Depth Priors for Monocular Dynamic 3D Reconstruction

Master Thesis

Jonas Klötzl

University of Tübingen
Faculty of Science
Department of Computer Science
Autonomous Vision Group

Tübingen, December 2025

Depth Priors for Monocular Dynamic 3D Reconstruction

Master Thesis

Jonas Klötzl
Student No. 5500688

Supervisor: Stefano Esposito (M.Sc.)
Autonomous Vision Group, University of Tübingen

First examiner: Prof. Dr.-Ing. Andreas Geiger
Autonomous Vision Group, University of Tübingen

Second examiner: Prof. Dr. Gerard Pons-Moll
Real Virtual Humans, University of Tübingen

University of Tübingen
Faculty of Science
Department of Computer Science
Autonomous Vision Group

Tübingen, December 2025

Klötzl, Jonas Manuel:
Depth Priors for Monocular Dynamic 3D Reconstruction
Master Thesis Computer Science
University of Tübingen
Thesis Period: 15.06.2025 – 15.12.2025

Kurzfassung

Die jüngsten Fortschritte in der Rekonstruktion dynamischer Szenen wurden hauptsächlich durch die Erweiterung bestehender 3D-Representationen wie NeRF und neuerdings 3D Gaussian Splatting auf den dynamischen Bereich vorangetrieben. Obwohl diese Ansätze beeindruckende Ergebnisse erzielen, basieren viele aktuelle Methoden des dynamischen Gaussian Splatting ausschließlich auf der Überwachung mittels RGB-Bildern, wodurch die inhärenten Herausforderungen des Problems weitgehend ungelöst bleiben. Die Integration von Tiefenprioren kann die Mehrdeutigkeit in der Tiefeninformation reduzieren und die Qualität der Rekonstruktion verbessern, insbesondere bei expliziten Darstellungen wie Gaussian Splatting. Die praktischen Vorteile der Verwendung von Tiefeninformationen in bestehenden monokularen dynamischen Rekonstruktionsverfahren blieben jedoch in der Vergangenheit weitestgehend unerforscht. Diese Arbeit untersucht unter Verwendung des standardisierten *Monoklularen Dynamischen Gaussian Splatting* Evaluationsframeworks, wie Tiefeninformationen moderne monokulare dynamische 3D-Rekonstruktionsmethoden verbessern können, und leistet dabei drei zentrale Beiträge: Zuerst wird durch eine umfassende Evaluation aktueller monokularer Tiefenvorhersagemodelle die geeignetste Methode identifiziert. Anschließend wird gezeigt, wie Tiefeninformationen in den Rekonstruktionsprozess integriert werden. Dies beinhaltet einen differenzierbaren Renderingmechanismus und eine speziell für Tiefenkarten entworfene Loss-Funktion, die ein adaptives Gewichtungsschema verwendet. Zuletzt wird ein auf Tiefenkarten basierender Initialisierungsprozess für den dynamischen Vordergrund vorgestellt, der eine präzise Platzierung von Gaussians zu Beginn des Trainings ermöglicht. Unsere Experimente zeigen, dass tiefenbasierte Überwachung und Initialisierung bei zwei der drei untersuchten Methoden zu einer besseren Rekonstruktionsqualität führt. Dabei liegen die PSNR-Steigerungen zwischen 0,3 und fast 1 dB. Allerdings ist die Effektivität stark vom zugrunde liegenden Bewegungsmodell, den Eigenschaften des Datensatzes und der Qualität der Tiefeninformationen abhängig. Dies bedeutet, dass die Integration von Tiefeninformationen eine sorgfältige und modellabhängige Bewertung unter Berücksichtigung sowohl der Qualität der Tiefenprioren als auch der modellspezifischen Bewegungsdarstellung erfordert. Code und Rekonstruktionsvideos sind verfügbar unter https://jonaskloetzl.github.io/MonoPriorsDyGauBench.

Abstract

Recent progress in dynamic scene reconstruction has largely been driven by extending existing 3D representations, such as NeRF and, more recently, 3D Gaussian Splatting, into the dynamic domain. While these approaches have demonstrated impressive results, many current dynamic Gaussian Splatting methods rely solely on RGB supervision, leaving the inherently ill-posed nature of the task largely unresolved. Incorporating depth priors might therefore reduce depth ambiguities and improve reconstruction quality, particularly for explicit representations such as Gaussian Splatting. However, the practical benefits within existing monocular dynamic reconstruction pipelines remain largely unexplored. Leveraging the standardized *Monocular Dynamic Gaussian Splatting* evaluation framework, this thesis examines how depth priors can enhance state-of-the-art monocular dynamic 3D reconstruction methods through three major contributions: First, an extensive evaluation of modern monocular depth prediction models exposes the most suitable for reconstruction. Then, depth supervision is integrated into the reconstruction pipeline through a differentiable inverse-depth rendering mechanism and a specifically designed loss function for depth maps using an adaptive weighting scheme. Lastly, a depth-guided initialization procedure is presented for the dynamic foreground area, enabling a more informed placement of Gaussian primitives at the start of training. Our experiments show that, for two out of the three investigated methods, depth-based supervision and initialization consistently enhance reconstruction quality, with improvements in PSNR of around 0.3 dB to nearly 1 dB. However, the effectiveness is highly dependent on the underlying motion model, dataset characteristics, and depth prediction quality, indicating that depth integration requires careful and model-specific assessment considering both the prior quality and the model's representation of motion. Code and reconstruction videos are available at https://jonaskloetzl.github.io/MonoPriorsDyGauBench.

Contents

Acronyms viii

Symbols x

1 Introduction **1**

2 Preliminaries **3**

2.1 NeRF . . . 3

2.1.1 Neural Radiance Field Scene Representation . . . 4

2.1.2 Volume Rendering with Radiance Fields . . . 4

2.1.3 Optimizing a Neural Radiance Field . . . 5

2.2 3D Gaussian Splatting . . . 6

2.2.1 Basics . . . 6

2.2.2 Differentiable 3D Gaussian Splatting . . . 6

2.2.3 Point-Based Rendering . . . 7

2.2.4 Optimization of 3D Gaussians . . . 7

2.2.5 Adaptive Density Control . . . 8

2.3 Depth Prediction Methods . . . 8

2.3.1 Monocular Depth Estimation . . . 9

2.3.2 Structure-from-Motion and Monocular SLAM Techniques . . . 10

2.3.3 Implication for Monocular 3D Reconstruction . . . 11

3 Related Work **13**

3.1 TiNeuVox . . . 14

3.2 STG . . . 16

3.3 RTGS . . . 16

3.4 DeformableGS . . . 17

3.4.1 Deformable 3D Gaussians . . . 18

3.4.2 Annealing Smooth Training . . . 19

3.5 EffGS . . . 19

3.5.1 Motion Representation . . . 20

3.5.2 Optimization . . . 20

3.6 4D-GS . . . 21

3.6.1 Spatial-Temporal Structure Encoder . . . 22

3.6.2 Multi-head Gaussian Deformation Decoder . . . 23
3.6.3 Optimization . . . 23
3.7 MonoDyBench . . . 24
3.7.1 Comparative Evaluation of Method Performance . . . 25
3.7.2 Impact of Scene Frequency on Convergence . . . 26
3.7.3 Influence of Multi-View Cues on Reconstruction Quality . . . 27

4 Method **29**
4.1 Evaluation of Depth Prediction Models . . . 29
4.1.1 Depth Map Alignment Strategies . . . 30
4.1.2 Quantitative Evaluation Metrics . . . 32
4.2 Integration of Depth Priors into 3D Reconstruction . . . 33
4.2.1 General Framework Modifications . . . 33
4.2.2 Depth Supervision Mechanism . . . 36
4.2.3 Depth-based Initialization Procedure . . . 40

5 Evaluation **45**
5.1 Evaluation of Depth Prediction Models . . . 45
5.1.1 Listing of all Depth Estimation Methods . . . 46
5.1.2 Key Observations . . . 47
5.1.3 Limitations of the Comparison Pipeline . . . 51
5.1.4 Implications for the Integration of Depth Priors . . . 52
5.2 Depth-Based Supervision . . . 53
5.2.1 Key Observations . . . 53
5.2.2 Failure Cases . . . 62
5.3 Depth-Based Initialization of Gaussians . . . 63
5.3.1 Evaluating Progressive Dataset Scheduling . . . 63
5.3.2 Key Observations . . . 64
5.3.3 Limitations . . . 71

6 Conclusion **75**

Bibliography 77

Appendices

A Additional Results: Evaluation of Depth Prediction Models **83**
A.1 Quantitative Result Plots Per Image . . . 84
A.1.1 Absolute Relative Error (AbsRel) . . . 84
A.2 Per Image Mean Squared Error (MSE) . . . 86
A.2.1 Foreground Evaluation . . . 86
A.2.2 Full-Image Evaluation . . . 87
A.3 Per Method Median Absolute Deviation . . . 88

A.4 Per Scene Mean Squared Error (MSE) Boxplots . . . 89
A.4.1 Per Image Median-Based Alignment . . . 89
A.4.2 Per Scene Median-Based Alignment . . . 90
A.4.3 Per Scene RANSAC-Based Alignment . . . 91

B Additional Results: Depth-Based Supervision **93**
B.1 Quantitative Results Per Dataset . . . 94
B.1.1 Nerfies . . . 94
B.1.2 iPhone . . . 95
B.2 Per Metric Quantitative Result Plots . . . 96
B.3 Per Dataset Quantitative Result Plots . . . 97
B.3.1 Foreground Evaluation . . . 97
B.4 Per Scene Quantitative Result Plots . . . 98
B.4.1 Nerfies . . . 98
B.4.2 iPhone . . . 102

C Additional Results: Depth-Based Gaussian Initialization **107**
C.1 Quantitative Results Per Dataset . . . 108
C.1.1 Nerfies . . . 108
C.1.2 iPhone . . . 109
C.2 Per Metric Quantitative Result Plots . . . 110
C.3 Per Dataset Quantitative Result Plots . . . 111
C.3.1 Foreground Evaluation . . . 111
C.4 Per Scene Quantitative Result Plots . . . 112
C.4.1 Nerfies . . . 112
C.4.2 iPhone . . . 116

D Erklärung Abschlussarbeit **121**

E Usage of AI **125**
E.1 Correction of Spelling & Grammar . . . 125
E.2 Code Generation . . . 125
E.2.1 Own viewmatrix transform . . . 125
E.2.2 Rectification of Out-of-Memory Error through Checkpointing . . . 126
E.2.3 Proposal of using Absolute Value of the Scale for Alignment in Loss Function . . . 126
E.2.4 Proposition of Memory Cleaning Function for Out-of-Memory Error . . . 126
E.2.5 Initialization of Gaussians via Depth Map . . . 127
E.2.6 Integration of Custom Sampler for Training Image Scheduling . . . 130
E.2.7 Creating Initial Script for Comparison of Depth Methods . . . 130
E.2.8 Creating Function for Alignment via RANSAC . . . 131

Acronyms

δ_1	Threshold Accuracy. (*p. 33, 45, 47–49*)
AbsRel	Absolute Relative Error. (*p. 32, 33, 45, 47, 48, 53, 84, 85*)
FPS	Frames Per Second. (*p. 25, 26, 54, 60, 65, 66, 94, 95, 108, 109*)
ICP	Iterative Closest Point. (*p. 41, 42*)
LiDAR	Light Detection and Ranging. (*p. 2, 30–32, 51, 52, 75*)
LPIPS	Learned Perceptual Image Patch Similarity. (*p. 25, 27, 34, 54, 56, 59, 60, 63, 65, 66, 68–73, 94, 95, 108, 109*)
MLP	Multilayer Perceptron. (*p. 14–19, 22, 23, 58*)
MS-SSIM	Multi-Scale Structural Similarity. (*p. 25, 54, 58–61, 64–66, 70, 71, 94, 95, 103, 108, 109*)
MSE	Mean Squared Error. (*p. 32, 33, 45, 47, 48, 53, 86, 87, 89–91*)
PSNR	Peak Signal-to-Noise Ratio. (*p. 25, 34, 54–56, 58–62, 64–69, 71, 75, 94, 95, 99, 102, 108, 109, 113, 114, 116*)
RANSAC	Random Sample Consensus. (*p. 31, 41, 45, 47–49, 51, 53, 84–88, 91*)
SfM	Structure-from-Motion. (*p. 1, 6, 8, 10, 11, 13, 14, 18, 19, 22, 34, 37, 38, 40, 41, 43*)
SSIM	Structural Similarity. (*p. 23, 25, 54, 56, 59–61, 63, 65, 66, 70, 71, 94, 95, 108, 109, 115*)
ViT	Vision Transformer. (*p. 3, 9, 10*)

Symbols

α	Value of a Gaussian at a pixel location. (*p. 5, 7, 8, 14, 17, 21*)
$\boldsymbol{c}$	Center of a Gaussian. (*p. 6, 14, 16–20, 22, 23*)
C	Color of a pixel or ray. (*p. 4, 5, 7, 14, 17*)
c	Color of an object in space (Gaussian, Voxel, ...). (*p. 4, 5, 7, 17, 21*)
d	Reference ground truth depth map. (*p. 30, 32, 33, 38*)
$\hat{d}$	Predicted depth map. (*p. 30–33, 38*)
$\mathcal{F}_\theta$	MLP network with weights θ. (*p. 4, 14–16, 19, 22, 23*)
G	3D Gaussian. (*p. 6, 14, 18, 20, 22, 23*)
$\mathcal{G}$	4D Gaussian. (*p. 16, 17*)
g	3D Gaussian primitive. (*p. 6, 17*)
ϱ	2D projection of a 3D Gaussian primitive onto the image plane. (*p. 17*)
k	Spherical Harmonics (SH) of a Gaussian. (*p. 6, 14, 16, 18, 20, 22, 23*)
$\mathcal{L}$	Loss term. (*p. 5, 7, 20, 21, 23, 24, 37, 38*)
$\boldsymbol{\mu}$	uv coordinate's 2D image plane projection of the 3D Gaussian center. (*p. 7*)
$\boldsymbol{p}$	Pixel of an image. (*p. 7, 14, 17, 31, 32*)
γ	Positional encoding. (*p. 14, 15, 17–19*)
$\boldsymbol{r}$	Rotation of a Gaussian. (*p. 6, 7, 14, 16, 18–20, 22, 23*)
r	Camera ray. (*p. 4, 5*)
$\boldsymbol{s}$	Scale of a Gaussian. (*p. 6, 7, 14, 16, 18, 20, 22, 23*)
s	Scale parameter applied in the depth alignment procedure. (*p. 30–32, 37, 38*)
t	Shift parameter applied in the depth alignment procedure. (*p. 30–32, 37, 38*)
σ	Opacity of a Gaussian. (*p. 4–7, 14, 16–18, 20, 22, 23*)

T Transmittance. (*p. 4, 5, 7, 17*)

θ Model weights. (*p. 18, 19*)

1 Introduction

Inspired by the human visual system with its two eyes enabling it to perceive the environment three-dimensionally, reconstructing the 3D world has been a vibrant research area in the field of computer vision. Now and then, the fundamental goal remains the same: to design an algorithm that, given a collection of unstructured images or a video sequence as an input, can produce a complete 3D reconstruction of the observed scene or object.

Historically, one of the first attempts to recover the 3D shape from a single image was the seminal work *Shape from Shading* by Horn et al., 1989. This was followed by the original Structure-from-Motion (SfM) approach by Tomasi et al., 1992, which demonstrated that the 3D structure of static scenes could be inferred from multiple views to synthesize novel views. The release of *Structure-from-Motion Revisited* Schönberger et al., 2016a and *COLMAP* Schönberger et al., 2016b has proven that such techniques can produce excellent results if the scene is captured by a sufficient number of images with large baselines.

With the advent of learning-based techniques, implicit neural representation methods, such as *NeRF* Mildenhall et al., 2020, have achieved unprecedented levels of fidelity and visual realism. More recently, explicit Gaussian Splatting-based representations, first introduced by Kerbl et al., 2023, have gained significant attention due to their reduced training time and real-time rendering capabilities.

While decades of research have produced very sophisticated solutions for static scene reconstructions, recovering dynamic scenes from a single moving camera remains one of the most challenging and important problems in modern computer vision. In this setting, not only must the scene geometry be estimated, but also the motion of the objects within it. The core challenges can be condensed into two fundamental ambiguities: *depth ambiguity*, where multiple 3D configurations can correspond to the same 2D projection, and *motion ambiguity*, where apparent motion in the image can arise from either the object's motion or camera movement.

In recent years, several works have attempted to extend the advantages of Gaus-

sian Splatting from static to dynamic reconstruction. Since many of these methods have been published around the same time, each claims superiority over state-of-the-art approaches, typically meaning *3D Gaussian Splatting* or dynamic variants of *NeRF*. To provide an objective comparison, *Monocular Dynamic Gaussian Splatting: Fast, Brittle, and Scene Complexity Rules* by Liang et al., 2025 introduced a standardized evaluation framework, comparing the most promising dynamic Gaussian Splatting methods. All of them have in common that, in contrast to the seminal *3D Gaussian Splatting* work, none of the evaluated models incorporate depth priors for regularization. Instead, they rely solely on RGB supervision, which might result in geometrically inconsistent reconstructions.

This raises two central questions: Is it possible to increase the reconstruction quality of state-of-the-art monocular dynamic 3D reconstruction methods by integrating depth priors? And does the performance of depth prediction models have an impact on reconstruction quality? This thesis addresses these questions through the following key contributions:

- **Evaluation of Depth Prediction Models:** Several state-of-the-art depth prediction models are evaluated using multiple quantitative metrics and three distinct alignment strategies. The comparison is conducted on the iPhone (Gao et al., 2022) dataset, which provides ground truth Light Detection and Ranging (LiDAR) depth for reliable validation.

- **Depth Supervision Mechanism:** A robust integration of depth information into the training pipeline is proposed to enable depth-based supervision during optimization. To achieve this, the differentiable rasterization pipeline is extended to render inverse depth in a differentiable manner. Furthermore, this inverse depth rendering is paired with an alignment step that adjusts the predicted depth to the ground truth depth prior to computing the loss. A decaying weighting factor is introduced to balance the influence of the depth loss across different scenes and datasets.

- **Depth-based Initialization Procedure:** To leverage depth maps for Gaussian initialization, alignment methods in both two and three dimensions are explored. This initialization process is complemented by the omission of the warm-up phase and additional refinements for color and scale initialization, ensuring consistent and robust quality of the placed Gaussian primitives.

