

1 Appendix

2 Contents

3	A Preliminaries on 3D Gaussian Splatting and Diffusion Model	2
4	A.1 3D Gaussian Splatting	2
5	A.2 Diffusion Models	2
6	B Implementation details	3
7	B.1 Camera Trajectory Design for View Interpolation	3
8	B.2 Algorithm for Pseudo-View Generation	3
9	B.3 Pseudo-View Selection for Gaussian Primitive Densification	4
10	B.4 Point Cloud Optimization from Multi-view Stereo Estimation	4
11	B.5 More Implementation Details	5
12	C Additional Experiments	6
13	C.1 Performance Variation as the Optimization Cycle Increases	6
14	C.2 Comparison with Methods Based on Image Inpainting	6
15	C.3 Runtime of Our Pipeline	6
16	C.4 Additional Qualitative Results	7

17 A Preliminaries on 3D Gaussian Splatting and Diffusion Model

18 A.1 3D Gaussian Splatting

19 3D Gaussian Splatting (3D-GS) [2] introduces an explicit representation of 3D scenes using a
 20 collection of 3D Gaussian primitives. In this framework, each Gaussian is defined by a mean position
 21 $\mu \in \mathbb{R}^3$ and a covariance matrix $\Sigma \in \mathbb{R}^{3 \times 3}$, which together determine its spatial distribution and
 22 shape in 3D space:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^\top \Sigma^{-1}(\mathbf{x} - \mu)\right). \quad (\text{S-1})$$

23 To model the anisotropic shape of each Gaussian, the covariance matrix Σ is factorized into a scaling
 24 matrix $S = \text{diag}(s_x, s_y, s_z)$ and a rotation matrix R , derived from a quaternion:

$$\Sigma = RSS^\top R^\top. \quad (\text{S-2})$$

25 Beyond its spatial properties, each Gaussian carries additional attributes that define its appearance
 26 and transparency. These include spherical harmonics (SH) coefficients \mathbf{c} for view-dependent color
 27 representation and an opacity parameter α . This combination of spatial and appearance parameters
 28 allows for the flexible modeling of complex 3D scenes.

29 To render a scene, 3D Gaussians are projected onto the 2D image plane through a differentiable
 30 splatting process. Given a viewing transformation W , each Gaussian is transformed into the camera
 31 coordinate system, and its covariance matrix is projected as:

$$\Sigma^{2D} = JW\Sigma W^\top J^\top, \quad (\text{S-3})$$

32 where J denotes the Jacobian of the projective transformation. The rendering process then computes
 33 the contribution of each Gaussian to each pixel, blending their respective colors and opacities through
 34 alpha compositing:

$$C_{\text{pix}} = \sum_i \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (\text{S-4})$$

35 Here, \mathbf{c}_i and α_i represent the color and opacity of each Gaussian, respectively. The final pixel color
 36 is obtained by blending overlapping Gaussian contributions, producing a smooth and high-quality
 37 rendering of the 3D scene. To get the depth rendering, we can substitute \mathbf{c}_i with the z-buffer of the
 38 corresponding Gaussian.

39 A.2 Diffusion Models

40 Stable Video Diffusion (SVD) [1] can be used as an image-to-video diffusion model that generates a
 41 natural video conditioned on an input image. By default, the video generation starts with the given
 42 image and autonomously evolves, incorporating random camera movements and scene dynamics.

43 Given a forward diffusion process expressed by $d\mathbf{x} = f(t)\mathbf{x}dt + g(t)d\mathbf{w}$, where \mathbf{x} is the noisy latent
 44 state at timestamp t , \mathbf{w} denotes the standard Wiener process, and $f(t)$ and $g(t)$ are scalar functions,
 45 its reverse process ODE [8] can be expressed as

$$d\mathbf{x} = \left[f(t)\mathbf{x} - \frac{1}{2}g^2(\mathbf{x})\nabla_{\mathbf{x}} \log(q_t(\mathbf{x})) \right] dt. \quad (\text{S-5})$$

46 In the case of the variance exploding (VE) diffusion [8] adopted by Stable Video Diffusion (SVD) [1],
 47 Eq. (S-5) can be simplified as:

$$d\mathbf{x} = \frac{\mathbf{x} - \hat{\mathbf{x}}_0}{\sigma_t} d\sigma_t, \quad (\text{S-6})$$

48 where the noise of the diffusion process is parameterized as Gaussian noise with a variance of σ_t and
 49 $\hat{\mathbf{x}}_0$ is the currently predicted clean video by the network based on the latent state at the previous step.

