and consistent editing.

E.2. Different Noises Lead to Varied Results

As shown in Fig. E.2, generation from different noises leads to completely different images, which is the fundamental constraint of all the baselines, which do not control the noise. Even with surrounding views, the diffusion model [1] still generates images in highly inconsistent ways. The diversity of the diffusion model under different noises is desirable in 2D generation and editing, but has to be controlled in 3D generation for consistency.

F. Additional Ablation Study Analysis

F.1. 'No Str. Noise' vs. 'Only Sur. Views'

Both variants do not have structured noise. Hence, the consistency-enforcing training in 'No Str. Noise' forces the model to generate the same result from different noises, which leads to mode collapse and degrades the editing result towards blurred, averaged color. These negative effects of training in 'No Str. Noise' leads to similar and even worse results and DFS than 'Only Sur. Views' with no training.

F.2. 'Only Sur. Views' vs. 'IN2N'

Tasks B,C,D are *style transfer*, specifically well supported by our current 2D diffusion model [1]. Our DFS metric, based on FID, uses a feature extractor with more tolerance for different style transfer results in the same image. Hence, even 'IN2N' performs comparably with a slightly lower DFS.

By contrast, task A is a *general object-centric editing* with diversified editing manners – different valid editing re-

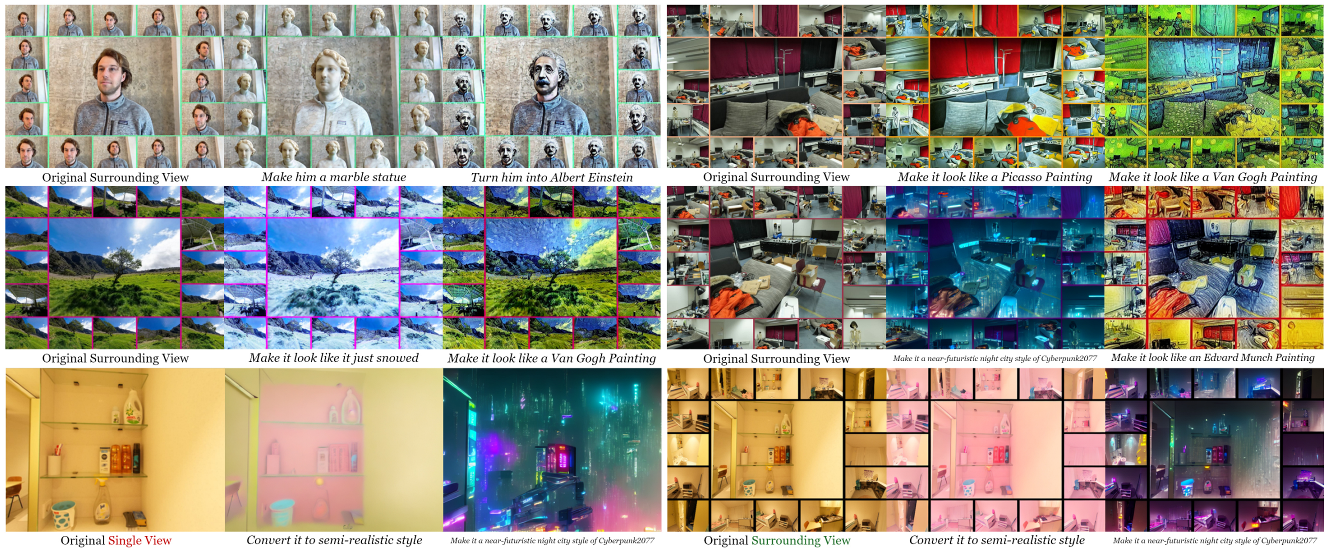
Original Surrounding View
Make him a marble statue
Turn him into Albert Einstein
Original Surrounding View
Make it look like a Picasso Painting
Make it look like a Van Gogh Painting
Original Surrounding View
Make it look like it just snowed
Make it look like a Van Gogh Painting
Original Surrounding View
Make it a near-futuristic night city style of Cyberpunk2077
Make it look like an Edvard Munch Painting
Original Single View
Convert it to semi-realistic style
Make it a near-futuristic night city style of Cyberpunk2077
Original Surrounding View
Convert it to semi-realistic style
Make it a near-futuristic night city style of Cyberpunk2077

Figure E.1. The pre-trained diffusion model [1] works as expected on surrounding views, by editing each sub-view in the instructed way individually but in a consistent style. Notably, as shown in the last row, the surrounding view enriches the context, making the diffusion model succeed in views that fail in single-view editing.