2
Preliminaries

This chapter introduces the core theoretical foundations and representations that form the basis for the experimental work of this thesis, which investigates the impact of incorporating depth priors into monocular dynamic 3D reconstruction methods. Section 2.2 presents *3D Gaussian Splatting* (*3DGS*), a recent and highly efficient explicit point-based rendering technique that achieves state-of-the-art reconstruction quality while supporting real-time rendering. Following this, Section 2.1 revisits *Neural Radience Filds* (*NeRF*), one of the most influential approaches for implicit neural scene representation that has inspired many of the methods discussed later. Finally, Section 2.3 presents several depth prediction methods, including Vision Transformer (ViT)-based models and geometric reconstruction techniques. Both provide valuable depth information that can serve as priors, helping to reduce ambiguity when recovering 3D scenes from monocular images.

2.1 NeRF

The seminal work *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis* by Mildenhall et al., 2020 introduced a new way to represent 3D scenes. Instead of modeling geometry explicitly through voxels, meshes, or point clouds, NeRF represents the scene implicitly using a deep neural network. This *Neural Radiance Field Scene Representation* (Section 2.1.1) maps 3D coordinates and viewing directions to color and density values, allowing the model to predict the color of any pixel from an arbitrary viewpoint. The section *Volume Rendering with Radiance Fields* (Section 2.1.2) explains how these color and density values along a camera ray produce realistic images of the scene, enabling high-quality novel view synthesis. Figure 2.1 provides an overview of the complete NeRF pipeline, showing the flow from the input camera parameters and scene samples to the final rendered images.

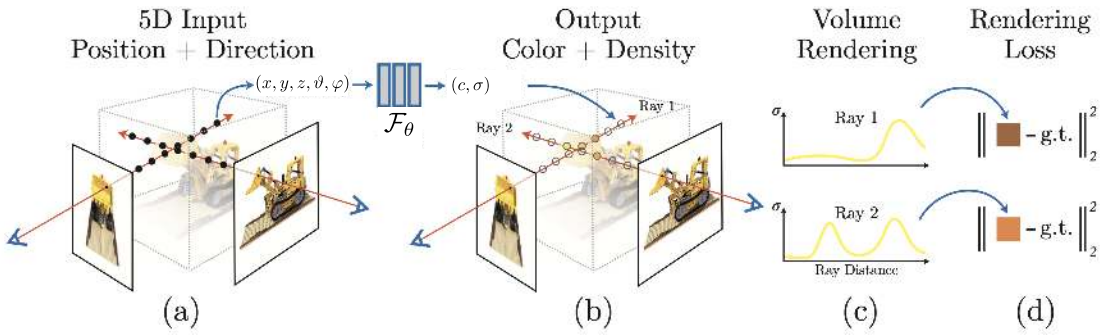


Figure 2.1: *Complete fully differentiable pipeline of the NeRF method. (a) The process begins by defining the 5D input, consisting of the 3D spatial location of a point and the corresponding camera viewing direction. (b) The neural network outputs the RGB color and volume density for the queried point. (c) Volume rendering determines the contributions of all sampled points along each camera ray to determine the final pixel color in the image plane. (d) The network is optimized by minimizing the squared L2 loss between the rendered and ground truth images. Figure adapted from Mildenhall et al., 2020.*

2.1.1 Neural Radiance Field Scene Representation

To reconstruct a scene Mildenhall et al., 2020 define a neural network $\mathcal{F}_\theta : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (c, \sigma)$ whose input is composed by a 3D-location $\boldsymbol{x} = (x, y, z)$ and the camera's viewing direction $\boldsymbol{d} = (\vartheta, \varphi)$. The network outputs the emitted RGB color c and the volume density σ, the latter being crucial for the volume rendering process described in Section 2.1.2.

The network is designed such that the density σ depends solely on the spatial position $\boldsymbol{x}$, whereas the color c additionally depends on the viewing direction $\boldsymbol{d}$. To meet these requirements, the first eight fully connected layers, each with 256 channels and ReLU activations, process only the 3D coordinate $\boldsymbol{x}$. Its 256-dimensional output σ is then concatenated with the two angles of the viewing direction $\boldsymbol{d}$. This combined representation is passed through a final fully connected layer, again with 256 channels and ReLU activation, to predict the RGB color c.

2.1.2 Volume Rendering with Radiance Fields

The volume rendering approach proposed by Mildenhall et al., 2020 enables the computation of the RGB pixel value on the image plane by querying the volume density and directional emitted radiance at all points along each camera ray. Given the near and far bounds t_n and t_f the color $C(r)$ of a camera ray $r(t) = \boldsymbol{o} + t\boldsymbol{d}$ is defined as

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), \boldsymbol{d})\, dt, \quad \text{where} \quad T(t) = \exp\left(-\int_{t_n}^{t_f} \sigma(r(s))\, ds\right). \tag{2.1}$$

Here, $T(t)$ denotes the transmittance which, analogous to the definition in Equation 2.9, represents the amount of light that reaches point $\boldsymbol{x}$ from the camera along the ray after passing through all the points in front of it.

In contrast to the point-based rendering approach in 3DGS (Section 2.2.3), NeRF

does not have a finite number of Gaussian primitives through which the ray passes. Instead, the model theoretically requires evaluating color and opacity at infinitely many points along the ray to determine its final color value. To make the integral in Equation 2.1 tractable, Mildenhall et al., 2020 use quadrature-based sampling techniques to approximate its value numerically. This is done by dividing the ray into N discrete bins from which points are drawn randomly. This stochastic sampling strategy is more effective than evaluating the same fixed set of N points along each ray, as it enables a quasi-continuous evaluation of points during training and improves the smoothness of the reconstruction. Based on this discretization, the color of a ray is redefined as:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))c_i \quad \text{where} \quad T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \tag{2.2}$$

with $\delta_i = t_{i+1} - t_i$ denoting the distance between two consecutive samples. This discrete formulation of the color closely resembles regular alpha blending as it is also used in 3D Gaussian Splatting by Kerbl et al., 2023 (see Equation 2.8), with $\alpha = (1 - \exp(-\sigma_i \delta_i))$.

2.1.3 Optimizing a Neural Radiance Field

In order to achieve state-of-the-art reconstruction quality Mildenhall et al., 2020 introduce two additional strategies:

First, due to high-frequency variations present in real-world data, the 5D input $(x, y, z, \vartheta, \varphi)$ is mapped to a higher-dimensional space using positional encoding.

Second, sampling points randomly along each ray, as described in Section 2.1.2, can be computationally inefficient in empty and occluded regions. To address this issue, the authors propose a hierarchical sampling scheme by defining two separate networks. The *coarse* network samples N_c points uniformly along each ray (see Equation 2.2), which provides a rough estimate of the scene's geometry and appearance. Based on these preliminary results, the *fine* network samples an additional N_f points around previously defined regions of interest, guided by the coarse network's output. As a result, a total of $N_c + N_f$ samples are evaluated per ray.

Finally, both the coarse and fine networks are jointly optimized using a squared error loss between the rendered and ground truth pixel colors:

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \left[\left\| \hat{C}_c(r) - C(r) \right\|_2^2 + \left\| \hat{C}_f(r) - C(r) \right\|_2^2 \right] \tag{2.3}$$

Here, $\mathcal{R}$ denotes the set of rays in a training batch with $C(r)$, $\hat{C}_c(r)$, $\hat{C}_f(r)$ represent the ground truth color, the coarse prediction, and the fine prediction for ray r, respectively.

2.2 3D Gaussian Splatting

The following pages define the key terms and principles from the seminal paper *3D Gaussian Splatting for Real-Time Radiance Field Rendering* by Kerbl et al., 2023, which form the basis for the Gaussian Splatting-based dynamic reconstruction methods presented in Chapter 3.

2.2.1 Basics

A 3D Gaussian is defined as five groups of parameters

$$G(\boldsymbol{c}, \boldsymbol{r}, \boldsymbol{s}, k, \sigma), \tag{2.4}$$

where the first parameter $\boldsymbol{c} = [c_x, c_y, c_z]^\top \in \mathbb{R}^3$ defines the position of the center of the Gaussian in 3D space, also denoted as the mean. In order to represent the rotation of the Gaussian, a unit quaternion $\boldsymbol{r} = [q_r, q_x, q_y, q_z]^\top \in \mathbb{R}^4$ is used, which compactly encodes a 3D rotation. The scaling parameter $\boldsymbol{s} = [s_x, s_y, s_z]^\top \in \mathbb{R}^3$ defines the standard deviations along the principal axes of the Gaussian ellipsoid before rotation. How the latter two parameters are used to compute the covariance matrix will be explained in Section 2.2.2. Spherical harmonics (SH) $k : h \in \mathbb{R}^{3\times(k+1)^2}$ are used, following the standard definition of Müller et al., 2022 and Fridovich-Keil et al., 2022 to model view-dependent color. This allows each Gaussian to emit color that varies smoothly with the viewing direction. Lastly, the scalar $\sigma \in \mathbb{R}$ controls the opacity of the Gaussian, which influences its contribution during rasterization.

Together, these parameters allow a sparse set of 3D Gaussians to approximate complex geometry and view-dependent appearance in a differentiable rendering pipeline.

2.2.2 Differentiable 3D Gaussian Splatting

For a differentiable rendering of the 3D Gaussians Kerbl et al., 2023 propose a Gaussian rasterization pipeline which starts by initializing the Gaussians with a sparse point cloud obtained by Structure-from-Motion (SfM) (Schönberger et al., 2016a). Following Zwicker et al., 2001, each Gaussian consists of a full 3D covariance matrix Σ defined in world space as well as its center defined by $\boldsymbol{c}$

$$g(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{c})}. \tag{2.5}$$

In order to render each individual pixel, Zwicker et al., 2001 use a 2D covariance matrix Σ' defined as

$$\Sigma' = JV\Sigma V^\top J^\top \tag{2.6}$$

to perform the projection of the 3D Gaussians to 2D. In this context, J refers to the Jacobian of the affine approximation of the projective transformation, V is the view matrix

responsible for mapping world coordinates into the camera coordinate frame, and Σ is the 3D covariance matrix. As covariance matrices need to be positive semi-definite, retaining this property is intractable when optimizing their parameters using gradient descent. Indicated in the definition of the Gaussians, Kerbl et al., 2023 solve this problem by representing Σ as a combination of the rotation matrix R and the scaling matrix S

$$\Sigma = RSS^\top R^\top . \tag{2.7}$$

Note that R and S are created by the two learnable components $\boldsymbol{r}$ and $\boldsymbol{s}$, where $\boldsymbol{r}$ is a quaternion defining the rotation and $\boldsymbol{s}$ the scaling.

2.2.3 Point-Based Rendering

Influenced by α-blending and NeRF-style volumetric rendering (Kopanas et al., 2021) Kerbl et al., 2023 define the color C of a pixel $\boldsymbol{p}$ as

$$C(\boldsymbol{p}) = \sum_{i \in N} T_i \alpha_i c_i , \tag{2.8}$$

where N represents the set of Gaussians with the corresponding color c_i intersecting the ray shoot from the pixel $\boldsymbol{p}$. Furthermore, the value α_i of the Gaussian i and the transmittance T_i is defined as

$$\alpha_i = \sigma_i e^{-\frac{1}{2}(p-\mu_i)^\top \Sigma'(p-\mu_i)} \quad \text{and} \quad T_i = \prod_{j=1}^{i-1}(1-\alpha_j) , \tag{2.9}$$

respectively. Here, α_i is the value of the i-th 2D Gaussian with covariance Σ (Yifan et al., 2019) evaluated at pixel $\boldsymbol{p}$, with $\boldsymbol{\mu}_i$ being the uv coordinate's 2D image plane projection of the 3D Gaussians, finally being multiplied with the learned per point opacity parameter σ_i. The transmittance T_i, however, can be described as the amount of light reaching Gaussian i from the camera along the ray after passing through all the Gaussians in front of it.

2.2.4 Optimization of 3D Gaussians

For optimization, Kerbl et al., 2023 repeatedly take a training image and use its extrinsics to render the reconstructed scene from that specific viewpoint, as illustrated in Figure 2.2. To quantify the quality of reconstruction, a weighted sum of $\mathcal{L}_1 = |\hat{I} - I|$ and structural dissimilarity loss $\mathcal{L}_{\text{D-SSIM}} = 1 - \text{SSIM}(\hat{I}, I)$ is used, with $\hat{I}$ and I being the reconstructed and the ground truth image, respectively. In the following, this combination of losses is called $\mathcal{L}_{\text{recon}}$ and is defined as

$$\mathcal{L} = \mathcal{L}_{\text{recon}} = (1-\lambda)\mathcal{L}_1 + \lambda \mathcal{L}_{\text{D-SSIM}} . \tag{2.10}$$

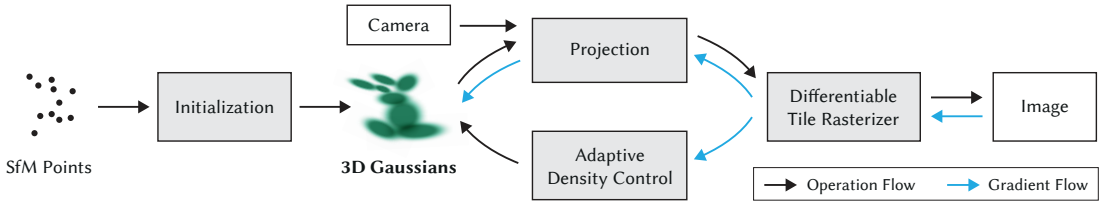


Figure 2.2: *Visualization of the 3D-GS Pipeline taking images, camera positions and initialized SfM points as input. After projecting the 3D Gaussians to 2D, differentiable Gaussian rasterization enables the joint optimization of their position and motion. Gradients also flow back to the Adaptive Density Control module, where they are used to determine which Gaussians need to be split or cloned. Operation and Gradient Flow are marked with black and blue arrows, respectively. Illustration taken from Kerbl et al., 2023.*

2.2.5 Adaptive Density Control

A key element of the 3DGS paper by Kerbl et al., 2023 is the introduction of what they call *Adaptive Density Control of 3D Gaussians*. This technique is crucial to handle void or crowded reconstruction areas, as well as transparent Gaussians. The latter is done by pruning Gaussians whose α value is under a certain threshold ϵ_α, which is chosen as a hyperparameter. Next, Kerbl et al., 2023 identify the following two reconstruction problems which they solve by evaluating the positional gradient: they call a scene *under-reconstructed* if there are not enough Gaussians to model the geometric features, which is quantified by checking if the positional gradient t_{pos} is larger than a certain threshold, typically chosen to be 0.0002. In this case, the existent Gaussian is cloned and shifted towards the direction of the positional gradient. If the positional gradient is smaller than the threshold, the scene is denoted *over-reconstructed*, which means that an existent Gaussian is too large to model the geometric features appropriately, causing the Gaussians to overlap. As a consequence, the corresponding Gaussian is split into two, with the scale of the division typically being defined via $\xi = 1.6$.

2.3 Depth Prediction Methods

Estimating scene depth from one or more images is a fundamental problem in computer vision, as it supports various downstream tasks such as 3D reconstruction, novel view synthesis, advanced image editing, and robotic manipulation. However, depth estimation remains a significant challenge due to its inherently ill-posed nature, as different 3D environments can result in the same 2D projection.

In general, depth prediction methods can be categorized into two main types: The first category includes *monocular depth estimation models*, as described in Section 2.3.1, where the model predicts either relative or metric depth from a single RGB image. The second category consists of *Structure-from-Motion and monocular SLAM techniques* (see Section 2.3.2), which exploit geometric consistency across multiple images to reconstruct a scene, with depth maps emerging as a byproduct of the reconstruction process. This section provides an overview of relevant publications in both categories and

highlights their relevance for using them as *Depth Priors in Monocular Dynamic 3D Reconstruction*.

2.3.1 Monocular Depth Estimation

Depth Anything V2 Similar to its predecessor, *Depth Anything V1* L. Yang et al., 2024b, the second version L. Yang et al., 2024a also builds upon the *DINOv2* (Oquab et al., 2023) Vision Transformer (ViT) backbone, which is fine-tuned for the task of monocular depth estimation. The training pipeline follows a three-stage teacher-student architecture. First, the authors use synthetic datasets consisting of 595,000 images with highly accurate ground truth depth to train the *teacher model* on top of *DINOv2* backbone. In the second stage, this teacher model is then used to generate pseudo ground truth depth for over 62 million real-world images. Finally, this large pseudo-labeled dataset is used to train the *student model*, resulting in improved robustness and generalization across diverse environments.

Video Depth Anything As the name suggests, *Video Depth Anything* by Chen et al., 2025 is a depth estimation model specifically designed to generate temporally consistent depth predictions for long video sequences. It builds upon *Depth Anything V2* (L. Yang et al., 2024a), which serves as a frozen backbone encoder while a dedicated spatial-temporal head (STH) is trained on top to capture temporal dependencies across consecutive frames. In order to ensure video-consistent depth, the authors introduce a *temporal gradient matching loss* that penalizes deviations in depth gradients between consecutive frames. Similar to *Depth Anything V2*, the model is trained first on approximately 730,000 video frames with ground truth depth and later is fine-tuned on around 620,000 pseudo-labeled images. To efficiently handle temporal consistency across long video sequences, the authors propose a *segment-wise processing strategy* in which sequences of 32 frames are jointly optimized, including eight overlapping frames and two key frames of previous video clips.

UniDepthV2 Similar to *Depth Anything V2* (L. Yang et al., 2024a), *UniDepthV2* by Piccinelli et al., 2025 builds upon its predecessor *UniDepth* (Piccinelli et al., 2024) with four key improvements. First, the authors update their camera-conditioned monocular metric depth estimator network by simplifying the connections between the Camera Module and Depth Module and adding convolutional layers to the depth decoder. Second, they increase the sharpness of the predicted depth through an edge-guided scale-shift-invariant loss. Third, Piccinelli et al., 2025, increase the diversity of training image shapes and resolutions within a mini-batch, in order to improve the model robustness during inference. Finally, the model introduces a new output that quantifies the uncertainty of each depth prediction, which is useful for downstream applications that require confidence-aware depth inputs.