50 In practice, we can obtain the estimated denoised sample \mathbf{x}_{t-1} at the previous time step by discretizing
 51 the diffusion process above:

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \frac{\mathbf{x}_t - \hat{\mathbf{x}}_0}{\sigma_t} (\sigma_{t-1} - \sigma_t). \quad (\text{S-7})$$

52 B Implementation details

53 B.1 Camera Trajectory Design for View Interpolation

54 Given sparse-view inputs, we create a set of paired views according to their adjacency and interpolate
 55 the camera poses between each pair. For interpolation, we use spherical linear interpolation (slerp)
 56 for the rotation component and cubic spline interpolation for the translation component. To enhance
 57 coverage of under-observed areas, we repeatedly perturb each interpolated camera pose with random
 58 noise to generate a local candidate pool, and then select the candidates with the highest overall
 59 uncertainty (as estimated by Eq. (4) in the main paper) from each pool. The selected poses constitute
 60 the camera trajectory for view interpolation.

61 B.2 Algorithm for Pseudo-View Generation

62 With the paired input views and their interpolated trajectories, we employ the video diffusion model to
 63 generate pseudo views between each pair of input images. Since the original video diffusion model [1]
 64 is designed to accept only a single input image at a time, we propose a strategy, detailed in Sec. 4.1.4
 65 of the main paper, to effectively incorporate both views for interpolation.

66 Specifically, we independently encode the start and end views of each image pair using the VAE
 67 encoder, and run the denoising U-Net twice: once conditioned on the start view and once on the end
 68 view. This process generates two denoised latent sequences, which conceptually correspond to a
 69 forward video clip and its reversed version. We denote these two latent sequence at timestep $t - 1$ as
 70 $\mathbf{x}_{t-1}^{\text{forward}}$ and $\mathbf{x}_{t-1}^{\text{backward}}$, respectively. We then adaptively merge these two latent sequences to provide a
 71 better scene understanding for the video diffusion model: $\mathbf{x}_{t-1} = \beta \mathbf{x}_{t-1}^{\text{forward}} + (1 - \beta) R(\mathbf{x}_{t-1}^{\text{backward}})$,
 72 where $R(\cdot)$ is the reverse operation along the frame index dimension to align the latent $\mathbf{x}_{t-1}^{\text{backward}}$ to
 73 $\mathbf{x}_{t-1}^{\text{forward}}$ in the frame dimension. $\beta \in \mathbb{R}^N$ is the blending weight, with $\beta[i] = (N - i)/(N - 1)$
 74 for $i = 1, 2, \dots, N$, where N is number of interpolated frames between two inputs. The specific
 75 algorithm is provided in Alg. 1.

Algorithm 1 Pseudo-View Interpolation via Uncertainty-Aware Modulation

Require:

$\mathbf{x}_T \in \mathbb{R}^{N \times C \times H \times W}$: Initialized Gaussian noise;
 $\mathbf{c}_{\text{clip}}, \mathbf{c}'_{\text{clip}}$: CLIP condition for the start and end frame;
 $\mathbf{c}_{\text{vae}}, \mathbf{c}'_{\text{vae}}$: VAE condition for the start and end frame;
 $\mathbf{g} \in \mathbb{R}^{N \times C \times H \times W}$: The guidance image feature encoded by VAE;
 σ_t : Gaussian noise variance at timestep t .