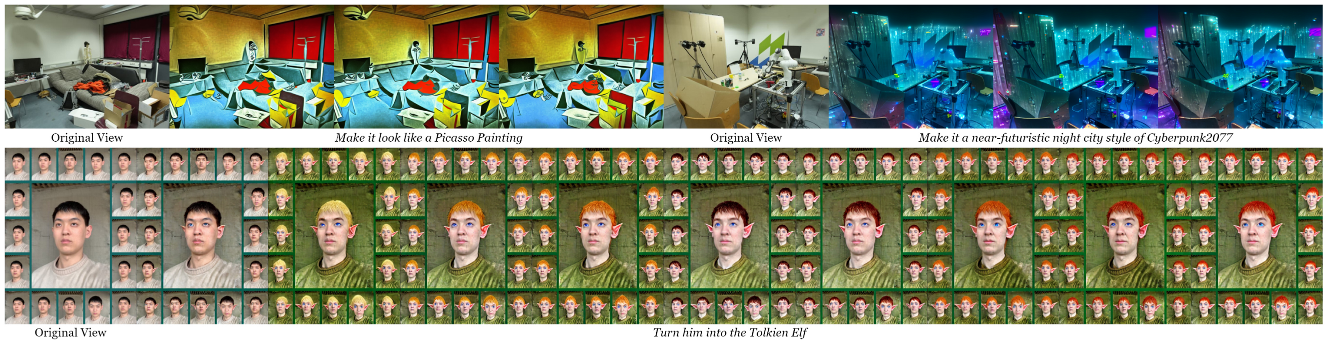
Original View
Make it look like a Picasso Painting
Original View
Make it a near-futuristic night city style of Cyberpunk2077
Original View
Turn him into the Tolkien Elf

Figure E.2. Even for the same view, generating from different noises does not necessarily lead to the consistent *i.e.*, the same, edited result. Each column represents a generation from a noise different from other columns.

sults can have jackets with completely different colors and styles. There can even be geometric changes in the clothing without surrounding views as context to constrain the editing, leading to a significantly worse DFS for 'IN2N.'

G. Discussion

G.1. Extension to Scene Generation

The proposed ConsistDreamer primarily focuses on the distillation-guided 3D scene editing task. However, the core contributions – structured noise, surrounding views, and consistency-enforcing training – can also be extended to the scene generation task. For example, these components can be used in the refinement phase, when the shape of a scene is roughly determined. In this way, these components could help achieve consistent and high-fidelity generation, refining the shape with slight adjustments for more detailed and precise geometry. Compared with previous methods [8, 14], this method can generate scenes with detailed, high-fidelity textures and shapes, without mesh exportation or fixing geometry.

G.2. Limitations

This section discusses the limitations of ConsistDreamer, which are also the *common challenges encountered by existing 3D scene editing methods*.

View-Dependent or Specular Effects. Our ConsistDreamer pipeline performs consistency-enforcing training by warping and averaging between different views. This procedure enforces that each part of the scene "looks the same" in different views, *i.e.,* is view-independent, making the edited scene unlikely to show view-dependent or specular effects. To preserve the ability to generate view-dependent effects, our ConsistDreamer has introduced a regularization loss that trades off between consistency and similarity to original [1] (detailed in Sec. D.8). With this regularization, our ConsistDreamer could still achieve 3D consistency while allowing natural view-dependent effects. The baselines, though are not trained towards consistency or view-independence, only generate blurred results without notable effects, or even overfit to inconsistent editing with the view-dependency of NeRF.

Editing Capabilities Constrained by 2D Diffusion Models. Our ConsistDreamer distills from the diffusion model [1] to edit scenes. Therefore, the editing ability, style, and diversity of ConsistDreamer are inherently constrained by [1]. Our ConsistDreamer edits a scene in a specific manner following [1]. For example, in the "Vincent Van Gogh" editing in Fig. B.2, our ConsistDreamer, along with IN2N [4] and ViCA [3] which use the same [1] for editing, shows a side effect that transfers the style of the image to Van Gogh's painting style. Moreover, we cannot support editing tasks on which the diffusion model cannot perform. Despite this common constraint among all the distillation-based methods, our ConsistDreamer successfully transfers most of the editing capabilities of the 2D diffusion model to 3D, by achieving high-quality and high-diversity 3D scene editing.