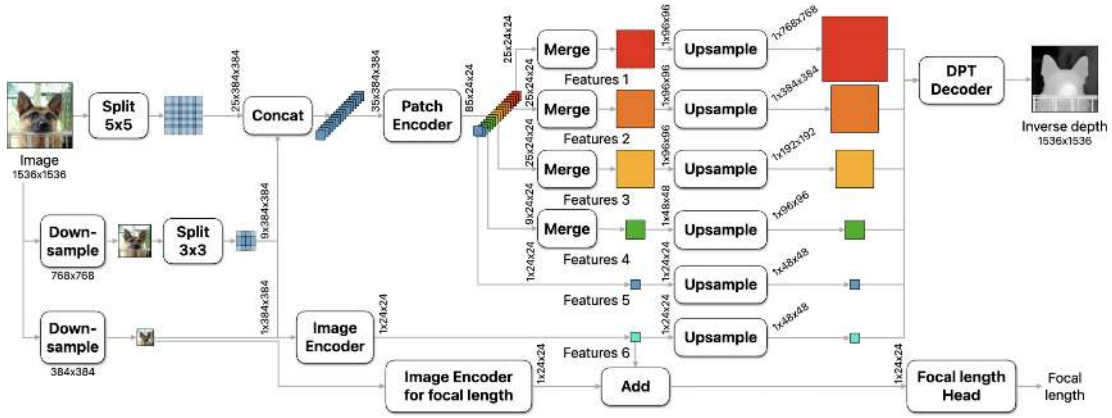


Figure 2.3: *Depth Pro pipeline, shown here as a representative example of modern monocular depth estimation architectures that uses a DinoV2 (Oquab et al., 2023) ViT backbone for the path and image encoders. Similar backbones are utilized in the other presented methods, Depth Anything V2 (L. Yang et al., 2024a), Video Depth Anything (Chen et al., 2025), UniDepthV2 (Piccinelli et al., 2025) and MoGe (R. Wang et al., 2025). Illustration taken from Bochkovskii et al., 2025.*

MoGe The authors of *MoGe*, R. Wang et al., 2025 proceed slightly different compared to the previously discussed methods. While they also exploit the *DinoV2* (Oquab et al., 2023) ViT backbone as the encoder, their model does not predict the depth map directly. Instead, it estimates an *affine-invariant point map*, which encodes relative 3D geometry independently of global scale and shift. In order to enable supervision with ground truth depth maps during training, R. Wang et al., 2025 present a custom *Robust, Optimal, and Efficient* (*ROE*) global alignment solver to accurately infer scale and shift parameters between the predicted and ground truth depths. Moreover, *MoGe* presents a novel *multi-scale local geometry loss*, which preserves geometric consistency by enforcing local similarities within reconstructed 3D point clouds.

Depth Pro From an architectural perspective, *Depth Pro* (Bochkovskii et al., 2025) also uses the *DinoV2* (Oquab et al., 2023) ViT backbone, similar to the *Depth Anything* series (L. Yang et al., 2024b, L. Yang et al., 2024a, Chen et al., 2025). Their multi-scale architecture (see Figure 2.3) is specifically designed to capture fine structures and preserve sharp edges at high resolution, which is additionally promoted by a new set of loss functions. To undermine the model performance in accuracy of boundary tracing, Bochkovskii et al., 2025 introduce novel metrics specifically designed to quantify the precision of depth discontinuities and edge alignment. In addition, *Depth Pro* features a unique zero-shot focal length estimation mechanism that infers camera intrinsics directly from a single image.

2.3.2 Structure-from-Motion and Monocular SLAM Techniques

Seminal works such as Structure-from-Motion (SfM) (Schönberger et al., 2016a) or *DROID-SLAM* (*Simultaneous Localization and Mapping*) (Teed et al., 2021) have established the foundations for recovering camera parameters and 3D scene geometry from

image sequences. However, dynamic scenes, uncontrolled in-the-wild settings with limited camera motion and rotating nearly stationary cameras can cause severe problems for traditional SfM and SLAM models. Despite these challenges, such methods solve dense or sparse estimation of depth en passant, as a byproduct of their geometric optimization process, effectively solving the depth estimation problem as an intermediate step. Importantly, the predicted depths must be geometrically consistent across all views, as small inconsistencies already can lead to reconstruction errors.

MegaSaM A recent work that tries to overcome the aforementioned problems of SLAM-based systems is *MegaSaM* by Zhengqi Li et al., 2024. As a foundation, the authors use the deep visual SLAM system *DROID-SLAM* (Teed et al., 2021) that is complemented with additional modules that jointly learn flow and uncertainty in order to predict object movement maps within dynamic scenes. The method further integrates depth priors into its training and inference pipeline, enabling more accurate reconstructions. Specifically, the authors align disparity predictions from *Depth Anything V1* (L. Yang et al., 2024b) to the scale of *UniDepthV2* (Piccinelli et al., 2025) in order to obtain metric depth estimates. Although *MegaSaM* is trained entirely on synthetic datasets, it still generalizes well to real-world data. As a final step, estimated camera parameters are used to refine the temporal and geometric consistency of depth predictions throughout the video sequence.

2.3.3 Implication for Monocular 3D Reconstruction

Reconstructing dynamic 3D scenes from monocular video is an inherently ill-posed problem, as both scene geometry and motion have to be derived from 2D images alone. In particular, good geometry estimation is necessary for achieving reliable novel-view synthesis. Therefore, depth priors can play an important role in reducing this ambiguity by introducing spatial constraints that stabilize optimization and improve the model's generalization capabilities. Methods such as *Shape of Motion* (Q. Wang et al., 2025) have recently shown that combining predicted depth maps together with long range 2D track estimation enables state-of-the-art 4D reconstructions. In a similar spirit, the depth priors used in this work act as a geometric guidance to improve the quality of dynamic scene reconstruction at best. However, quality and temporal consistency of these methods can vary considerably depending on the selected architecture and training data.

3
Related Work

Over the past few years, a variety of approaches have been published to achieve efficient and accurate reconstruction of dynamic 3D scenes. Earlier methods, such as *TiNeuVox* (Fang et al., 2022, see Section 3.1), have demonstrated that Neural Radiance Fields (NeRF) based techniques can produce high-quality reconstructions while significantly reducing the training time, which, traditionally, is one of the main drawbacks of such methods. More recently, with the release of *3D Gaussian Splatting* (*3DGS*) (Kerbl et al., 2023), Gaussian Splatting-based techniques have become dominant, aiming to exploit its real-time rendering capabilities and high photorealism already established in the static case.

As a consequence, several variants have been proposed for dynamic scenes, including *Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis* (*STG*) (Zhan Li et al., 2023, see Section 3.2), *Real-Time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting* (*RTGS*) (Zeyu Yang et al., 2024, see Section 3.3), *Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction* (*DeformableGS*) (Ziyi Yang et al., 2024, see Section 3.4), *A Compact Dynamic 3D Gaussian Representation for Real-Time Dynamic View Synthesis* (*EffGS*) (Katsumata et al., 2024, see Section 3.5) and *4D Gaussian Splatting for Real-Time Dynamic Scene Rendering* (*4D-GS*) (Wu et al., 2024, see Section 3.6). Each of them proposes different strategies for improving deformation handling, efficiency, and temporal coherence. The paper *Monocular Dynamic Gaussian Splatting: Fast, Brittle, and Scene Complexity Rules* (*MonoDyBench*) (Liang et al., 2025, see Section 3.7) provides an extensive comparison of these methods within a unified evaluation framework.

In this chapter, particular focus is laid on *EffGS*, *DeformableGS*, and *4D-GS*, as they form our study foundations for exploring depth priors in monocular dynamic 3D reconstructions using *MonoDyBench*'s evaluation setup (see Chapter 4 and Chapter 5).

All three of the methods share a common input setup: besides the image sequence with corresponding timestamps and camera intrinsics, they also rely on camera extrinsics obtained by SfM (Schönberger et al., 2016a). For initialization of the Gaussians, de-

pending on the nature of the dataset, a randomly generated uniformly distributed set of Gaussians is used for synthetic scenes and a sparse SfM point cloud for real scenes, respectively. As introduced in *3DGS*, a Gaussian $G(\boldsymbol{c}, \boldsymbol{r}, \boldsymbol{s}, k, \sigma)$ is defined via a combination of attributes consisting of its center location $\boldsymbol{c}$, the quaternion $\boldsymbol{r}$ and the scaling $\boldsymbol{s}$, both used for creating the 3D covariance matrix Σ as well as its opacity σ. Additionally, for each Gaussian spherical harmonics k are used to model the view-dependent appearance. Although the motion representations differ across methods, they follow a similar training principle: in order to avoid difficulties in convergence during training, all authors propose a warm-up phase during which 3D Gaussians are optimized as if it were a static scene. The motion model is activated only after a predefined number of 3000 iterations, allowing the deformation fields later on to focus more on the dynamic part of the scene.

For differentiable rendering of 3D Gaussians, all three authors, together with *STG*, use the same Gaussian rasterization pipeline by Kerbl et al., 2023 as described in Section 2.2.2, capable of supporting α-blending of anisotropic splats. Similarly, the definition of the pixel color $C(\boldsymbol{p})$ follows the point-based rendering setup as it is described in Section 2.2.3.

Furthermore, it is important to mention that adaptive density control discussed in Section 2.2.5 is used in all five Gaussian Splatting-based techniques to handle void or crowded reconstruction areas as well as transparent Gaussians.

3.1 Fast Dynamic Radiance Fields with Time-Aware Neural Voxels (TiNeuVox)

One of the most prominent approaches to dynamic 3D reconstruction based on Neural Radiance Fields (NeRF) is *Fast Dynamic Radiance Fields with Time-Aware Neural Voxels* (*TiNeuVox*) published by Fang et al., 2022. In contrast to other NeRF-based methods such as *D-NeRF* (Pumarola et al., 2021), *TiNeuVox* does not rely only on an implicit scene representation with a neural network. Instead, it introduces so-called *Neural Voxels*, an explicit voxel-based representation, that drastically reduces training time from several days to just around eight minutes. An overview of the architecture is shown in Figure 3.1.

To accelerate the convergence of radiance fields, Fang et al., 2022 propose a new explicit representation stage in between the *Coordinate Deformation* and the *Radiance Network*. The process begins with a simple *Coordinate Deformation* block, which transforms the positional-encoded location $\gamma(x, y, z)$ and the time embedding $\boldsymbol{t}_i = \mathcal{F}_{\theta_t}(\gamma(t_i))$ using a deformation network $\mathcal{F}_{\theta_d}$ to obtain shifted coordinates (x', y', z') in canonical space. Note that the time embedding $\gamma(t_i)$ is first processed through a separate lightweight two-layer Multilayer Perceptron (MLP) $\mathcal{F}_{\theta_t}$ before being passed to the deformation network.

Next, the *Multi-Distance Interpolation* block samples features from a voxel grid $\boldsymbol{V} \in$

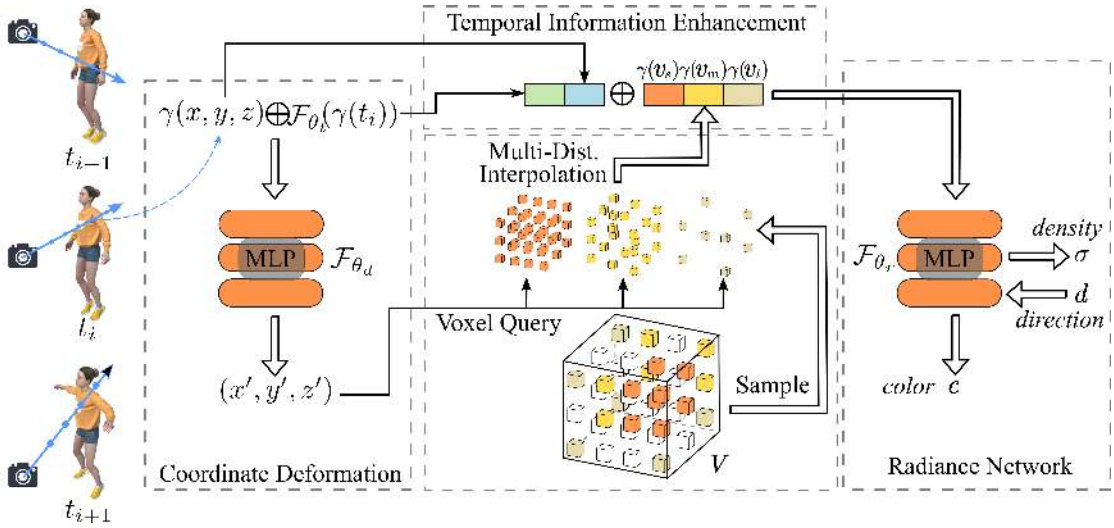


Figure 3.1: *The TiNeuVox architecture introduces two new components enabling intermediate explicit representation, the Temporal Information Enhancement and Multi-Distance Interpolation blocks. These complement the Coordinate Deformation and Radiance Network modules, which are conceptually similar to the deformation and canonical networks used in D-NeRF (Pumarola et al., 2021). Figure adapted from Fang et al., 2022.*

$\mathbb{R}^{C_v \times N_x \times N_y \times N_z}$ at multiple resolutions, where (N_x, N_y, N_z) define the spatial extent and C_v denotes the number of feature channels per voxel. This multi-distance sampling is essential for capturing arbitrary motion: coarse voxel grids help represent large motion, while having bigger resolution voxels helps perceive and model small movement more precisely. The voxel features for a given 3D point and grid resolution are obtained via trilinear interpolation by querying the eight neighboring voxels surrounding a given point in space.

In the *Temporal Information Enhancement* block, the resulting interpolated and positional encoded sampling stride features $\gamma(v_1), \gamma(v_2), \cdots, \gamma(v_M)$ are concatenated with the enhanced positional encoding $\gamma(x, y, z)$ and the time embedding $\boldsymbol{t}_i$. Finally, the *Radiance Network* block, a small MLP defined as $\mathcal{F}_{\theta_r}$, takes this combined representation as input to predict the color and density of the queried point (x, y, z) at time t.

However, as noted by Fang et al., 2022, this method also comes with some limitations: in addition to difficulties with long-distance motion, *TiNeuVox* especially struggles with specular surfaces in scenes, which is a typical vulnerability of existing neural representations. Moreover, as shown in Liang et al., 2025, its training time is significantly shorter, with roughly half that of Gaussian Splatting-based dynamic reconstruction methods. Nevertheless, this advantage is offset by extremely slow rendering frame rates, which are reported to be slower by a factor of about 500. Also, it is worth noting that the approach is purely vision-based, relying exclusively on RGB supervision without incorporating additional priors.

3.2 Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis (STG)

The motion model presented in the paper *Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis* (*STG*) by Zhan Li et al., 2023 is similar to the one used in EffGS (Katsumata et al., 2024, see Section 3.5) as they also represent the rotation over time by approximating each quaternion with a linear function (polynomial function with degree 1). In addition, the motion of each Gaussian center is not modeled by a MLP but rather by a polynomial function of degree 3 ensuring a good trade-off between motion representation flexibility and memory efficiency. Notably, STG is the only method that introduces a temporal component to the opacity, achieved through a 1D Gaussian parametrized by three parameters. Specifically, the center of the opacity modeling 1D Gaussian, its opacity, as well as its temporal stretching or compression, thus defining how long the 3D Gaussian remains visible. The biggest difference to all of the other presented methods is that the color of the Gaussian is neither defined by spherical harmonics nor by a standard RGB color value. Instead, Zhan Li et al., 2023 introduce a set of features

$$f_i(t) = \left[f_i^{\text{base}}, f_i^{\text{dir}}, (t - c_i)_i f_i^{\text{time}}\right]^\top, \qquad f_i^{\text{base}}, f_i^{\text{dir}}, f_i^{\text{time}} \in \mathbb{R}^3 \tag{3.1}$$

including the base color f_i^{base} as well as encodings of the view direction f_i^{dir} and time f_i^{time}. Following this, the authors splat these features by dual using the rasterizer's color rendering to obtain the splatted features $\boldsymbol{F}^{\text{base}}$, $\boldsymbol{F}^{\text{dir}}$, and $\boldsymbol{F}^{\text{time}}$. At the core of their color representation lies a two-layer MLP $\mathcal{F}_\theta$, which, in a final step, maps the splatted features to an RGB value for each pixel as follows:

$$\boldsymbol{I} = \boldsymbol{F}^{\text{base}} + \Phi(\boldsymbol{F}^{\text{dir}}, \boldsymbol{F}^{\text{time}}, \boldsymbol{d}), \tag{3.2}$$

with $\boldsymbol{d}$ being the viewing direction. The motivation behind this design is to reduce the number of parameters from 48, as required for a 3-degree SH, down to only 9 whilst preserving high rendering speeds.

3.3 Real-Time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting (RTGS)

The only method to extend the formulation of Kerbl et al., 2023 towards true 4D Gaussian Splatting was introduced by Zeyu Yang et al., 2024, titled *Real-Time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting* (*RTGS*). In contrast to static 3D Gaussians, each element in RTGS is represented as a 4D Gaussian $\mathcal{G}(\boldsymbol{c}, \boldsymbol{r}, \boldsymbol{s}, k, \sigma)$ with the covariance matrix being parametrized as $\Sigma = RSS^\top T^\top$. Here, the scaling matrix is also defined by its diagonal elements as $S = \text{diag}(s_x, s_y, s_z, s_t)$, whereas the rotation matrix is obtained by decomposing it into two isotropic rota-

tions $L(q_l), R(q_r) \in \mathbb{R}^4$. In combination with the the utilization of the quaternions $q_l = (a, b, c, d)$ and $q_r = (p, q, r, s)$, this enables its construction with $R = L(q_l)\,R(q_r)$. Finally, the Gaussian center is defined by $\boldsymbol{c} = (c_x, c_y, c_z, c_t)$.

To render a dynamic scene at a given time t using Equation 3.5, each 4D Gaussian is decomposed into a conditional 3D Gaussian $\mathcal{G}(c_x, c_y, c_z \mid c_t)$ and a marginal temporal Gaussian $\mathcal{G}(c_t)$ according to the factorization $\mathcal{G}(c_x, c_y, c_z, c_t) = \mathcal{G}(c_x, c_y, c_z \mid c_t)\mathcal{G}(c_t)$. The conditional spatial parameters are derived analytically as

$$\begin{aligned} \boldsymbol{c}_{x,y,z|t} &= \boldsymbol{c}_{1:3} + \Sigma_{1:3,4}\Sigma_{4,4}^{-1}(t - c_t), \\ \Sigma_{x,y,z|t} &= \Sigma_{1:3,1:3} - \Sigma_{1:3,4}\Sigma_{4,4}^{-1}\Sigma_{4,1:3}. \end{aligned} \tag{3.3}$$

Note that the marginal $\mathcal{G}(c_t) = \mathcal{N}(t; c_4, \Sigma_{4,4})$ is a Gaussian as well, acting as a temporal weighting factor to indicate the activity of each Gaussian over time.