for $t = T..1$ **do**

$\hat{\mathbf{x}}_0 = \mathcal{U}(\mathbf{x}_t; \mathbf{c}_{\text{clip}}, \mathbf{c}_{\text{vae}}, t)$
 $\hat{\mathbf{x}}'_0 = \mathcal{U}(R(\mathbf{x}_t); \mathbf{c}'_{\text{clip}}, \mathbf{c}'_{\text{vae}}, t)$ $\triangleright R(\cdot)$ reverses the input in the frame dimension.

$$\tilde{\mathbf{x}}_0 = \frac{1}{1+\gamma_t} \hat{\mathbf{x}}_0 + \frac{\gamma_t}{1+\gamma_t} \mathbf{g}. \triangleright \text{Vectorized solution of Eq. (2)}$$

$$\tilde{\mathbf{x}}'_0 = \frac{1}{1+\gamma_t} \hat{\mathbf{x}}'_0 + \frac{\gamma_t}{1+\gamma_t} R(\mathbf{g}). \triangleright \text{Vectorized solution of Eq. (2)}$$

$$\mathbf{x}_{t-1}^{\text{forward}} = \mathbf{x}_t + \frac{\mathbf{x}_t - \tilde{\mathbf{x}}_0}{\sigma_t} (\sigma_{t-1} - \sigma_t).$$

$$\mathbf{x}_{t-1}^{\text{backward}} = R(\mathbf{x}_t) + \frac{R(\mathbf{x}_t) - \tilde{\mathbf{x}}'_0}{\sigma_t} (\sigma_{t-1} - \sigma_t).$$

$$\mathbf{x}_{t-1} = \beta \mathbf{x}_{t-1}^{\text{forward}} + (1 - \beta) R(\mathbf{x}_{t-1}^{\text{backward}}).$$

end for

$\mathcal{I}^{\text{pse}} = \text{VAE_DEC}(\mathbf{x}_0)$ \triangleright Decode the denoised latent with the VAE decoder to be the output video frames.

return \mathcal{I}^{pse}

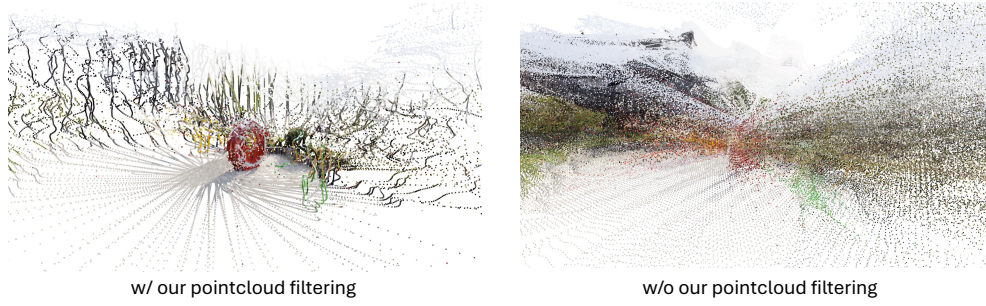


Figure S-1: Effectiveness of the point filtering for Gaussian primitive densification.

76 B.3 Pseudo-View Selection for Gaussian Primitive Densification

77 Sparse-input 3D-GS training often yields poor reconstruction in under-observed regions due to limited
 78 supervision. To enhance the novel view synthesis in under-observed regions, we propose to densify
 79 the Gaussian primitives leveraging the generated pseudo-views \mathcal{I}^{pse} and a dense stereo model [9].

80 To improve the efficiency and reduce the redundancy, we select a subset of pseudo views for scene
 81 reconstruction. To this end, we define a distance metric that quantifies the covisibility between two
 82 views:

$$\text{CovisibilityScore} = \exp(-\alpha \|\mathbf{t}_1 - \mathbf{t}_2\|) \exp(-\beta \arccos(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|})), \quad (\text{S-8})$$

83 where \mathbf{t}_1 and \mathbf{t}_2 denote the translation components of the two camera poses, and \mathbf{v}_1 and \mathbf{v}_2 are their
 84 viewing directions, respectively. α and β are parameters to balance the two terms, and are both set to
 85 1 in our case.