3D Understanding and Reasoning. Though our ConsistDreamer is 3D-informed and 3D-aware with the additional input of 3D positional embedding – already surpassing all the baselines – it is unable to reason and understand the semantics of each part of 3D scenes. Therefore, while our ConsistDreamer can edit a view using the knowledge of the whole scene's shape (via 3D positional embedding) and appearance (through the surrounding view), it may still encounter multi-face issues or Janus problems. Specifically, it does not understand what the correct orientation of the face is, does not know that a person can only have one face, and thus cannot avoid this problem.

Shape Editing. Some instructions for editing tasks may involve modifying the geometry or shape of a scene, *e.g*., "give him a beard" creates a beard on the face. Like the baselines, our ConsistDreamer is designed to support simple shape editing tasks that can be achieved by slightly and gradually alternating the surface. For example, in the editing of "give him a beard," our pipeline gradually "grows" the beard's shape from the face's surface. Notice that both ConsistDreamer and the baselines cannot perform aggressive and complicated editing (*e.g*., removing an object while reconstructing the whole occluded part), or direction-related editing (*e.g*., performing "lower down her arm" for a scene of a person raising her arm requires a multi-view consensus on which direction the arm is moved to).

Efficiency. In contrast to the diffusion training-free baselines such as [3–5], our ConsistDreamer needs additional training of a 2D diffusion model. This extends the editing duration, resulting in taking 12 hours to edit a scene in the IN2N dataset, and up to 24 hours to edit a large-scale indoor scene in the ScanNet++ dataset. However, as a trade-off against efficiency, our ConsistDreamer excels in achieving high-fidelity editing, surpassing all the training-free baselines.

G.3. Future Directions

Supporting Specular Effects. One direction is to support specular effects and better view-dependency. This may need an improved formulation of consistency under specular reflections, or modeling the ambient environment.

3D Understanding for Scene Editing. Another direction is to enable the diffusion model to understand and reason the semantics of a scene. Introducing a model that generates 3D semantic embeddings for each point in the scene allows for combining this information with the 3D positional embedding as the input to the diffusion model, potentially mitigating Janus problems.

References

[1] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Learning to follow image editing instructions. In *CVPR*, 2023. 2, 3, 5, 6, 7, 8

[2] Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. GaussianEditor: Swift and controllable 3d editing with gaussian splatting, 2023. 1, 3

[3] Jiahua Dong and Yu-Xiong Wang. ViCA-NeRF: View-consistency-aware 3D editing of neural radiance fields. In *NeurIPS*, 2023. 1, 8

[4] Ayaan Haque, Matthew Tancik, Alexei Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-NeRF2NeRF: Editing 3D scenes with instructions. In *ICCV*, 2023. 1, 3, 4, 6, 8

[5] Hiromichi Kamata, Yuiko Sakuma, Akio Hayakawa, Masato Ishii, and Takuya Narihira. Instruct 3D-to-3D: Text instruction guided 3D-to-3D conversion. *arXiv preprint arXiv:2303.15780*, 2023. 1, 3, 8

[6] Subin Kim, Kyungmin Lee, June Suk Choi, Jongheon Jeong, Kihyuk Sohn2, and Jinwoo Shin1. Collaborative score distillation for consistent visual editing. In *NeurIPS*, 2023. 1, 2

[7] Juil Koo, Chanho Park, and Minhyuk Sung. Posterior distillation sampling, 2023. 1, 3

[8] 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*, 2023. 7

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 1, 3, 4

[10] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 3

[11] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *CVPR*, 2023. 2, 3

[12] Liangchen Song, Liangliang Cao, Jiatao Gu, Yifan Jiang, Junsong Yuan, and Hao Tang. Efficient-NeRF2NeRF: Streamlining text-driven 3d editing with multiview correspondence-enhanced diffusion models, 2023. 1, 3

[13] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *SIGGRAPH*, 2023. 3

[14] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. ProlificDreamer: High-fidelity and diverse text-to-3D generation with variational score distillation. In *NeurIPS*, 2023. 7

[15] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. ScanNet++: A high-fidelity dataset of 3D indoor scenes. In *ICCV*, 2023. 3, 5

[16] Lu Yu, Wei Xiang, and Kang Han. Edit-DiffNeRF: Editing 3D neural radiance fields using 2D diffusion model. *arXiv preprint arXiv:2306.09551*, 2023. 1, 2

[17] Joseph Zhu and Peiye Zhuang. HiFA: High-fidelity text-to-3D with advanced diffusion guidance. *arXiv preprint arXiv:2305.18766*, 2023. 4

[18] Jingyu Zhuang, Chen Wang, Lingjie Liu, Liang Lin, and Guanbin Li. DreamEditor: Text-driven 3D scene editing with neural fields. In *SIGGRAPH Asia*, 2023. 1, 2