At the heart of their 4D representation is the reformulation of the color C (Equation 2.8) of pixel $\boldsymbol{p} = (u, v)$ at time t via

$$C(\boldsymbol{p}, t) = \sum_{i \in N} c_i \alpha_i(\boldsymbol{p}, t) T_i(\boldsymbol{p}, t). \tag{3.4}$$

By separating the opacity σ_i from the value α_i of Gaussian i with $\alpha_i(\boldsymbol{p}, t) = \sigma_i\, \varrho_i(\boldsymbol{p}, t)$ the Gaussian can be factorized into the conditional and marginal probability with $\varrho_i(\boldsymbol{p}, t) = \varrho_i(\boldsymbol{p} \mid t) \cdot \varrho_i(t)$. Given the conditional 3D Gaussian $g(x, y, z \mid t)$, its projection to the image plane Gaussian $\varrho(\boldsymbol{p} \mid t)$ follows the same differentiable rasterization principle introduced by Kerbl et al., 2023. Integrating this knowledge into Equation 3.4 yields

$$C(\boldsymbol{p}, t) = \sum_{i=1}^{N} \varrho_i(t)\varrho_i(\boldsymbol{p} \mid t)\sigma_i c_i \prod_{j=1}^{i-1}(1 - \varrho_j(t)\varrho_j(\boldsymbol{p} \mid t)\sigma_j). \tag{3.5}$$

3.4 Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction (DeformableGS)

The following section is based on the Paper *Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction* by Ziyi Yang et al., 2024, which will gonna be abbreviated as DeformableGS in the following. It introduces a deformable 3D Gaussians Splatting method which combines the learning of the 3D Gaussians in canonical space with a deformation field MLP to achieve monocular dynamic scene reconstruction. The method can be divided into three main contributions: firstly, the *Differentiable Rendering Through 3D Gaussians Splatting in Canonical Space*; secondly, the *Deformable 3D Gaussians*; and lastly, the *Annealing Smooth Training*.

As can be seen in Figure 3.2, the deformation field gets the center location of the Gaussians $\gamma(\boldsymbol{c})$ and the time $\gamma(t)$ as input, where γ is a positional encoder defined as

$$\gamma(p) = (\sin(2^k \pi p), \cos(2^k \pi p))_{k=0}^{L-1}. \tag{3.6}$$

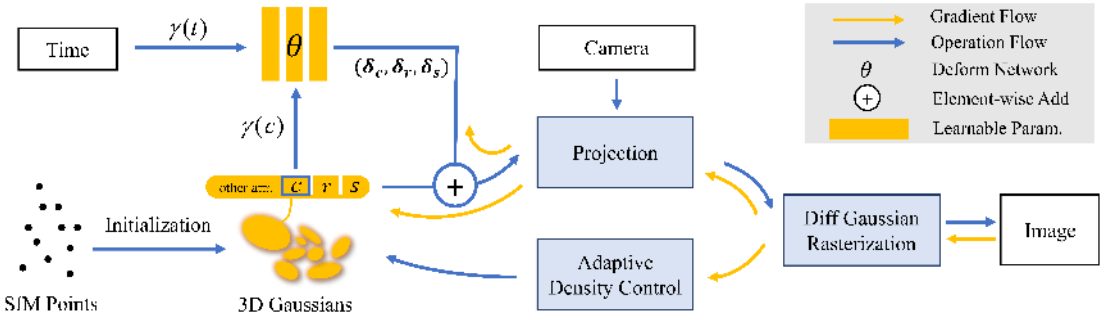


Figure 3.2: *Visualization of the DeformableGS Pipeline taking images, camera positions, timestamps, and initialized SfM points for real-world images as input. Positional encoded center location $\gamma(\boldsymbol{c})$ and time $\gamma(t)$ are processed by a deform network with learnable parameters θ. The outputs $\delta\boldsymbol{c}$, $\delta\boldsymbol{r}$, and $\delta\boldsymbol{s}$ are then added to the canonical 3D Gaussians, which model their position and shape at a specific time step. After projecting the 3D Gaussians to 2D, differentiable Gaussian rasterization enables joint optimization of the deformation network, the adaptive density control, and the 3D Gaussians. Learnable parameters are marked in yellow, operation and gradient flow with blue and yellow arrows, respectively. Figure taken from Ziyi Yang et al., 2024.*

For the reconstruction of synthetic scenes, the authors use positional encoding with $L = 10$ for x and $L = 6$ for t, while for real scenes, both x and t are encoded with $L = 10$. The MLP's output $\delta\boldsymbol{x}$, $\delta\boldsymbol{r}$ and $\delta\boldsymbol{s}$ are added to the Gaussian's attributes, resulting in a deformed 3D Gaussian $G(\boldsymbol{c} + \delta\boldsymbol{c}, \boldsymbol{r} + \delta\boldsymbol{r}, \boldsymbol{s} + \delta\boldsymbol{s}, k, \sigma)$.

3.4.1 Deformable 3D Gaussians

The centerpiece of DeformableGS is its deformation MLP consisting of eight fully connected layers with ReLU activations featuring 256-dimensional hidden layers. Note that a skip connection is introduced after the fourth layer by concatenating the input to the feature vector. As can be seen in Figure 3.3, the output of the final hidden layer is then fed into three separate fully connected layers without activation, leading to the desirable output of $(\delta\boldsymbol{x}, \delta\boldsymbol{r}, \delta\boldsymbol{s})$.

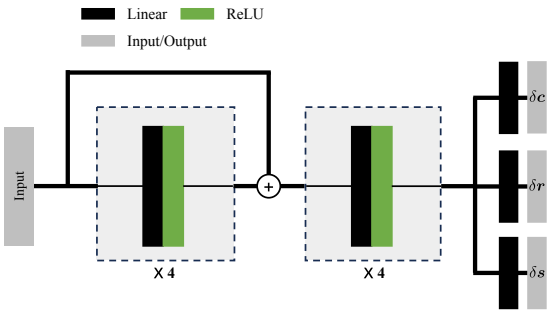


Figure 3.3: *Architecture of the Deformable 3DGS Multilayer Perceptron (MLP) which consists of Linear and ReLU layers combined with a skip connection. Figure adapted from Ziyi Yang et al., 2024*

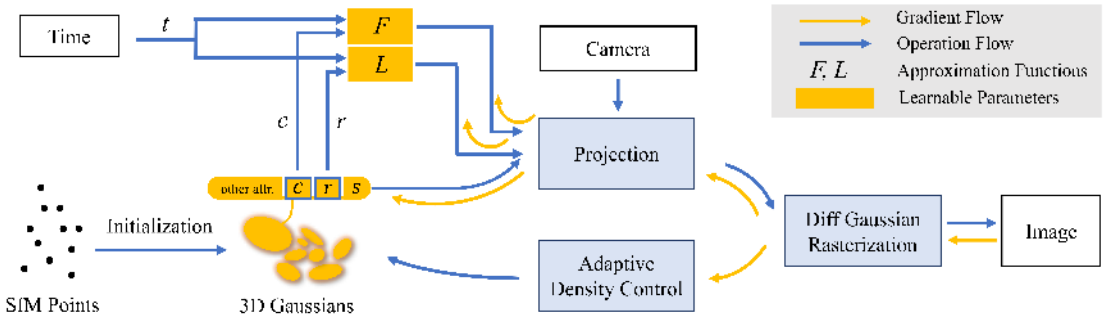


Figure 3.4: *Visualization of the EffGS Pipeline taking images, camera positions, timestamps, and initialized SfM points for real-world images as input. Center location* $\boldsymbol{c}$ *and rotation* $\boldsymbol{r}$ *are processed by a Fourier* (F) *and Linear* (L) *approximation function with learnable parameters, respectively. The output, along with the other attributes, then models their position and shape at a specific time step. After projecting the 3D Gaussians to 2D, differentiable Gaussian rasterization enables joint optimization of the approximation Functions, the adaptive density control, and the 3D Gaussians. Learnable parameters are marked in yellow, while operation and gradient flow are represented by blue and yellow arrows, respectively. Figure adapted from Ziyi Yang et al., 2024*

3.4.2 Annealing Smooth Training

As the goal of this paper is not only to handle novel-view rendering at a fixed time step $t = 0$ or $t = 4$ but rather to enable an interpolated rendering at time step $t = 1.5$ for instance, inaccuracies in pose estimations lead to irregular rendering jitters. To handle this problem, Ziyi Yang et al., 2024 proposes an Annealing Smooth Training (AST) mechanism by adding a fading Gaussian noise $\mathcal{X}(i)$ to the input of the deformation MLP:

$$\begin{aligned} \Delta &= \mathcal{F}_{\theta\theta}(\gamma(\text{sg}(\boldsymbol{x})), \gamma(t) + \mathcal{X}(i)), \\ \mathcal{X}(i) &= \mathbb{N}(0,1) \cdot \beta \cdot \Delta t \cdot (1 - i/\tau). \end{aligned} \tag{3.7}$$

Here, $\mathbb{N}(0,1)$ defines a standard normal distribution, scaled by an empirically chosen factor $\beta = 0.1$, the mean time interval Δt, and lastly a decay term $(1 - i/\tau)$, where τ is the annealing threshold iteration, set to $20k$ in practice.

3.5 A Compact Dynamic 3D Gaussian Representation for Real-Time Dynamic View Synthesis (EffGS)

In their paper *A Compact Dynamic 3D Gaussian Representation for Real-Time Dynamic View Synthesis*, Katsumata et al., 2024 also propose an extension of the 3DGS paper (Kerbl et al., 2023) by introducing a novel approach in modeling the motion of the Gaussians in a dynamic setting. The contributions of this work include the introduction of time-varying parameters for modeling the position and rotation of Gaussians as a combination of Fourier and Linear approximation (see Figure 3.4). Additionally, an optical flow-based loss term is incorporated into the reconstruction objective to improve performance in cases where only a limited number of views are available at each time step.

3.5.1 Motion Representation

Each Gaussian $G(\boldsymbol{c}, \boldsymbol{r}, \boldsymbol{s}, k, \sigma)$ consists of both time-invariant and time-dependent components. The time-invariant ones are the scaling $\boldsymbol{s}$, the spherical harmonics k, and the opacity σ. The remaining time-varying parameters are then defined as follows:

The 3D position of the dynamic Gaussian center $\boldsymbol{c}$ is approximated by means of a Fourier approximation at time t by

$$\begin{aligned} c_x(t) &= w_{x,0} + \sum_{i=1}^{L} w_{x,2i-1}\sin(2i\pi t) + w_{x,2i}\cos(2i\pi t), \\ c_y(t) &= w_{y,0} + \sum_{i=1}^{L} w_{y,2i-1}\sin(2i\pi t) + w_{y,2i}\cos(2i\pi t), \\ c_z(t) &= w_{z,0} + \sum_{i=1}^{L} w_{z,2i-1}\sin(2i\pi t) + w_{z,2i}\cos(2i\pi t), \end{aligned} \tag{3.8}$$

where $w_{\cdot,0}$ and $w_{\cdot,1}, \cdots, w_{\cdot,2L}$ are the learnable intercept and parameters, respectively. L is a hyperparameter defining the number of Fourier terms, also known as harmonics.

For the second time-varying parameter, the rotation parameter $\boldsymbol{r}$, the authors choose a simple linear approximation defined as follows

$$\begin{aligned} q_x(t) &= w_{qx,0} + w_{qx,1}t, & q_y(t) &= w_{qy,0} + w_{qy,1}t, \\ q_z(t) &= w_{qz,0} + w_{qz,1}t, & q_w(t) &= w_{qw,0} + w_{qw,1}t, \end{aligned} \tag{3.9}$$

where again $w_{\cdot,0}$ and $w_{\cdot,1}$ are the learnable intercept and coefficients defining the rotation of the Gaussian at time t.

Summing up, the number of parameters needed for a dynamic 3D Gaussian representation is not dependent on the length of the sequence. It is rather restricted on the one hand by the parameters of the time-varying parameters of the Gaussian center $\boldsymbol{c}$ and the rotation $\boldsymbol{r}$ with $3L$ and 8, respectively. Taking into account the definitions in Section 2.2.1, on the other hand, the time-invariant parameters scale, color, and opacity add another $3+3(k+1)^2+1$ parameters. As a result, the number of parameters per Gaussian is limited by $\#param = 3L+8+3+3(k+1)^2+1 = 3[L+(k+1)^2]+12 = 3(L+k^2+2k+5)$

3.5.2 Optimization

In order to quantify the quality of the reconstruction, Katsumata et al., 2024 not only uses the reconstruction loss $\mathcal{L}_{\text{recon}}$ as defined in Section 2.2.4 to compare the rendered and the training images. Especially altering shapes and object motion makes it difficult for the method to consistently share scene information across frames at different time steps. In order to address this uncertainty of reconstruction of time steps where only few viewpoints capture the scene, a new flow loss $\mathcal{L}_{\text{flow}}$ is introduced. Adding it with

a balancing hyperparameter to the reconstruction loss $\mathcal{L}_{\text{recon}}$ yields

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_{\text{flow}} \mathcal{L}_{\text{flow}}(\hat{F}, F). \tag{3.10}$$

The loss is then computed by taking the L_1 distance of the pseudo optical flow of the Gaussians $\hat{F}$ and the ground truth flow $F = \{f_{\text{fwd}}, f_{\text{bwd}}\}$ obtained by RAFT (Teed et al., 2020). In order to be able to compare the flows in 2D on the image plane, Katsumata et al., 2024 present a multi-step approach that computes and maps the scene flow into a 2D camera plane to get $\hat{F}$. Firstly, the pseudo-optical forward flow is computed for each dimension by subtracting the Gaussian center position queried from the Fourier approximation at time step t from its position at time step $t + \Delta t$. The backward flow is computed similarly, subtracting the Gaussian center position at $t - \Delta t$ from its position at time step t, which can be summarized in the following six equations:

$$\begin{aligned}
\hat{f}^{c_x}_{\text{fwd}} &= c_x(t + \Delta t) - c_x(t), & \hat{f}^{c_x}_{\text{bwd}} &= c_x(t) - c_x(t - \Delta t), \\
\hat{f}^{c_y}_{\text{fwd}} &= c_y(t + \Delta t) - c_y(t), & \hat{f}^{c_y}_{\text{bwd}} &= c_y(t) - c_y(t - \Delta t), \\
\hat{f}^{c_z}_{\text{fwd}} &= c_z(t + \Delta t) - c_z(t), & \hat{f}^{c_z}_{\text{bwd}} &= c_z(t) - c_z(t - \Delta t).
\end{aligned} \tag{3.11}$$

The next step is to project the flows into the 2D camera plane by multiplying the Jacobian of the affine approximation of the projective transformation, as described in Section 2.2.2, with the stacked and transposed forward and backward flows of each dimension

$$\hat{f}^{c_x c_y c_z}_{\{\text{fwd,bwd}\}} = \mathbf{J} \left[\hat{f}^{c_x}_{\{\text{fwd,bwd}\}}, \hat{f}^{c_y}_{\{\text{fwd,bwd}\}}, \hat{f}^{c_z}_{\{\text{fwd,bwd}\}} \right]^\top. \tag{3.12}$$

Finally, the authors use the same approach as described in Section 2.2.3, replacing the color c_i with the computed optical forward and backward flow of the i-th Gaussian $\hat{f}^{c_x c_y c_z}_{\text{fwd},i}$ and $\hat{f}^{c_x c_y c_z}_{\text{bwd},i}$, respectively. Together with the unchanged α blending, the final point-based rendering is defined as

$$\hat{f}_{\text{fwd}} = \sum_{n=1}^{N} \hat{f}^{c_x c_y c_z}_{\text{fwd,i}} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \qquad \hat{f}_{\text{bwd}} = \sum_{n=1}^{N} \hat{f}^{c_x c_y c_z}_{\text{bwd,i}} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \tag{3.13}$$

This is one example of how a data-driven prior, in this case optical flow, can be used to aid reconstruction in under-constrained settings. However, as can be seen later in Section 3.7, incorporating optical flow supervision during training is optional rather than a necessary requirement when implementing *EffGS*.

3.6 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering (4D-GS)

The paper *4D Gaussian Splatting for Real-Time Dynamic Scene Rendering*, hereafter referred to as *4D-GS* by Wu et al., 2024, proposes a dynamic Gaussian Splatting method that combines the explicit representation of a scene via 3D Gaussians with an implicit

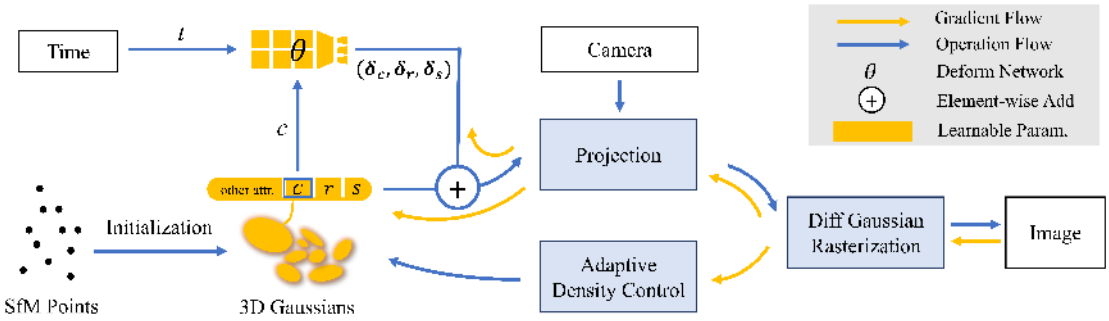


Figure 3.5: *Visualization of the 4D-GS Pipeline taking images, camera positions, timestamps and initialized SfM points for real world images as input. Center location* $\boldsymbol{c}$ *and time* t *are processed by a HexPlane-inspired network* $\mathcal{F}_\theta$ *with learnable parameters. The outputs* $\delta\boldsymbol{c}$, $\delta\boldsymbol{r}$, *and* $\delta\boldsymbol{s}$ *are then added to the canonical 3D Gaussians, which model their position and shape at a specific time step. After projecting the 3D Gaussians to 2D, differentiable Gaussian rasterization enables joint optimization of the network, the adaptive density control, and the 3D Gaussians. Learnable parameters are marked in yellow, while operation and gradient flow are represented by blue and yellow arrows, respectively. Figure taken from Ziyi Yang et al., 2024.*

4D neural voxel encoding in latent space. For this purpose, they extend the work *K-Planes: Explicit Radiance Fields in Space, Time, and Appearance* by Fridovich-Keil et al., 2023 to make use of the explicit 3D Gaussian Splatting representation.