86 We further define a distance metric, $\text{CovisibilityDistance} = 1 - \text{CovisibilityScore}$, to measure the
 87 distance between two camera poses. Based on this metric, we perform furthest point sampling over
 88 all pseudo-views to select a total of S views. Specifically:

- 89 1. We define a set \mathcal{I}^{den} to contain the views used to densify the Gaussian primitive and initialize
 90 it as an empty set;
- 91 2. We begin by randomly sampling one view from the set \mathcal{I}^{pse} and adding it to \mathcal{I}^{den} ;
- 92 3. We define the distance between a candidate view and the set of views, \mathcal{I}^{den} , as the minimum
 93 $\text{CovisibilityDistance}$ between the candidate and any view already in \mathcal{I}^{den} . We compute this
 94 distance for each view in $\mathcal{I}^{\text{pse}} \setminus \mathcal{I}^{\text{den}}$, and select the view with the maximum distance to add
 95 to \mathcal{I}^{den} ;
- 96 4. We repeat step 3 until S views are finally selected, forming the final densification set \mathcal{I}^{den} .

97 We empirically set $S = 36$, which we found provides sufficient coverage while maintaining
 98 computational efficiency.

99 B.4 Point Cloud Optimization from Multi-view Stereo Estimation

100 We construct a complete graph over the selected views and jointly optimize their depth maps, guided
 101 by the stereo model outputs [9]. Concretely, we treat the depth maps and per-frame scale coefficients
 102 as optimizable parameters. Both the optimizable depth maps and the point clouds predicted by
 103 the stereo model are projected into the 3D-GS coordinate system using the corresponding camera
 104 parameters. For each pair of connected nodes in the graph, we minimize the distance between the
 105 projected points from the depth maps and the stereo predictions. The complete graph structure ensures
 106 global scale consistency across views. During this process, camera parameters—initialized in the
 107 3D-GS coordinate system—are held fixed. After optimization, the depth maps from all views are
 108 fused into a global point cloud.

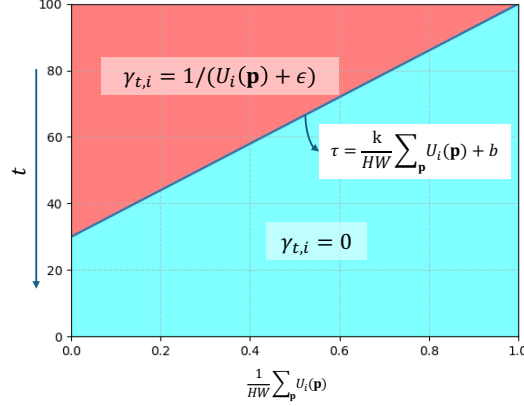


Figure S-2: Illustration of the relation between threshold τ and the uncertainty U_i . The boundary between the red and blue regions is defined by $\tau = \frac{k}{HW} \sum_{\mathbf{p}} (U_i(\mathbf{p})) + b$.

To alleviate the outliers from depth estimation and reduce the computational burden, we uniformly downsample the resultant point clouds and analyze the spatial distribution of the points obtained from the depth maps to filter out those that significantly deviate from the global average distance to neighboring points. We visualize the point clouds with and without this processing step in Fig. S-1.

Finally, we query existing Gaussian primitives within a fixed radius of each remaining point and only add new Gaussian primitives at positions without nearby primitives to augment the current set. The query radius is empirically set to the 85th percentile of the inter-point distances among the current Gaussian primitives.

B.5 More Implementation Details

LLFF. Following FSGS [11], we select every eighth image as the test set, and evenly sample sparse views from the remaining images for training. We utilize 3 views to train all the methods. We follow previous methods to initialize the 3DGS with the point clouds from SfM [7].

DL3DV. We use DL3DV’s test set for evaluation and hold every eighth image as our test split, and evenly sample sparse views (*i.e.*, 3, 6, 9 views) from the remaining images for training. We implement the 3D-GS baseline using the point clouds estimated with [9] from the sparse input views.

DTU. For the DTU dataset, we follow the protocol from RegNeRF [3], using 3 training views (IDs 25, 22, and 28) across 15 evaluation scenes. To focus on the object of interest, we mask out the background during evaluation using the provided object masks, consistent with [10, 3].

Our training framework is conducted cyclically. In each cycle, we train the 3D-GS model with 10K iterations and run the video diffusion model to update the pseudo-view images guided by the current 3D-GS. After each update of the pseudo-view images, we reset the learning rate schedule for 3D-GS and start the training of the next cycle. We empirically conduct 3 cycles for our experiments, although more optimization cycles can lead to better performance. In our implementation, we use DPT [6] to predict pseudo depth ground truth for regularization in \mathcal{L}_{reg} . For depth regularization on the pseudo views, in the first 30% training iterations, we produce pseudo depth based on the generated pseudo views. For the latter iterations, we generate pseudo-depth based on the color rendering from 3D-GS for better fidelity. For the experiments on LLFF and DTU datasets, the point clouds used for initialization are estimated by SfM [7] using the sparse input views. All experimental results are obtained on a single NVIDIA A40 GPU.

We illustrate the relation between the threshold τ (used in Eq. (5) in the main draft) and the overall uncertainty of an image. During the reverse sampling, the timestamp t decreases from the maximum timestamp, *e.g.* 100, to 0. In the early stage, $t \geq \tau$, the $\gamma_{t,i}$ in Eq. (6) (main paper) is set to $1/(U_i(\mathbf{p}) + \epsilon)$, while after $t < \tau$, $\gamma_{t,i}$ is set to 0. In the red region, the sampling process will rely more on the synthesized guidance images, while in the blue region, the prediction from the U-Net \mathcal{U}_θ

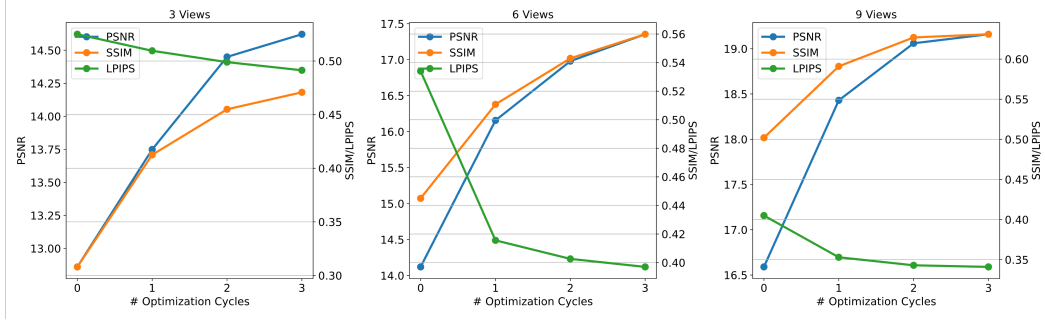


Figure S-3: Novel view synthesis performance across different optimization cycles, evaluated on DL3DV. We report PSNR, SSIM, and LPIPS metrics under varying numbers of input training views. SSIM and LPIPS share the right y-axis due to their similar value ranges.

is adopted. Generally, the larger the overall uncertainty is, the fewer reverse sampling steps will rely on the guidance image I_i^g . By changing k and b , the boundary can be adjusted. We found that our method is relatively robust to the selection of k and b . In our experiment, we set $k = 70$ and $b = 30$.

C Additional Experiments

C.1 Performance Variation as the Optimization Cycle Increases

We evaluated the novel view synthesis performance after different optimization cycles of 3D-GS. Fig. S-3 plots the performance variation on DL3DV. We observed that the performance steadily improves as more cycles of optimization are conducted, which demonstrates that the 3D-GS and the output from the diffusion model can enhance each other based on our proposed uncertainty-aware modulation mechanism. Though more optimization cycles may further improve overall performance, we set the number of optimization cycles to 3 in our experiments for training efficiency.