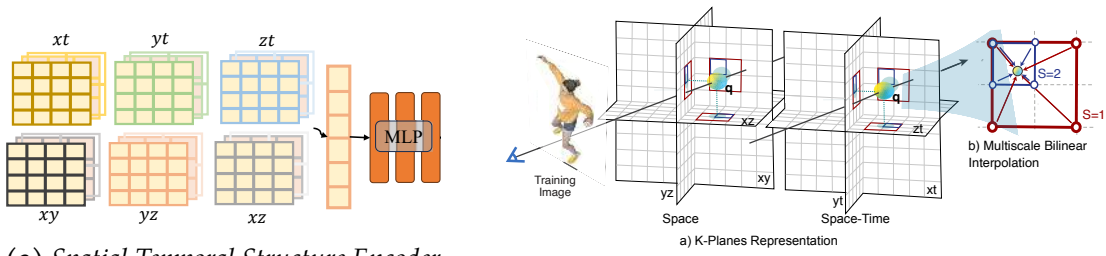
As can be seen in Figure 3.5, similar to *Deformable GS* (Ziyi Yang et al., 2024), the motion of the Gaussians is modeled using a neural network which outputs the rate of change for the center $\delta\boldsymbol{c}$, rotation δr and scale δs resulting in the deformed Gaussian $G(\boldsymbol{c} + \delta\boldsymbol{c}, \boldsymbol{r} + \delta\boldsymbol{r}, \boldsymbol{s} + \delta\boldsymbol{s}, k, \sigma)$. However, the architecture of this so-called *Gaussian Deformation Field Network* is inspired by the *K*-Planes method (Fridovich-Keil et al., 2023), consisting of a *Spatial-Temporal Structure Encoder* (see Figure 3.6a) and a *Multi-head Gaussian Deformation Decoder*, which will be presented in the following.

3.6.1 Spatial-Temporal Structure Encoder

An effective and efficient way of modelling 3D Gaussians' features is to use a multi-resolution HexPlane $R(i, j)$ structure followed by a tiny MLP $\mathcal{F}_{\theta_d}$ like it has been previously done in Fridovich-Keil et al., 2023 and Cao et al., 2023. Instead of using highly memory-consuming vanilla 4D neural voxels, the authors use six multi-resolution planes such that the position of the 3D Gaussians are enclosed in the bounding plane voxels $R(x, y)$, $R(x, z)$, $R(y, z)$ and the deformation over time in the temporal voxels $R(x, t)$, $R(y, t)$, $R(z, t)$. Together with the tiny MLP $\mathcal{F}_{\theta_d}$, the encoder $\mathcal{H}(G, t)$ can be defined as

$$\mathcal{H}(G,t) = \{R_l(i,j), \mathcal{F}_{\theta_d} \,|\, (i,j) \in \{(x,y),(x,z),(y,z),(x,t),(y,t),(z,t)\}, l \in \{1,2\}\}, \tag{3.14}$$

where the dimensions of each plane are defined by $R_l(i, j) \in \mathbb{R}^{h \times lN_i \times lN_j}$. Here, hidden dimensions of the features are denoted by h, the basic resolution of the voxel grid by N, and the upsampling scale by l. To evaluate the value of the HexPlane module of a 4D-point $\boldsymbol{q} = (x, y, z, t)$, the authors follow the procedure described in Fridovich-



(**a**) *Spatial-Temporal Structure Encoder*

(**b**) *K-Planes Representation*

Figure 3.6: *Core architectural components of the 4D-GS method. (a) The Spatial-Temporal Structure Encoder is composed of six multi-resolution planes followed by a lightweight MLP $\mathcal{F}_{\theta_d}$. (b) Visualization of the underlying K-Planes representation, illustrating the six spatial-temporal planes in space. The figure also shows the Multiscale Bilinear Interpolation process for scales $s = 1$ and $s = 2$, where features are obtained by querying the four neighboring voxels. Figures taken from Wu et al., 2024 and Fridovich-Keil et al., 2023, respectively.*

Keil et al., 2023 (see Figure 3.6b) where the point first gets projected into each of the six planes. Then, for each plane, bilinear interpolation (b-interpol) is performed and afterwards multiplied with each other. As the interpolation is done using all scales $l \in \{1, 2\}$, the results are concatenated in the end, which can be formalized via

$$\begin{aligned} f_h &= \bigcup_l \prod_{(i,j)} \text{b-interpol}(R_l(i,j)) \\ (i,j) &\in \{(x,y),(x,z),(y,z),(x,t),(y,t),(z,t)\}, \end{aligned} \tag{3.15}$$

where $f_h \in \mathbb{R}^{h*l}$ is the feature of the neural voxel. Finally, these features are passed through a tiny MLP $\mathcal{F}_{\theta_d}$ which merges all of them by $f_d = \mathcal{F}_{\theta_d}(f_h)$.

3.6.2 Multi-head Gaussian Deformation Decoder

In order to get the deformation values δ_c, δ_r, and δ_s, the features in latent space need to be decoded using the proposed *Multi-head Gaussian Deformation Decoder* $\mathcal{D}(\mathcal{F}_{\theta_c}, \mathcal{F}_{\theta_r}, \mathcal{F}_{\theta_s})$. This is done by feeding the output of the tiny MLP f_d into the three different MLPs computing the deformation of center $\delta\boldsymbol{c} = \mathcal{F}_{\theta_c}(f_d)$, rotation $\delta\boldsymbol{r} = \mathcal{F}_{\theta_r}(f_d)$ and scaling $\delta\boldsymbol{s} = \mathcal{F}_{\theta_s}(f_d)$. As a result, the deformed Gaussian $G(\boldsymbol{c}', \boldsymbol{r}', \boldsymbol{s}', k, \sigma)$ is composed by $\boldsymbol{c}' = \boldsymbol{c} + \delta\boldsymbol{c}$, $\boldsymbol{r}' = \boldsymbol{r} + \delta\boldsymbol{r}$ and $\boldsymbol{s}' = \boldsymbol{s} + \delta\boldsymbol{s}$.

3.6.3 Optimization

Different from 3DGS (Kerbl et al., 2023), Wu et al., 2024 do not apply a structural dissimilarity loss $\mathcal{L}_{\text{D-SSIM}}$ in combination with the L1 color loss $\mathcal{L}_1$. Instead they use a grid-based total-variational loss $\mathcal{L}_{\text{TV}}$ also used by Fridovich-Keil et al., 2023 which is defined as

$$\mathcal{L}_{\text{TV}}(R) = \frac{1}{|R|\, lN_i\, lN_j} \sum_{(i,j),u,v} (\|R(i,j)^{u,v} - R(i,j)^{u-1,v}\|_2^2 + \|R(i,j)^{u,v} - R(i,j)^{u,v-1}\|_2^2). \tag{3.16}$$

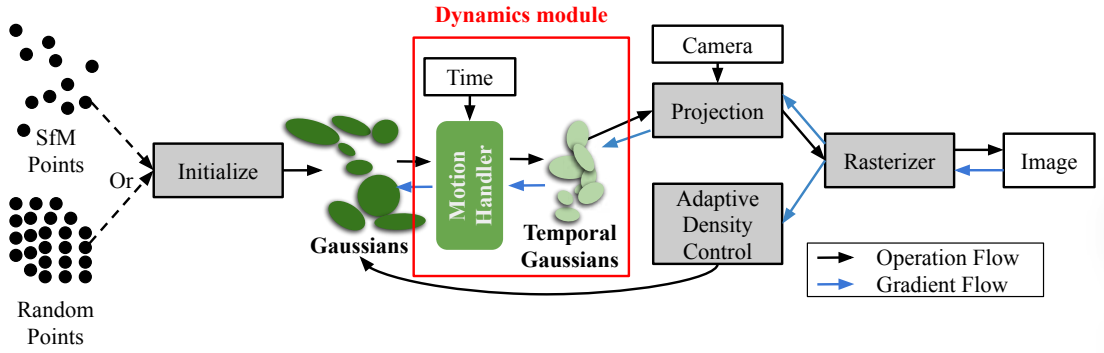


Figure 3.7: *Illustration of the 3DGS-inspired architecture extended with a Dynamics Module that enables the motion of Gaussians over time. The Motion Handler block is the key component where the previously discussed methods differ. Figure taken from Liang et al., 2025.*

This loss targets to reward smooth prior over edges with the L2 loss enforcing smooth gradients and $R(i,j)^{u,v}$ denoting plane $(i,j) \in \{(x,y),(x,z),(y,z),(x,t),(y,t),(z,t)\}$ at location (u,v) with $|R|$ indicating the number of planes. Without using a weighting hyperparameter, the total loss is defined as the sum of the L1 loss $\mathcal{L}_1$ and the grid-based total-variational loss $\mathcal{L}_{\text{TV}}$

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_{\text{TV}}. \tag{3.17}$$

3.7 Monocular Dynamic Gaussian Splatting: Fast, Brittle, and Scene Complexity Rules (MonoDyBench)

This section presents the key publication *Monocular Dynamic Gaussian Splatting: Fast, Brittle, and Scene Complexity Rules* (*MonoDyBench*) of the thesis by Liang et al., 2025, which builds the foundation of the experiments presented in Chapter 4 and Chapter 5.

As already stated in the introduction to this chapter, since the release of the seminal work *3D Gaussian Splatting for Real-Time Radiance Field Rendering* by Kerbl et al., 2023, which pioneered Gaussian Splatting for reconstruction of unbounded and complete static scenes, numerous papers have explored extensions of this approach to dynamic scenes. Among the numerous notable works published over the past two years, the authors have selected five publications as a representative and diverse set of dynamic Gaussian Splatting techniques. Notably, neither of them was evaluated against each other but primarily against NeRF-based methods like *TiNeuVox* (Fang et al., 2022) or *D-NeRF* (Pumarola et al., 2021) as well as the static *3D-GS* (Kerbl et al., 2023) method. *MonoDyBench* aims to close this gap by systematically benchmarking all the methods presented in this chapter against each other in a unified evaluation framework, whose architecture is illustrated in Figure 3.7. The following summarizes the author's key findings and observations along with a brief overview of the evaluation framework, including datastes, metrics, and other relevant details.

The authors evaluate the compared models across five datasets: Nerfies (Park et

al., 2021b), iPhone (Gao et al., 2022), HyperNeRF (Park et al., 2021a), NeRF-DS (Yan et al., 2023), and D-NeRF (Pumarola et al., 2021), with the latter being the only synthetic dataset. To quantify the reconstruction quality, Liang et al., 2025 choose a set of commonly used metrics, including Peak Signal-to-Noise Ratio (PSNR), which measures pixel-wise fidelity, Structural Similarity (SSIM) (Z. Wang et al., 2004), which evaluates structural similarity, its multi-scale variant Multi-Scale Structural Similarity (MS-SSIM) (Z. Wang et al., 2003), which incorporates structural similarity across different resolutions, and Learned Perceptual Image Patch Similarity (LPIPS) (Zhang et al., 2018), a perceptual metric based on deep feature distances. Additionally, the authors report training time and rendering speed measured in Frames Per Second (FPS). To more accurately evaluate the reconstruction quality of the motion, they also employ the masked versions of the metrics that only evaluate the dynamic parts of the scene. Finally, for every method and scene in a dataset, three independent runs are conducted on each scene to measure performance variability, with both the mean and standard deviation reported for each experiment.

What is important to highlight, however, is that the *MonoDyBench* framework does not implement the optical flow supervision for *EffGS* as the authors had foreseen. Furthermore, none of the models implemented and evaluated in this benchmarking pipeline employs any kind of regularization of Gaussian motion. This stands in contrast to approaches such as *Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis* by Luiten et al., 2024, which integrate special motion-regularization losses. This includes, for example, the *short-term local-rigidity loss* that enforces nearby Gaussians to perform transformations consistent with locally rigid motion. Or a *local-rotation similarity loss* that restricts the rotation of each Gaussian to be similar to that of its twenty nearest neighboring Gaussians.

3.7.1 Comparative Evaluation of Method Performance

As can be seen in Table 3.1, none of the evaluated methods consistently outperforms the others across all metrics and datasets. While *TiNeuVox* achieves the best or second-best results in PSNR, SSIM, MS-SSIM, and training time, this advantage comes with the

Table 3.1: ***Summary of Quantitative Results.*** *Quantitative comparison of all evaluated methods, averaged over the five benchmark datasets. Table taken from Liang et al., 2025.*

Method\Metric	PSNR↑	SSIM ↑	MS-SSIM ↑	LPIPS ↓	FPS ↑	TrainTime (s) ↓
TiNeuVox	**24.54**	0.706	**0.804**	0.349	0.29	**2664.89**
3DGS	19.48	0.651	0.688	0.358	**243.47**	2964.81
EffGS	21.84	0.672	0.725	0.347	177.21	3757.81
STG-decoder	21.81	0.678	0.742	0.352	109.42	5980.64
STG	19.51	0.583	0.643	0.475	181.70	5359.56
DeformableGS	24.07	0.694	0.755	0.283	20.20	6227.43
4DGS	23.55	**0.708**	0.765	**0.277**	62.99	8628.89
RTGS	21.61	0.663	0.720	0.350	143.37	7352.52

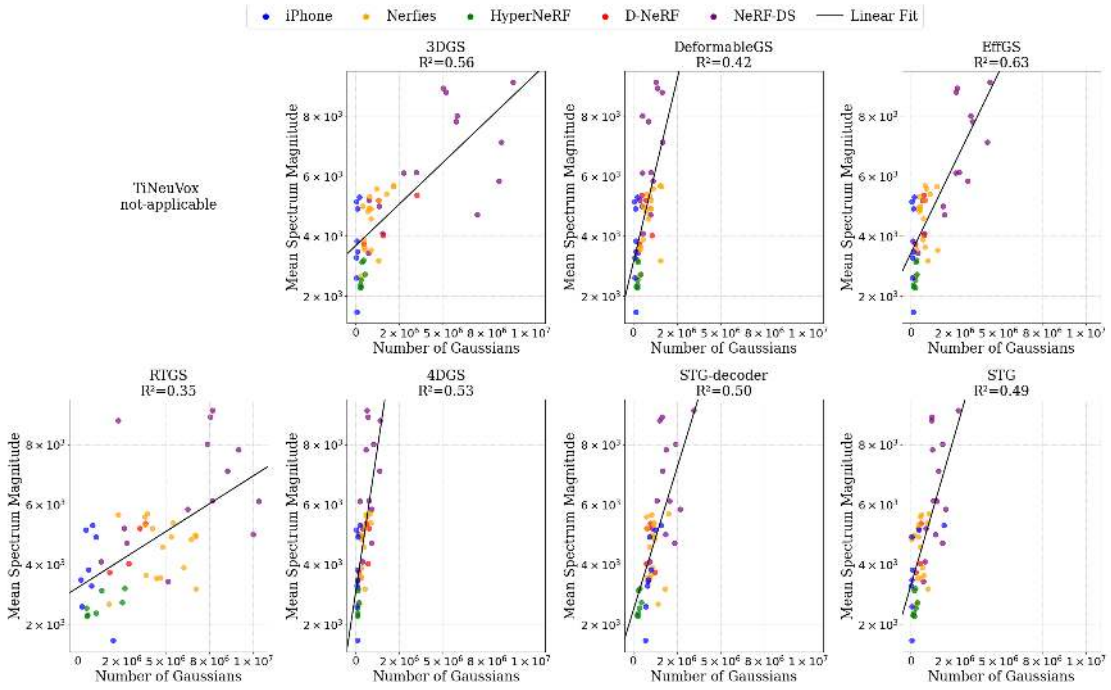


Figure 3.8: *Relationship between scene frequency and the number of Gaussians used for reconstruction across all datasets, showing that scenes with high frequency require a larger number of Gaussians for reconstruction. Figure taken from Liang et al., 2025.*

cost of very low rendering speeds of 0.29 FPS compared to 20-181 FPS for Gaussian-based methods. When considering performance across individual datasets, a similar trend emerges: No single method dominates across all metrics. From this, the authors conclude that "dataset variations overwhelm Gaussian method variations", implying that differences between datasets have a greater impact than architectural changes among Gaussian Splatting models. Interestingly, the results also indicate that low-dimensional motion representations are not necessarily a hindrance. Lightweight Fourier and linear motion functions, as used in *EffGS*, even lead to slightly better results across all metrics than the high-capacity 4D representation of *RTGS*, while requiring almost half the training time. Although *DeformableGS* claims to outperform NeRF-based methods on the NeRF-DS dataset, known for its specular objects, this does not imply a general structural superiority of Gaussian Splatting-based approaches. Liang et al., 2025 clarify that "specular objects are a challenge for all methods", a conclusion supported by the per-method results on the NeRF-DS dataset, where only *DeformableGS* shows a noticeable advantage.

3.7.2 Impact of Scene Frequency on Convergence

To further analyze the factors influencing reconstruction performance, Liang et al., 2025 investigate the relationship between scene frequency content and model convergence behavior (see Figure 3.8). Using a Fast Fourier Transform (FFT), the authors capture the mean frequency across all images of a scene and dataset in order to obtain a quan-

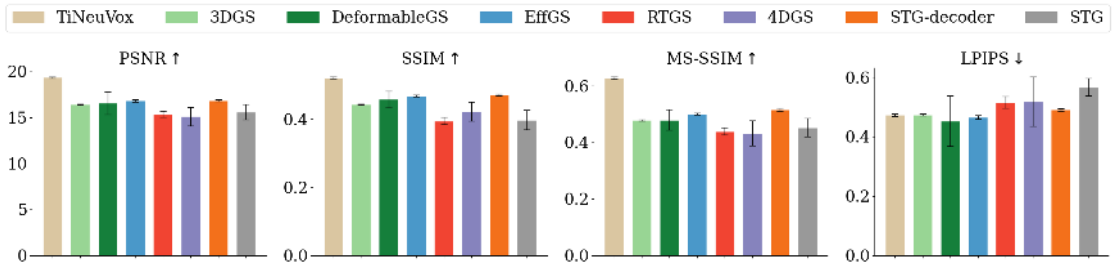


Figure 3.9: *Quantitative comparison of all methods evaluated on the strictly-monocular iPhone dataset. Figure taken from Liang et al., 2025.*

titative measure of texture complexity and detail. This allows the authors to make the following point: scenes with high frequency require a larger number of Gaussians to accurately reconstruct fine textures and shapes. Thus, these scenes show slower convergence during training, underscoring the trade-off between reconstruction quality and computational efficiency in Gaussian Splatting-based methods.