C.2 Comparison with Methods Based on Image Inpainting

To demonstrate the advantages of using a video diffusion model for view completion, we integrate an image inpainting model [4] into our pipeline as a replacement. Specifically, we use the uncertainty mask to indicate regions for inpainting. Evaluation results of these variants are shown in Table S-1. We implement two versions: (1) directly using the inpainting model’s outputs to supervise 3D-GS training (“Baseline 3D-GS+RePaint”), and (2) guiding the inpainting process with 3D-GS renderings using SDS [5]. While image inpainting improves upon the baseline, it remains inferior to our video diffusion-based method, as shown in Table S-1. This is because, in our framework, the video diffusion model leverages intact, paired input views to better infer intermediate content, whereas the inpainting model can only operate on a single, incomplete image for scene hallucination.

	PSNR↑	SSIM↑	LPIPS↓
Baseline 3D-GS+ RePaint [4]	17.45	0.516	0.403
Baseline 3D-GS+SDS[5]	18.12	0.551	0.738
Ours	19.19	0.616	0.335

Table S-1: We replace our video completion module with image-based inpainting techniques and compare the performance of these variants on DL3DV.

C.3 Runtime of Our Pipeline

The 3D-GS training in each optimization cycle takes approximately 30 minutes, while the video diffusion model requires 20 minutes to generate 25 frames on a single NVIDIA A40 GPU. The slow inference speed of the diffusion model is the primary bottleneck. Training efficiency could be

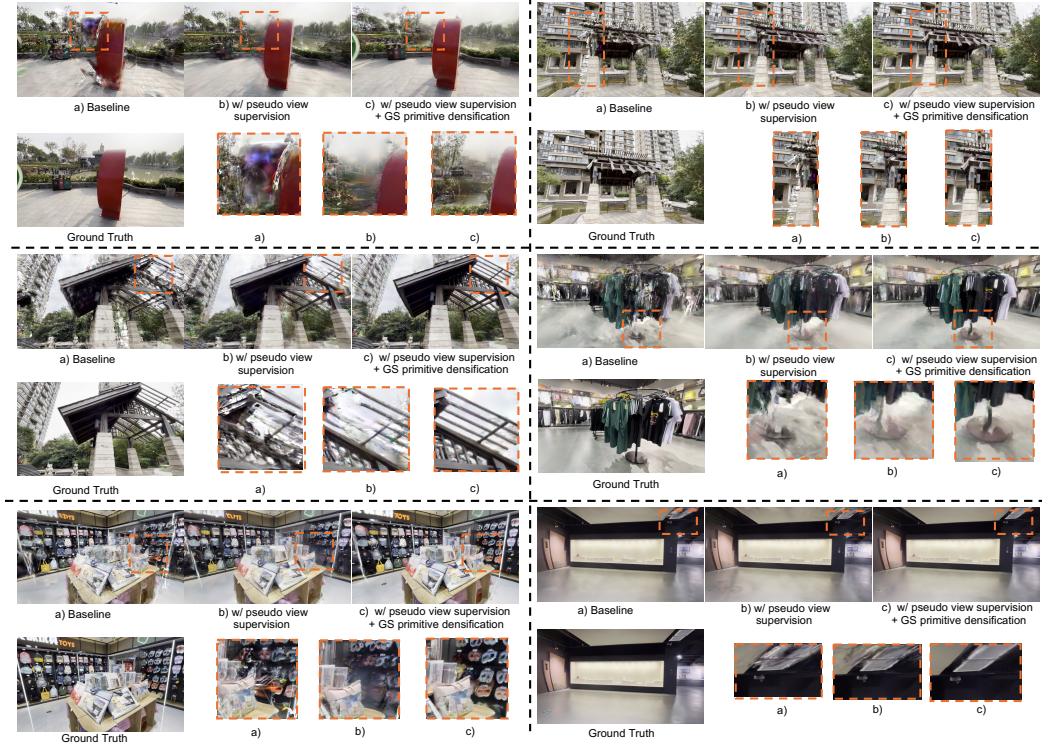


Figure S-4: Qualitative comparisons demonstrate the effectiveness of our generation-guided reconstruction pipeline. The proposed pseudo-view supervision and Gaussian primitive densification significantly enhance training in under-observed regions, leading to more photorealistic novel view synthesis results.

improved by distributing 3D-GS and diffusion inference across separate GPUs and running them in parallel—an optimization we leave for future work. Importantly, the final 3D-GS model maintains real-time rendering performance, achieving over 400 FPS on the LLFF dataset.

C.4 Additional Qualitative Results

Fig. S-4 provides more novel view synthesis results to demonstrate the effectiveness of our proposed modules. We adopt a strong baseline model by initializing the 3D-GS with point clouds constructed by [9]. However, our proposed modules—the pseudo-view supervision and Gaussian primitive densification—can further enhance the novel view synthesis quality, especially in the regions under-observed from the sparse input views.

Fig. S-5 and Fig. S-6 show additional qualitative comparisons on the LLFF and DL3DV datasets. By incorporating constraints from interpolated views generated by the video diffusion model, our method more effectively supervises the 3D-GS representation, especially in under-observed regions. These pseudo views offer direct visual guidance where sparse inputs fall short. In contrast, baseline methods rely solely on sparse inputs, resulting in inferior performance on these challenging datasets.

Fig. S-7 presents additional qualitative comparisons on the DTU dataset. Our method remains robust in this object-centric setting, producing more detailed textures compared to FreeNeRF and vanilla 3D-GS. Notably, these improvements are achieved while maintaining real-time rendering speed.

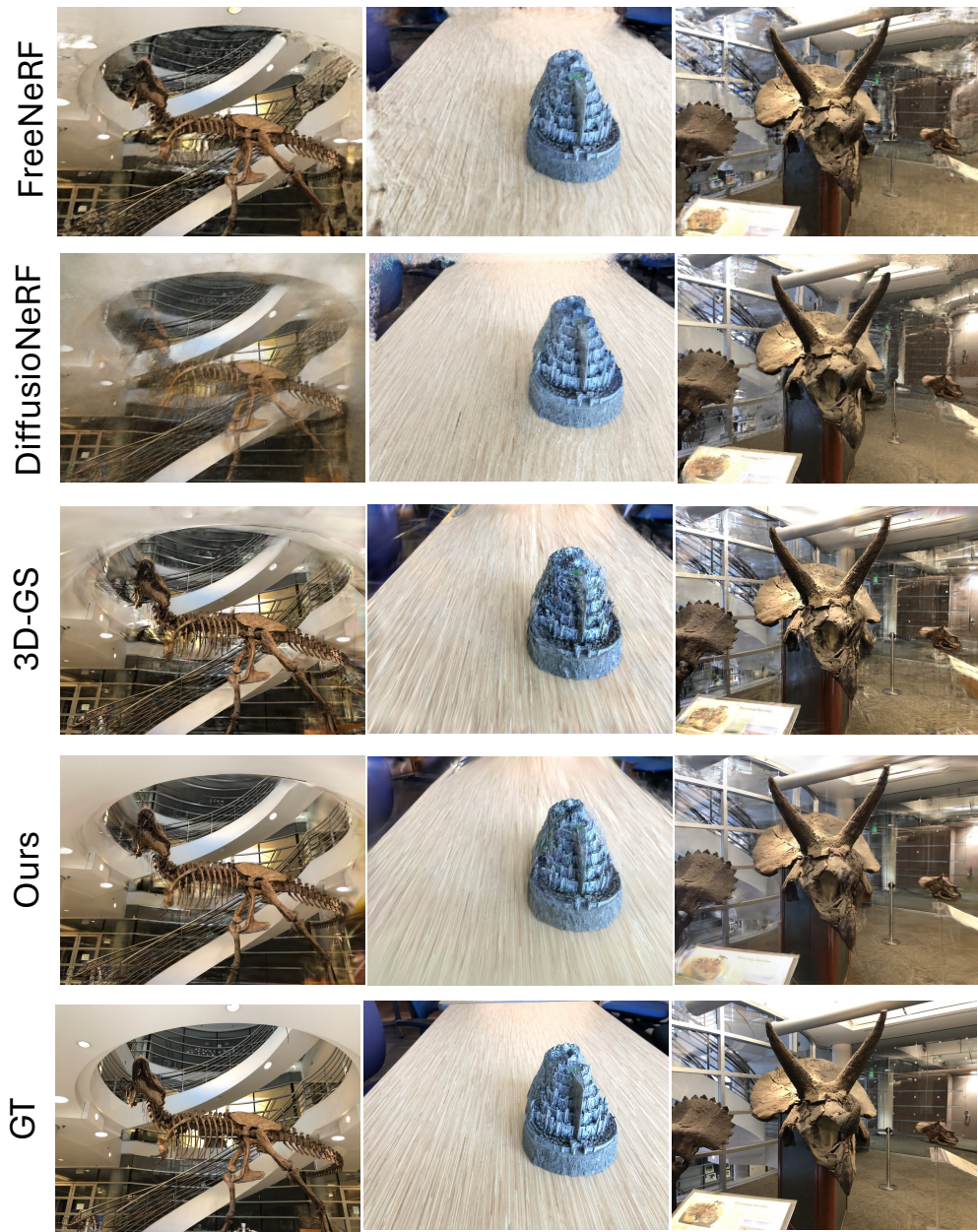


Figure S-5: Qualitative comparison on the LLFF dataset.



Figure S-6: Qualitative comparison with other cutting-edge counterparts on DL3DV dataset with 9 views as input.