3.7.3 Influence of Multi-View Cues on Reconstruction Quality

Another key observation by Liang et al., 2025 is that multi-view cues are more important for Gaussian Splatting-based methods than for NeRF-based ones. This claim is supported by experiments on the iPhone dataset (Gao et al., 2022), which provides *strictly-monocular* scenes emphasizing natural camera motion rather than designated multi-view captures. Figure 3.9 shows that except for LPIPS, *TiNeuVox* significantly outperforms all Gaussian-based methods on this dataset across all metrics, suggesting that these methods are disproportionately vulnerable to the absence of multi-view cues. As a consequence, the authors conclude that while Gaussian-based methods narrow the gap in multi-view settings, the ”lack of multi-view cues hurts dynamic Gaussians more”.

4 Methodology

This chapter presents the methodological foundations underlying the contributions of this thesis, divided into two major parts: the evaluation setup for benchmarking monocular depth prediction models, and the architectural design for integrating depth priors into the dynamic 3D reconstruction pipeline.

The first part, introduced in Section 4.1, describes the evaluation framework used to benchmark monocular depth estimation models. For this, three different alignment strategies are outlined in Section 4.1.1, which ensure fair comparisons of the predicted depth maps to the ground truth. The quantitative metrics used throughout the benchmark are then presented in Section 4.1.2.

In Section 4.2, the core architectural choices enabling the incorporation of depth priors into the reconstruction system are presented. This includes structural modifications to the underlying framework (see Section 4.2.1), the formulation of the depth supervision mechanism (see Section 4.2.2), and the initialization procedure for placing Gaussian primitives based on depth information (see Section 4.2.3).

4.1 Evaluation of Depth Prediction Models

Depth prediction models can generally be categorized into two types based on their output representation. The first type produces affine-invariant depth, also called *relative depth* maps, where the predicted values are normalized to a fixed range, typically $d \in [0, 1]$. The second type predicts *metric depth*, which attempts to recover the absolute distance of scene points from the camera in meters. The latter is typically preferred for downstream tasks such as 3D reconstruction because it introduces global scale consistency and meaningful geometric constraints that benefit the optimization of dynamic Gaussian Splatting. The goal is therefore to find a depth prediction model that satisfies the two following key objectives:

1. **High-quality predictions:** To best help the dynamic Gaussian Splatting approaches, the depth prediction method should offer robust and accurate performance

on real-world scenes. This includes capturing fine geometric details while maintaining smooth and consistent predictions for surfaces.

2. **Temporal consistency:** The method should estimate temporal consistent and coherent depth across consecutive frames within a video sequence.

To evaluate those two goals, the iPhone dataset by Gao et al., 2022 serves as an ideal benchmark, as it contains casually captured monocular videos paired with Light Detection and Ranging (LiDAR)-based ground truth. However, this setup comes with several limitations: LiDAR-derived depth maps can be sparse due to incomplete measurements at object boundaries or distant regions; additionally, they often struggle to capture high-frequency details. Nevertheless, it is one of the few practical options to obtain depth measurements in real-world scenes.

4.1.1 Depth Map Alignment Strategies

The alignment of a predicted depth map $\hat{d}$ to a reference depth map d refers to the process of estimating a scale and shift value pair (s, t) such that the transformed prediction $\hat{d}^{\text{aligned}} = s \cdot \hat{d} + t$ minimizes the deviation to the reference depth map d according to a chosen error metric.

As Gaussian Splatting methods are trained in a normalized coordinate frame, perfectly accurate metric depth values are not strictly required, as long as consistent scale and shift values can be estimated. To ensure a fair and meaningful comparison between depth prediction models, a preliminary alignment step is therefore allowed, where the predicted depth map is aligned to the corresponding LiDAR depth. Two alignment strategies are investigated:

Median Based Alignment

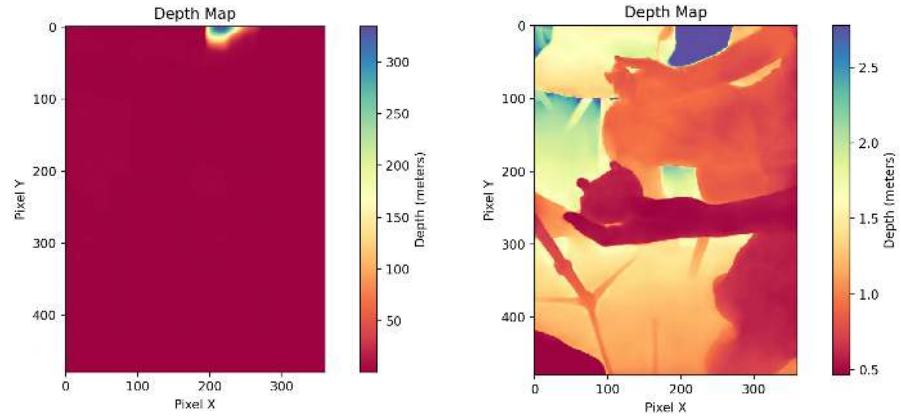
This alignment strategy is adapted from *Shape of Motion: 4D Reconstruction from a Single Video* by Q. Wang et al., 2025, a recent dynamic 3D reconstruction method that uses the following median-based approach to align depth maps used for depth supervision during training. In the context of this thesis, however, the same procedure is applied to align predicted depth maps to a ground truth reference.

For each frame $f \in \{1, \dots, F\}$ in the scene, let $d_f \in \mathbb{R}^{H \times W}$ be the reference LiDAR depth map and $\hat{d}_f \in \mathbb{R}^{H \times W}$ the corresponding predicted depth map. First, both depth maps are centered by subtracting their median values to receive the median-centered (mc) depth maps d^{mc} and $\hat{d}^{mc}$:

$$d_f^{mc} = d_f - \text{median}(d_f) \qquad \hat{d}_f^{mc} = \hat{d}_f - \text{median}(\hat{d}_f). \tag{4.1}$$

Then, the scale s_f and shift t_f for each frame f are estimated as:

$$s_f = \text{median}\left(\frac{d_f^{mc}}{\hat{d}_f^{mc}}\right) \qquad t_f = d_f^{mc} - s_f \cdot \hat{d}_f^{mc}. \tag{4.2}$$



(a) (b)

Figure 4.1: *Visualization of the same predicted Video Depth Anything depth map: (a) original output showing an outlier region with extreme depth values (~ 300 m), (b) clipped visualization with depth values only smaller than 2.7 to reveal the underlying depth map*

However, to reward temporal consistency of the prediction across frames, a single global scale and shift are applied for each scene by averaging the per-frame estimates:

$$\hat{d}_f^{\text{aligned}} = \bar{s} \cdot \hat{d}_f + \bar{t} \qquad \text{with} \qquad \bar{s} = \frac{1}{F}\sum_{f=1}^{F} s_f \quad \text{and} \quad \bar{t} = \frac{1}{F}\sum_{f=1}^{F} t_f. \tag{4.3}$$

It should be noted that LiDAR maps in general are often sparse and noisy. This is the reason why scale and shift are estimated only from pixels containing valid LiDAR values, instead of using all available pixels.

Random Sample Consensus (RANSAC) Alignment

Although the previously explained median-based approach can handle certain outliers, it assumes that the majority of pixel correspondences are inliers. However, LiDAR maps, and especially depth prediction methods, can sometimes produce extreme values, either very small or very large, within small image regions, which may undesirably affect the median computation (see Figure 4.1). For this reason, the performance of the depth estimation models is also evaluated using the Random Sample Consensus (RANSAC) alignment algorithm. In contrast to the previous method, scale and shift values are not averaged after estimating them per frame, but are instead jointly estimated for the entire scene by stacking all valid pixels of all frames into a single array and passing it to the RANSAC algorithm. In the implementation used for this thesis, the alignment is performed using the `sklearn.linear_model.RANSACRegressor` with a `LinearRegression` base estimator. The procedure can be summarized as follows:

First, three pixels (p_1, p_2, p_3) are randomly sampled from all valid pixels of the scene and used to obtain an initial estimate of the scale and shift parameters, introducing non-determinism into the alignment process. Next, the absolute residuals between the aligned and reference depths are computed, and the number of pixels whose residuals

fall below a predefined threshold of $\tau = 0.3$ is counted:

$$N_{\text{inliers}} = \sum_{p} \mathbb{I}[r(\boldsymbol{p}) < \tau] \qquad \text{with} \qquad r(t) = |d(\boldsymbol{p}) - s \cdot \hat{d}(\boldsymbol{p}) + t|, \tag{4.4}$$

where $\mathbb{I}[\cdot]$ denotes the Iverson bracket, which evaluates to 1 if the condition inside the brackets is true and 0 otherwise. This procedure is repeated for a fixed number of iterations, where the total count is determined by the overall size of the scene, including the number of frames and their spatial resolution. Finally, the inliers from the best estimate are used to compute the final scale and shift values using Least Squares regression.

4.1.2 Quantitative Evaluation Metrics

The evaluation metrics are chosen following common practices in the quantitative assessment of depth prediction methods, as used for example in *MoGe* (R. Wang et al., 2025). The following section briefly introduces each metric, along with its mathematical definition based on formulations from Suhoi, 2024, and its intended interpretive goal. Let in the following d_i denote the LiDAR ground truth depth and $\hat{d}_i$ the predicted depth at pixel $\boldsymbol{p}_i$ with $i \in \{1, \ldots, N\}$.

Mean Squared Error (MSE)

The Mean Squared Error (MSE) is one of the most fundamental error measures, computing the average of the squared differences between predicted and reference depths:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(d_i - \hat{d}_i \right)^2 \tag{4.5}$$

This metric penalizes large deviations more strongly due to the squaring term, making it particularly sensitive to outliers. However, MSE can be problematic in the context of depth prediction, as it does not account for the relative nature of depth errors. In particular, a deviation of one meter is far less significant in an outdoor scene with an overall depth range of up to 500 meters than in an indoor setting with a maximum depth of just one meter, for instance. Despite this difference in importance, MSE treats both cases equally, which results in disproportionately high error values for large-scale datasets.
Interpretation: Lower values indicate a smaller depth prediction error and more accurate absolute depth estimation, although careful consideration is required when comparing datasets with different depth ranges.

Absolute Relative Error (AbsRel)

The Absolute Relative Error (AbsRel) is a widely used metric for evaluating monocular depth prediction, as it normalizes the absolute difference between predicted and

ground truth depths by the ground truth depth before taking the average:

$$AbsRel = \frac{1}{N}\sum_{i=1}^{N}\frac{|d_i - \hat{d}_i|}{d_i} \tag{4.6}$$

In contrast to MSE, this metric quantifies the relative dimension of errors, making it more meaningful when comparing across datasets with different depth ranges. As a consequence, settings as described above, where objects are hundreds of meters away, are less penalized than with the MSE metric.
Interpretation: Lower AbsRel values indicate a smaller relative depth prediction error and more accurate absolute depth estimation, making this metric especially suitable for comparing methods across datasets with diverse depth scales.

Threshold Accuracy (δ_1)

Threshold Accuracy (δ_1) measures the proportion of pixels for which the predicted depth deviates from the ground truth by less than a certain threshold. It is formally defined as:

$$\delta_1 = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[\max\left(\frac{d_i}{\hat{d}_i}, \frac{\hat{d}_i}{d_i}\right) < 1.25\right], \tag{4.7}$$

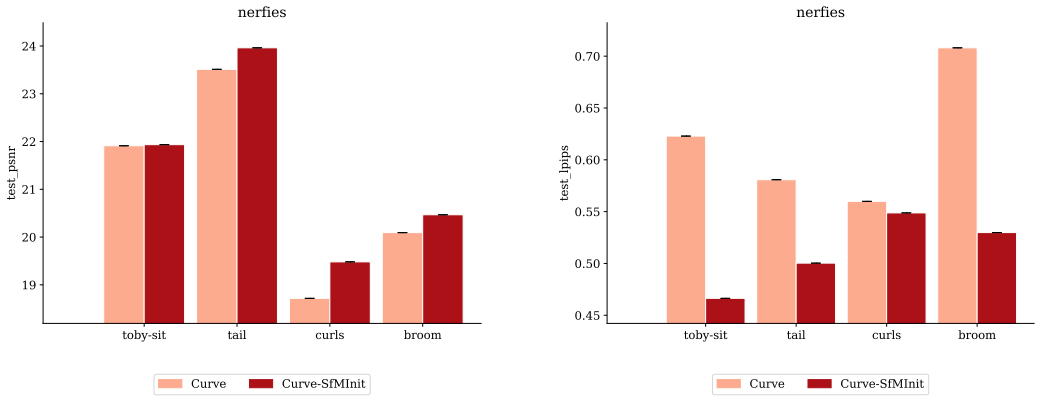
where the suffix determines the allowed deviation from the ground truth, with δ_1, δ_2, δ_3 corresponding to thresholds of 1.25, 1.25^2, and 1.25^3, respectively. Similar to the AbsRel, this metric also captures the relative magnitude of prediction errors, making it useful for datasets with varying depth ranges. It is particularly useful for assessing the overall robustness of a model, as it is less sensitive to large outliers than MSE or AbsRel.
Interpretation: A higher δ_1 value indicates better performance, as a large proportion of depth estimates fall within an acceptable range of the true depth.

4.2 Integration of Depth Priors into 3D Reconstruction

Integrating depth priors into the reconstruction pipeline can improve scene geometry estimation by providing additional geometric constraints beyond photometric consistency. This section outlines the proposed modifications required to incorporate depth information into the dynamic Gaussian Splatting framework. Specifically, it introduces *General Framework Modifications* in Section 4.2.1, the proposed *Depth Supervision Mechanism* in Section 4.2.2, as well as details for the *Depth-based Initialization Procedure* in Section 4.2.3.

4.2.1 General Framework Modifications

In contrast to the original *MonoDyBench* framework (Liang et al., 2025), several modifications are introduced in the following sections to better support the integration



(a) **(b)**

Figure 4.2: *Effect of enabling Structure-from-Motion Initialization for the EffGS Katsumata et al., 2024 method evaluated on the Nerfies test set: (a) PSNR, where higher values indicate better reconstruction quality, (b) LPIPS, where lower values indicate better perceptual similarity. A substantial performance improvement is particularly evident in LPIPS.*

of depth priors into the 3D reconstruction process. This comprises *Black Background Rendering* in Section 4.2.1, *Structure-from-Motion Initialization for EffGS* in Section 4.2.1, *Gaussian Count Limit* in Section 4.2.1, *Motion Model Weight Initialization Strategy* in Section 4.2.1 and finally *Rasterizer Configuration Adjustments* in Section 4.2.1.

Black Background Rendering

The *MonoDyBench* framework renders a white background for all datasets, regardless of whether they are synthetic or real. In contrast, the *3D Gaussian Splatting* paper (Kerbl et al., 2023) recommends adapting the background color depending on the nature of the dataset. Following this convention, the comparison framework in this thesis uses a white background for synthetic datasets and a black background for real-world datasets.

Structure-from-Motion Initialization for Curve

In the baseline *MonoDyBench* configuration, different Gaussian initialization strategies are applied, depending on the reconstruction method: *DeformableGS* (Ziyi Yang et al., 2024) and *4D-GS* (Wu et al., 2024) use initialization via Structure-from-Motion (SfM) points, whereas *EffGS* (Katsumata et al., 2024) initializes with random points. However, experimental results show that *EffGS* also benefits from SfM-based initialization, leading to improved reconstruction quality (see Figure 4.2).

Gaussian Count Limit

In comparison to *EffGS* or *4D-GS*, which typically require around 300 to 800 thousand Gaussian primitives to reconstruct a scene, *DeformableGS* generally operates with 1.0 to 1.5 million Gaussians, with certain scenes such as *backpack* and *paper-windmill* exceed-

ing 3.0 million. Even on the Nvidia A100 GPU used in these experiments, this leads to frequent out-of-memory failures. To ensure successful training, a limit of 2.5 million Gaussians is imposed. The cap is implemented in a way such that if the system attempts to exceed the limit, only a random subset of the proposed new Gaussians is admitted to fill the remaining available slots. Conversely, if pruning reduces the total number of Gaussians in the scene, new Gaussians up to the defined limit may again be created through splitting or cloning when required.

Motion Model Weight Initialization Strategy

This section concerns the multi-layer perceptrons (MLPs) used in *DeformableGS* (Ziyi Yang et al., 2024) and *4D-GS* (Wu et al., 2024), since for *EffGS*, the coefficients of the Fourier and linear approximation functions are already initialized to zero as described in the *MonoDyBench* (Liang et al., 2025) paper.

In many machine learning applications, randomized or carefully designed weight initializations can be beneficial for training diversity and convergence. However, for the deformation models used in this framework, standard initialization techniques applied to the final layers, such as Xavier (Glorot et al., 2010) in *4DGS* or He initialization (He et al., 2015) for *DeformableGS*, introduce unnecessary instability to the early optimization process. As illustrated in Figure 4.3, these weight initialization procedures can lead to erroneous deformations at the beginning of training, partially destroying previously learned geometry. To avoid this, it is a common approach to initialize the weights and biases of the final layers to zero. This ensures that, at the start of deformation model training, predicted deltas for Gaussian center positions, rotations, and scales are all zero, providing a non-distorted starting point for the deformation process.

However, this approach must be applied with care. For certain architectures, this may lead to vanishing gradients, as is the case for *4DGS*. As a consequence, this leads

(a) (b) (c)

Figure 4.3: *Gaussian reconstruction results: (a) before applying any deformation model, (b) after deformation with DeformableGS (Ziyi Yang et al., 2024), and (c) after deformation with 4D-GS (Wu et al., 2024). It is visible that the given random initialization of the deformation models degenerates the initially well-structured Gaussians in (a).*

to the collapse of the deformation model during reconstruction. To mitigate this issue, the initialization is adapted by constraining the magnitude of the final-layer weights rather than forcing them to zero. In practice, this is done by lowering the gain of the Xavier initialization from $gain = 1.0$ to $gain = 0.5$, which limits the maximum value of motion model weights at initialization, effectively limiting the initial deformation strength. At the same time, this ensures sufficient gradient flow for the optimization process.

Rasterizer Configuration Adjustments

One of the most important modifications concerns the rasterizer settings. While the default configuration is sufficient for wide scenes with limited depth ranges, such as those in the Nerfies dataset, it is inadequate for datasets with large depth variation, like the iPhone wheel scene, with distances ranging from approximately 0.05 to 20 meters. The limitation arises from the minimum distance requirements of the Gaussian centers to the camera, as defined in the rasterizer. Specifically, a Gaussian is rendered only if its center lies within the camera frustum, which, together with other constraints, requires a minimum distance of 0.2 meters from the camera. To address this, the minimum value is adjusted following *GSplat* (Ye et al., 2025), which uses 0.01 meters as the minimum. This modification ensures greater flexibility for the model while maintaining stable optimization.