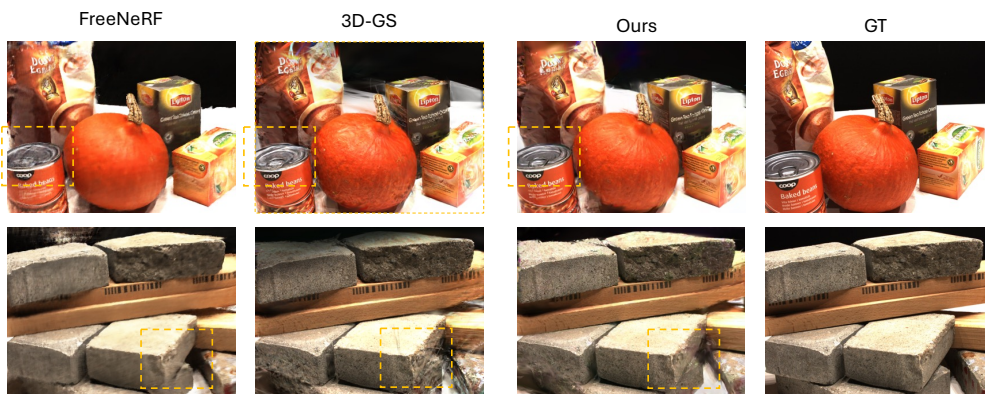


Figure S-7: Qualitative comparison with other methods on DTU dataset. Zoom in for better vision.

References

- [1] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.
- [2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.
- [3] Jiahe Li, Jiawei Zhang, Xiao Bai, Jin Zheng, Xin Ning, Jun Zhou, and Lin Gu. Dngaussian: Optimizing sparse-view 3d gaussian radiance fields with global-local depth normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20775–20785, 2024.
- [4] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and L Repaint Van Gool. Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11461–11471.
- [5] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *The Eleventh International Conference on Learning Representations*.
- [6] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. *ArXiv preprint*, 2021.
- [7] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [8] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.
- [9] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20697–20709, 2024.
- [10] Jiawei Yang, Marco Pavone, and Yue Wang. Freenerf: Improving few-shot neural rendering with free frequency regularization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8254–8263, 2023.
- [11] Zehao Zhu, Zhiwen Fan, Yifan Jiang, and Zhangyang Wang. Fsgs: Real-time few-shot view synthesis using gaussian splatting. In *European conference on computer vision*, pages 145–163. Springer, 2025.