4.2.2 Depth Supervision Mechanism

Several steps are required to enable the use of depth maps for supervising the training of dynamic 3D reconstruction methods. Once the depth maps are integrated into the pipeline, it is crucial that the model's reconstructed depths are rendered in a differentiable manner, allowing gradients to propagate during training, as detailed in Section 4.2.2. Next, the depth data need to be preprocessed, as described in Section 4.2.2, to ensure numerical stability. Finally, an appropriate loss formulation and weighting strategy need to be chosen, as discussed in Section 4.2.2.

Differentiable Rasterization Pipeline

The rasterizer used in *MonoDyBench* (Liang et al., 2025) does not natively support differentiable depth rendering. To maintain reproducibility and consistency with the original framework, a workaround is implemented in which the RGB channel of the rasterizer is repurposed to render the depth. For this, the distances of the Gaussians to the camera plane are computed by transforming the Gaussians from world space into camera space. To ensure the rendered depth values are numerically identical to those produced by the original (non-differentiable) depth renderer, the corresponding CUDA implementation is directly translated into Python. Then, inside the rasterizer, the resulting depth values are obtained through the regular point-based rendering procedure described in Section 2.2.3. It is important to note that the background color,

corresponding to pixels where no Gaussians are projected, results in zeros. The motivation for this design is further explained in Section 4.2.2.

Inverse Depth Representation

Directly rendering the inverse depth offers two main advantages. First, since the objective is to increase reconstruction quality primarily in the dynamic foreground regions of the scenes, one way to ensure this is to use the inverse depth for supervision, as done by Kerbl et al., 2023. This has the advantage that, when using the simple $\mathcal{L}_1$ loss, errors for pixels closer to the camera are larger than those for pixels further away.

Second, using inverse depth elegantly encodes "infinite" distance for empty regions without projected Gaussians. In standard depth rendering, the rasterizer's background values would need to be set to infinity. However, this would lead to erroneous values when applying alpha blending, forcing unintuitive zero-value background initialization. Inverse depth rendering avoids this problem directly, as a zero-valued pixel naturally corresponds to infinite distance.

Depth Loss Formulation

Following the approach proposed in *AD-GS: Object-Aware B-Spline Gaussian Splatting for Self-Supervised Autonomous Driving* Jiawei et al., 2025, the *scale-and-shift depth loss* introduced by *MonoSDF: Exploring monocular geometric cues for neural implicit surface reconstruction* Yu et al., 2022 is adopted. This formulation applies simple $\mathcal{L}_1$ loss for depth supervision but introduces an important modification: the loss is not computed directly on the raw (inverse) depth values. Instead, the predicted depth is first aligned to the ground truth by estimating a scale and shift pair (s, t) via a least squares approach with a closed-form solution.

However, during experimentation, this alignment procedure caused instabilities within the 3D reconstruction framework. At the early stages of training, the least squares alignment frequently estimated negative scale values. The issue arises because, at initialization, the rendered depth primarily consists of the sparse Structure-from-Motion (SfM) points, which are aligned to the ground truth. In this setting, the non-inverted depth can sometimes minimize the least-squares error more effectively, due to the presence of large Gaussian-less regions, which leads to degenerate negative-scale behavior. As a consequence, the model exploits the sign inversion, effectively learning to reconstruct inverse depths rather than true depths. This occurrence is illustrated in Figure 4.4.

To avoid this, the scale parameter is constrained to be positive. Two strategies were investigated: first, zero-clipping, and second, applying the absolute value function. Both approaches improve stability and reconstruction quality relative to the baseline. Empirically, however, using the absolute value function yields better results than zero

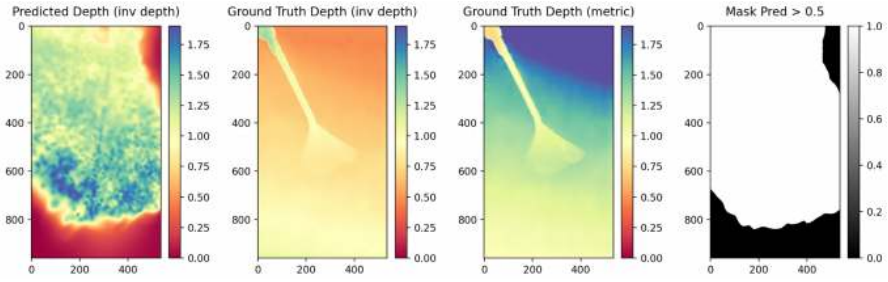


Figure 4.4: *Visualization of the initial training step on the Nerfies broom scene showing the predicted inverse depth immediately after the SfM initialization and before the first optimization step. The alignment procedure from MonoSDF (Yu et al., 2022) estimates a scale of $s = -0.0840$ and a shift of $t = 0.8454$. When regions without placed Gaussians are masked out, here, pixels with predicted inverse depths larger than 0.5 (right), the estimated parameters become $s = 0.2420$ and $t = 0.4145$. This demonstrates that empty regions can mislead the Least Squares solution, resulting in negative scale estimates.*

clipping. This leads to the following final loss function

$$\mathcal{L}_{\text{depth}} = \left\| \left(|s| \cdot \frac{1}{\hat{d}} + t \right) - \frac{1}{d} \right\|_1 . \tag{4.8}$$

For the depth loss, two strategies were also evaluated, where the first uses a constant weight $w = 0.01$. The second follows an update in the *3DGS* codebase (Kerbl et al., 2023) and uses a decaying weight that enforces strong geometric supervision at the start of optimization but gradually degrades towards the end. Experiments on the *Nerfies* dataset indicated that this decaying schedule outperforms the constant weight approach. Additionally, while the constant weighting is prone to over- or under-regularization across scenes of varying scales when the weighting factor is not set appropriately, the decaying schedule reduces this sensitivity. By applying strong geometric supervision at the beginning of the training and gradually relaxing it over time, the model learns a good initial geometry whose appearance can then be refined more freely by the existing RGB loss, which is more multi-view consistent. Figure 4.5 illustrates the tested depth-weight schedules. Both decaying schedules start from an initial weight of 1.0 and decrease to a final weight of $w\_final = 0.001$, in contrast to the higher final weight of $w\_final = 0.01$ used in *3DGS* (Kerbl et al., 2023). The figure also highlights the lower slope value of the proposed schedule compared to the original one from *3DGS*, giving the models earlier freedom in appearance optimization.

Out-of-Memory Problems (OOM)

During experimentation, a recurring problem was that the *DeformableGS* (Ziyi Yang et al., 2024) model occasionally tried to reserve tens of gigabytes of GPU memory for a single rasterizer call, resulting in many *Out-of-Memory* (*OOM*) crashes. This problem was not only introduced by the added depth supervision, as it also occurred in the baseline experiments, but was certainly aggravated by the double rasterizer call required for rendering both RGB and depth in a differentiable manner. Further investigation showed that OOM errors consistently originated from the same code location,

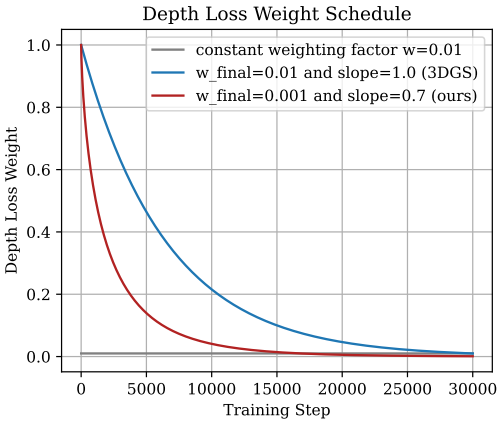


Figure 4.5: *Comparison of different depth-weight schedules. The traditional decaying schedule from 3DGS (Kerbl et al., 2023) is shown in blue, alongside a constant weighting scheme in grey. The proposed schedule features a stronger decay (lower slope value) and a smaller final weight value* w_final.

specifically, immediately before one of the two rasterizer calls. Throughout training, OOM errors frequently occurred in two distinct training phases:

The first occurrence was directly after the warm-up phase (at 3,000 of 30,000 iterations). In this stage, the deformation model is first activated after initially training only the static scene representation. Consequently, all parameters of the deformation model are loaded and used for computation on the GPU for the first time.

The second incident of OOM errors occurred after the densification phase (between iterations 15,000 and 30,000). Similar to the procedure used in *3D Gaussian Splatting* (Kerbl et al., 2023), after densification, the model does not clone, split, or prune any Gaussians. Importantly, during this phase, the model clears the CUDA cache after every densification step, which is far more often than in later stages, where cache clearing is performed only once per test step (every 1000 iterations).

As a result, three main strategies are implemented to reduce these OOM issues, ordered by increasing effectiveness:

1. Expandable CUDA memory segments The first improvement involves using PyTorch's environment variable configuration to allow expandable CUDA segments by setting `os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"`.[1] This allows the CUDA allocator to dynamically expand the initially created segment according to the required GPU memory.

2. Periodic CUDA cache clearing After the warm-up phase, the command `torch.cuda.empty_cache()` is added before and after each rasterizer call for ten consecutive iterations (see Table 4.1). Additionally, after the densification phase, this command is applied every 200 iterations to proactively release unused GPU memory.

[1] https://docs.pytorch.org/docs/stable/notes/cuda.html, last accessed on November 5, 2025.

Table 4.1: *GPU Memory.* *Exemplary history of GPU memory allocation and reservation during a single iteration using the proposed periodic CUDA cache clearing after the warm-up stage. After clearing the amount of reserved memory drastically drops from around 22 GB (Cleared Cache ? - No) to 9.74 GB (Cleared Cache ? - Yes)*

Stage	Cleared Cache ?	Allocated (GB)	Reserved (GB)
Before RGB rasterizer	No	9.58	21.97
	Yes	9.58	9.74
After RGB / Before Depth rasterizer	No	9.62	22.40
	Yes	9.62	9.74
After Depth rasterizer	No	9.63	22.40
	Yes	9.63	9.74

3. Checkpointed rasterization The third optimization is using PyTorch's checkpointing functionality via the `torch.utils.checkpoint.checkpoint()` function.[2] Normally, during training, the rasterizer is called in the forward pass, and the entire computation graph is stored in memory until the backward pass for gradient computation. Checkpointing, on the contrary, reduces memory consumption by discarding intermediate tensors and recomputing them on demand during the backward pass.

As a result, every iteration proceeds as follows: each rasterizer renders the current 3D reconstruction (one for RGB, the other for depth) and immediately releases its memory. Then, when gradients are required during backpropagation, the forward pass is re-executed for each rasterizer. Of course, this approach comes with some computational overhead. However, we observed its impact on training time to be negligible in this framework. Even if training were slightly slower, obtaining a complete reconstruction clearly outweighs the risk and cost of potential OOM crashes.

4.2.3 Depth-based Initialization Procedure

The work of *3D Gaussian Splatting* (Kerbl et al., 2023) showed that initializing Gaussians with Structure-from-Motion (SfM) points significantly improves the reconstruction quality compared to random initialization. This approach not only leads to finer details but also reduces the number of floaters in the scene that otherwise are difficult to remove during optimization. However, as *3DGS* was designed for static scenes, the SfM points typically cover most of the areas in the scenes, leading to an overall improvement. For the case of dynamic scenes, on the contrary, those dynamic foreground points are not captured by SfM. For this reason, in the context of *dynamic* scene reconstruction, we explored depth-based initialization for the dynamic foreground.

The following sections will cover several experimental extensions: *Alignment Methods* in Section 4.2.3, a *Progressive Dataset Scheduling* strategy in Section 4.2.3, training *Without Warm-Up Phase* in Section 4.2.3, and finally, various *Initialization Refinements* in Section 4.2.3.

[2] https://docs.pytorch.org/docs/stable/checkpoint.html, last accessed on November 5, 2025.

Alignment Methods

To integrate the foreground depth map points as 3D Gaussians into the scene alongside the existing Structure-from-Motion (SfM) points, two strategies are investigated to ensure spatial consistency between both sources.

The decision to add only foreground points comes down to the observation that the background is already well represented by the SfM points, which are typically highly accurate. Moreover, the models generally have little difficulty reconstructing the static background regions of a scene. Adding background Gaussians derived from the depth maps would therefore introduce unnecessary noise into areas that already have well-placed Gaussians. Furthermore, since the alignment procedure itself can be unstable, including background depth points would propagate these instabilities and presumably not lead to a better reconstruction.

3D Alignment. The first approach involves projecting the SfM points onto the image plane, allowing a rough 2D alignment of the depth map using either RANSAC or Least Squares. To do so, all depth map points without a corresponding SfM point are first excluded. Additionally, the dynamic foreground regions are masked out using the provided dataset masks. As a result, only background SfM points that are visible and not occluded by moving foreground objects are for alignment. After this 2D alignment, the depth map is unprojected into 3D space, followed by an alignment in 3D using the Iterative Closest Point (ICP) algorithm on the same set of non-occluded background SfM points. The ICP algorithm is a well-established method for aligning two meshes or point clouds. For each point, it iteratively selects the closest corresponding point in the other set, computes the distances, and optimizes the translation and rotation parameters until a predefined distance threshold is achieved.

However, this approach can become unstable when only a limited number of SfM points are available, either because large parts of the background are occluded by the dynamic foreground or due to incomplete SfM reconstruction. In such cases, the 3D alignment technique struggles due to its high degree of freedom, allowing points to be passed through arbitrary translations, rotations, and even scalings. Even though this can occasionally lead to good results, it may also lead to severe misalignments or even complete degeneration of the foreground depth points, as can be seen in the Curls scene of the Nerfies dataset in Figure 4.6.

2D Alignment. To reduce this risk, all subsequent experiments in Chapter 5 omit the 3D alignment completely. Instead, depth maps are aligned only in 2D using RANSAC, with Least Squares serving as a fallback option when RANSAC does not achieve the desired threshold. While this simplification sacrifices the potential higher accuracy of full 3D alignment, it substantially improves robustness and consistency across scenes and datasets.

(a) (b)

Figure 4.6: *Visualization of the ground truth RGB image (left), alongside the initialized Gaussians using (a) the 3D alignment via ICP and (b) 2D alignment only. It is clearly visible that in the 3D-aligned version, the foreground Gaussians deviate significantly from the true image, while the 2D alignment preserves a more accurate correspondence with the ground truth.*

Progressive Dataset Scheduling

In standard training setups for reconstruction techniques like *DeformableGS* (Ziyi Yang et al., 2024 or *EffGS* Katsumata et al., 2024, and machine learning models in general, images are sampled randomly from the full training set at each iteration. However, when the foreground Gaussians are initialized using depth maps, this can lead to the following unfavorable scenario: although the initialization is performed using the first training image, the initial randomly sampled images may have a large baseline or correspond to a later time step where the foreground geometry is completely different compared to the initialization frame. In this case, the introduced initialization procedure could even harm reconstruction quality as the network may initially try to remove or overwrite the existing, valid Gaussians.

To address this issue, we tested a *Progressive Dataset Scheduling* strategy. Instead of sampling from the entire training dataset, the model starts training using only the initialization image. Then, every k iterations, the number of exposed training images increases linearly by adding the next training image. Assuming the dataset is temporally ordered and the set from which the model is sampling is always the first n images, it progressively explores the scene, allowing it to make full use of the initialized Gaussians.

Without Warm-Up Phase

Another important setting for maximizing the benefits of depth-based initialization is to train the model *Without a Warm-up Phase*. In this context, the warm-up phase refers to the initial training period, up to iteration 3,000 in this case, during which the deformation model is not yet applied. This stage is typically used to optimize the background Gaussians, ensuring that the static parts of the scene are already well reconstructed before the motion model is activated.

However, large foreground motion occurring during this stage of training may actually harm the initialization, as previously fine details are degenerated by blending and misaligning Gaussians across inconsistent viewpoints. Therefore, when having

a reasonably accurate Gaussian initialization, skipping the warm-up phase allows for joint optimization of Gaussian placement and motion from step one, resulting in more detailed and robust reconstructions.

Initialization Refinements

Once the spatial location of the Gaussian primitives is set, two additional properties need to be initialized: their *scale* and *color*.

Color. After initialization, both the newly added foreground Gaussians as well as the SfM-based Gaussians are projected onto the image plane of the first training view, whose corresponding RGB image is also loaded. Assuming that the foreground points are always located in front of the SfM points, each foreground Gaussian directly receives its color from the corresponding pixel in the image.

For the SfM Gaussians, however, color assignment is only possible under two conditions: the Gaussian must lie within the camera frustum, and it must not be occluded by a foreground point. Points outside the frustum or those occluded by the foreground keep the standard random initialization.

Scale. For the scale initialization, SfM and foreground Gaussians are treated differently. SfM Gaussians are initialized using the same approach as introduced in *3D Gaussian Splatting* by Kerbl et al., 2023. Specifically, on the GPU, the distance to the nearest neighboring Gaussian is computed and used as the scale value. This ensures that Gaussians in densely populated regions receive smaller scales while those in sparse areas are larger, resulting in a closed and continuous background without gaps or excessive blurring.

For the foreground Gaussians, the fact that the points are added in full resolution can be exploited. Consequently, the goal is to assign each Gaussian a scale corresponding to the size of a single pixel when projected onto the image plane. We follow a dedicated function introduced in *ConeGS* (Baranowski et al., 2026) that determines the specific scale of a Gaussian based on its 3D position in camera-space, its depth value, and the inverse camera intrinsics matrix.

5 Evaluation

Depth prediction plays an important role in the proposed reconstruction pipeline as the quality of depth priors directly influence the depth-based supervision as well as the initialization of Gaussian primitives during training. Therefore it is essential to have a reliable quantitative assessment of the quality of the different depth estimation methods. This evaluation is presented in Section 5.1. Following this, Section 5.2 and Section 5.3 elaborate on the main results of the extended evaluation framework introduced by Liang et al., 2025 regarding the performance changes by introducing depth supervision and depth-based initialization of Gaussians, respectively.

All evaluations in the following sections are conducted on the iPhone dataset, specifically the scenes *apple*, *backpack*, *creeper*, *handwavy*, *haru-sit*, *mochi-high-five*, *paper-windmill*, *pillow*, *spin*, *sriracha-tree*, and *teddy*. For the experiments involving the integration of depth priors in Section 5.2 and Section 5.3, additional results are reported for the *broom*, *curls*, *tail*, and *toby-sit* scenes from the Nerfies dataset.

Given that modern dynamic 3D reconstruction methods struggle particularly with reconstructing dynamic foreground regions, the following chapter places special emphasis on evaluating both depth prediction methods and reconstruction quality specifically within these dynamic foreground areas.

5.1 Evaluation of Depth Prediction Models

This section presents a systematic comparison of several monocular depth prediction models, evaluating their accuracy on scenes of the iPhone dataset. An overview of all individual models used for comparison is given in Section 2.3. The quantitative results are then summarized in Section 5.1.2, highlighting the main observations across different alignment techniques introduced in Section 4.1.1. These include a per-image median-based alignment, a scene-level variant using the average scale and shift value, and a RANSAC-based alignment whose parameters are estimated per scene. All methods are assessed using the quantitative metrics MSE, AbsRel, and δ_1, as defined in Section 4.1.2, considering accuracy evaluation on both full image and dynamic foreground

regions.

5.1.1 Listing of all Depth Estimation Methods

Throughout this section, the notation *a + b* denotes *method a aligned to method b* using the per-image median-based scale-shift alignment technique described in Section 4.1.1.

Depth Anything V2 + Colmap Inspired by *Shape of Motion* Q. Wang et al., 2025, this combination uses COLMAP depth as a reference to convert the relative predictions of Depth Anything V2 into metric scale. As a consequence, this approach is only applicable to datasets where COLMAP reconstructions are available.

Depth Anything V2 + UniDepthV1 Also motivated by *Shape of Motion* Q. Wang et al., 2025, this variant is applied in scenarios where COLMAP data cannot be obtained, as is the case for in-the-wild scenes. Here, relative depth predictions from Depth Anything V2 are aligned to the metric scale of UniDepthV1.

Depth Pro A direct metric depth prediction method, chosen due to its fine details and reportedly strong performance on general monocular depth estimation datasets.

MoGe A standard direct metric depth prediction method, included as another state-of-the-art alternative for comparison.

MegaSaM (1) The original MegaSaM configuration for metric depth prediction, using Depth Anything V1 aligned to UniDepthV2 as its depth prior and COLMAP focal lengths. This reflects the authors' intended setup for depth estimation and serves as the primary reference configuration.

MegaSaM (2) An adapted variant of MegaSaM that replaces the original depth prior with metric Depth Pro while keeping COLMAP for focal length estimation. The aim is to test whether the finer detail of Depth Pro can improve MegaSaM's predictions.

MegaSaM (3) Another original MegaSaM configuration enabling processing of in-the-wild scenes. It uses Depth Anything V1, aligned to UniDepthV2, as the depth prior, but relies on UniDepthV2 focal length estimations instead of those from COLMAP.

UniDepthV2 A standard direct metric depth prediction method, included due to its widespread adoption in recent reconstruction and SLAM pipelines such as *Shape of Motion* and *MegaSaM*, respectively.

Video Depth Anything A direct metric depth prediction method, selected because of its strength in producing temporally consistent depth across video sequences.

Table 5.1: ***Summary of Quantitative Results for the depth-prediction benchmark.*** *The table reports the per-method median errors across the entire iPhone dataset for the three evaluation metrics (MSE, AbsRel, and δ_1) under two alignment settings: per-image median-based alignment and the scene-level RANSAC alignment. Each metric is shown separately for the dynamic foreground (°) and the full image (•), with the* ***best*** *and second-best methods highlighted.*

Method\Metric	Per-Image Alignment						RANSAC Alignment					
	MSE° ↓	MSE• ↓	AbsRel° ↓	AbsRel• ↓	δ_1° ↑	δ_1• ↑	MSE° ↓	MSE• ↓	AbsRel° ↓	AbsRel• ↓	δ_1° ↑	δ_1• ↑
Depth Anything V2 + Colmap	0.0023	0.0052	0.0581	0.0496	0.9923	0.9859	0.0026	0.0057	0.0691	0.0593	**0.9921**	0.9826
Depth Anything V2 + UniDepthV1	0.0045	0.0081	0.0983	0.0761	0.9614	0.9337	0.0086	0.0148	0.1450	0.1225	0.8671	0.8800
Depth Pro	0.0021	0.0077	0.0568	0.0576	0.9932	0.9825	0.0035	0.0149	0.0886	0.0924	0.9851	0.9575
MoGe	0.0028	0.0040	0.0672	0.0500	0.9907	0.9861	0.0056	0.0094	0.1130	0.0934	0.9649	0.9668
MegaSaM (1)	0.0022	**0.0036**	**0.0540**	0.0454	0.9930	**0.9890**	**0.0024**	**0.0040**	0.0688	0.0524	0.9907	0.9875
MegaSaM (2)	0.0023	0.0074	0.0548	0.0485	0.9923	0.9829	0.0038	0.0235	0.0982	0.1369	0.9764	0.8568
MegaSaM (3)	0.0021	0.0037	0.0542	0.0453	0.9930	**0.9890**	**0.0024**	**0.0040**	**0.0668**	**0.0518**	0.9919	**0.9901**
UniDepthV2	**0.0019**	0.0037	0.0566	**0.0444**	0.9930	0.9885	0.0037	0.0083	0.1007	0.0838	0.9819	0.9728
Video Depth Anything	0.0037	0.0075	0.0828	0.0711	0.9777	0.9684	0.0051	0.0107	0.1054	0.0952	0.9544	0.9561
Video Depth Anything + UniDepthV2	0.0029	0.0270	0.0665	0.0826	0.9885	0.9476	0.0045	0.0197	0.0976	0.0983	0.9825	0.9378
Depth Anything V2 + UniDepthV2	0.0023	0.0044	0.0578	0.0505	**0.9936**	0.9851	0.0041	0.0088	0.0933	0.0906	0.9797	0.9656
Depth Pro + Video Depth Anything	0.0021	0.0077	0.0568	0.0576	0.9932	0.9825	0.0035	0.0123	0.0827	0.0875	0.9898	0.9755

Video Depth Anything + UniDepthV2 A hybrid approach where metric Video Depth Anything predictions are aligned to UniDepthV2, reflecting the common use of UniDepthV2 as a scale reference.

Depth Anything V2 + UniDepthV2 A combination aligning updated metric Depth Anything V2 to UniDepthV2, used to analyze the performance of depth priors that also serve as input to MegaSaM.

Depth Pro + Video Depth Anything A complementary composition in which metric Depth Pro is aligned to Video Depth Anything to combine the fine details of Depth Pro with the video consistency of Video Depth Anything.

5.1.2 Key Observations

This section presents the final results of the depth map comparison pipeline and highlights the key insights derived from the evaluation. Table 5.1 reports all median errors in non-processed form. In addition to the figures below, additional plots are provided in the supplementary (see Appendix A).

No Influence of Alignment Technique when Using a Single Scale-Shift Estimate per Scene

When a single scale and shift value is applied to an entire scene, the choice of alignment strategy has virtually no effect beyond a global rescaling. Regardless of whether these parameters are obtained by averaging the per-image median-based scale and shift values or by estimating them once via RANSAC, the relative ordering of the methods in terms of their error remains almost unchanged. This behavior is visible in Figure 5.1 and motivates focusing exclusively on the RANSAC-based results in the following discussion.

One Model Consistently Ranks Among the Top Performers

Across nearly all metrics, regions, and alignment settings, the original *MegaSaM (1)* variant is almost always ranking among the strongest methods. And in the few cases where it is not the best performer, its margin to the leading method remains small.

Composed Priors Are Not Necessarily Better Than Single Models

Composed approaches, such as *Video Depth Anything + UnidepthV2* or *Depth Pro + Video Depth Anything*, can sometimes offer improvements in isolated scenarios, for example, in the foreground AbsRel evaluation, as shown in both per-scene alignment cases in Figure 5.1. However, the overall picture is mixed: depending on the metric and alignment strategy, composed depth methods may outperform, match, or underperform their individual counterparts.

MegaSaM Is Sensitive To Its Depth Prior but Robust to Focal-Length Priors

Especially for whole-image evaluation, substituting MegaSaM's internal depth prior with Depth Pro (*MegaSaM (2)*) leads to a significant drop in accuracy, making this variant the weakest performer in both full image single scale-shift alignment cases, as can be seen in Figure 5.1. Even in other settings, it consistently performs worse than the baseline *MegaSaM (1)* and the in-the-wild version *MegaSaM (3)*.

On the other hand, exchanging the focal-length estimation prior of COLMAP (*MegaSaM (1)*) to *UniDepthV2* (*MegaSaM (3)*) causes only a slight decrease in performance.

Foreground Accuracy Differs Substantially from Full-Frame Accuracy

As expected, MSE increases notably for whole-image evaluation due to the larger depth ranges in background regions, resulting in higher absolute error values. However, it is interesting to see that this behavior does not generally transfer to the scale-invariant metrics AbsRel and δ_1, which remain more stable across regions, validating their use in this setup. Two exceptions exist: *Depth Pro* and *MegaSaM (2)*, both show improved AbsRel and δ_1 performance when restricted to the foreground. Using these metrics, it becomes evident that both methods excel particularly in estimating depth in the foreground regions.

Video-Consistent Models Have Higher Temporal Consistency

The primary reason for introducing scene-level alignment strategies (scene-averaged median scale-shift and RANSAC) was to reveal temporal consistency by penalizing per-frame fluctuations. Comparing *Video Depth Anything* in the per-image alignment setup on the foreground in Figure 5.2 to the RANSAC aligned case in Figure 5.1 shows only a moderate increase in AbsRel from 0.0828 to 0.1054. At the same time, single-image predictors degrade far more strongly, for instance *UniDepthV2* drops from 0.0566 to 0.1007. The same holds true for *MegaSaM*, whose error increases by only 0.0148,

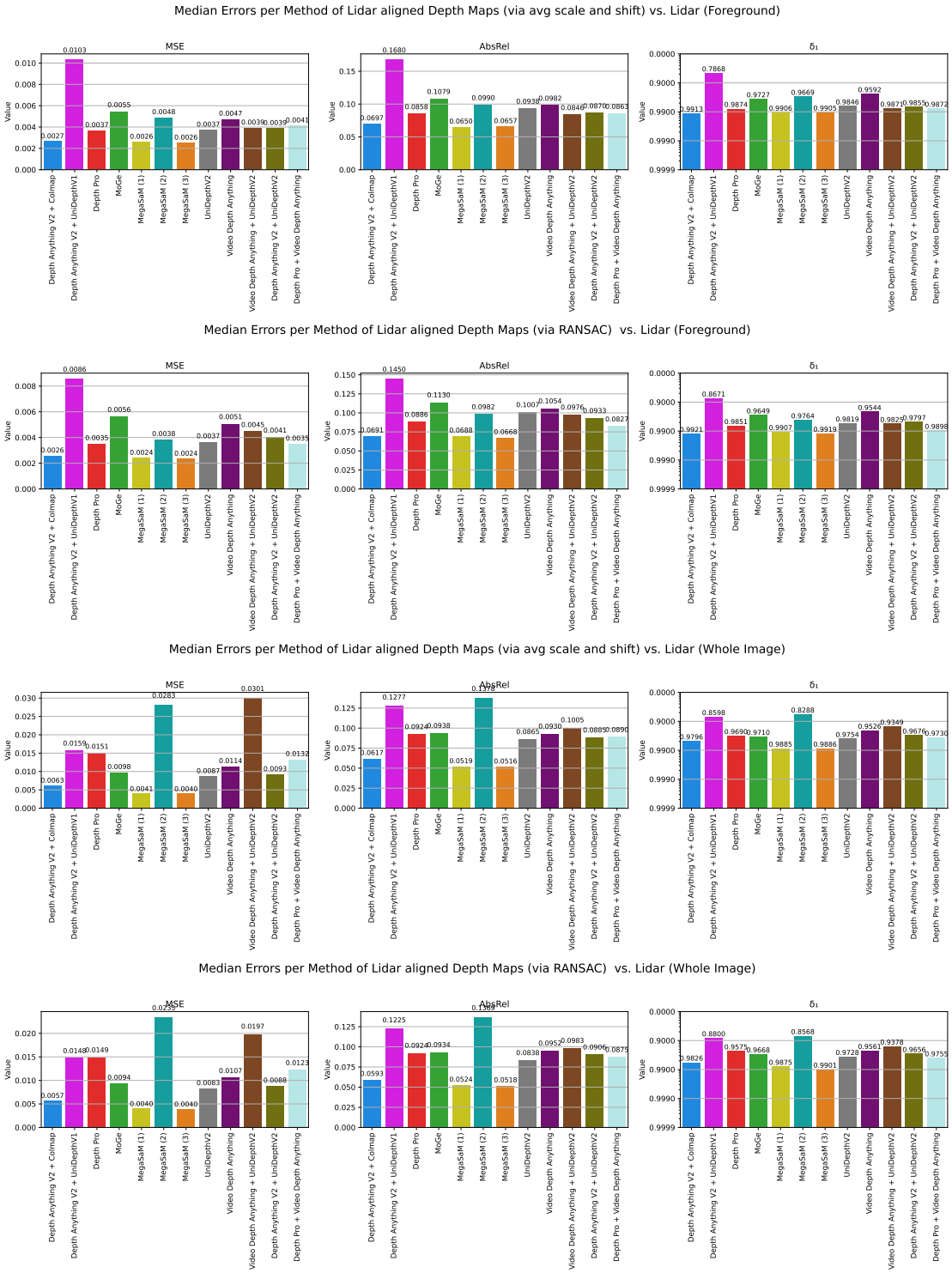


Figure 5.1: *Visualization of median errors for all depth prediction methods across the three alignment strategies, shown separately for foreground and whole-image evaluations. Foreground results are displayed for the scene-averaged median-based alignment (top) and the RANSAC alignment (upper middle), while whole-image results are shown in the corresponding plots below (lower middle and bottom). Note that the y-axis for δ_1 is inverted, so across all metrics the same rule applies: lower bars indicate better performance. It can be seen that both per-scene alignment strategies produce similar results in terms of the relative ranking of the methods.*

whereas single-frame methods worsen by roughly 0.03. This behaviour supports the claims that video-consistent models produce more temporally consistent predictions than single-frame ones.

Poor Alignment Targets Can Substantially Harm Combination Methods

This issue particularly affects *Depth Anything V2 + UniDepthV1* and *Depth Anything V2 + UniDepthV2* across all alignment types, metrics, and regions. When the metric depth used as an alignment target is itself inaccurate, the resulting combined prediction degrades accordingly. In the same way, when evaluating whole images, aligning a relative predictor, such as *Depth Anything V2*, to a metric depth model like *UniDepthV2* can negatively impact the latter's performance.

Single-Image Methods Do Not Necessarily Exhibit Higher Error Variability

Contrary to the expectation that single-frame predictors should produce depth maps with higher variation due to the absence of temporal cues, this is not reflected in the evaluation. To support this claim, Figure 5.3 shows the Median Absolute Deviation of Errors across the different methods. *Video Depth Anything* actually exhibits a higher median absolute deviation of 0.0201 compared to several single-image models, such as *Depth Pro* (0.0063) and *UniDepthV2* (0.0065), on the foreground, with similar trends in whole-image results.

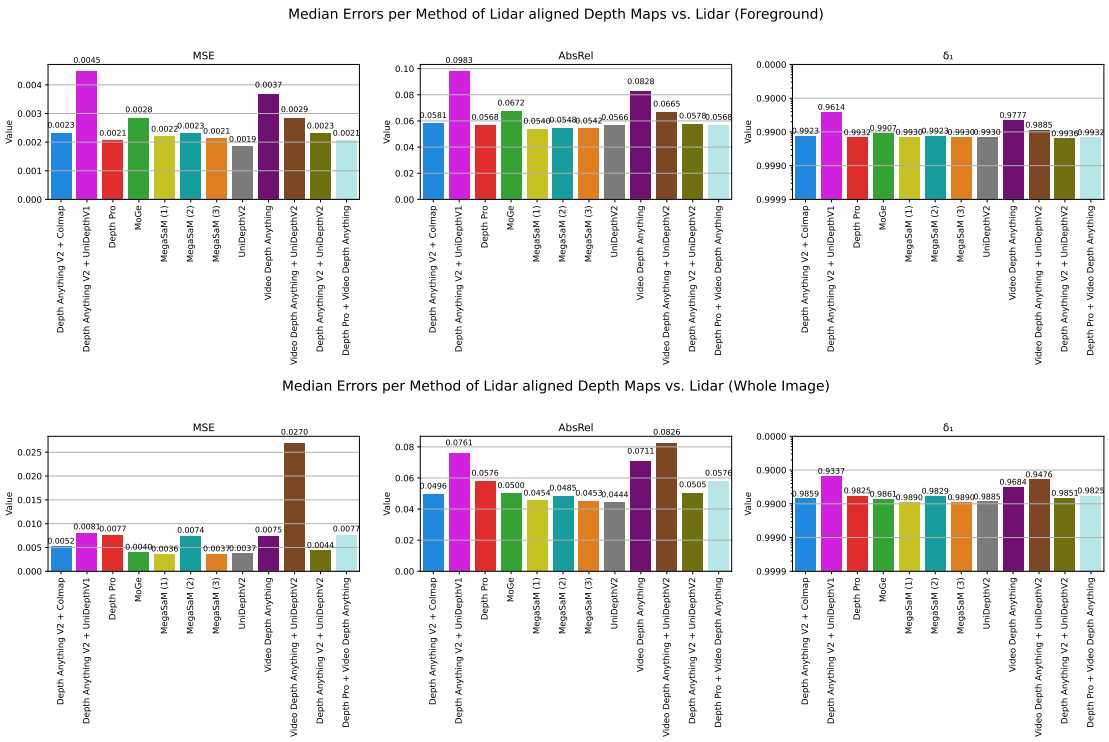


Figure 5.2: *Per method median error using the per-image median-based alignment shown for the foreground (top) and for the whole image (bottom). It is important to note that, especially for MSE, the accuracy on the foreground differs substantially from full-frame accuracy.*

5.1.3 Limitations of the Comparison Pipeline

When comparing the qualitative depth maps of the *sriracha-tree* scene from the iPhone dataset (see Figure 5.4) to the corresponding quantitative results, an important limitation can be formulated: the evaluation metrics used in this comparison do not adequately capture the quality of fine geometric structures in depth maps. Two main factors contribute to this shortcoming.

First, none of the proposed metrics explicitly focuses on edge sharpness or the fidelity of high-frequency geometric details. One possibility would be to use a *gradient matching* error, as previously introduced in *MiDaS* Ranftl et al., 2022 and also employed in *Depth Anything V2* L. Yang et al., 2024a. Such metrics, however, are only beneficial when dense and accurate ground truth data is available, which is typically limited to synthetic datasets.

This connects directly to the second limitation: the inherent sparsity of the LiDAR measurements. As illustrated in Figure 5.4d, the LiDAR sensor captures depth accurately for surfaces whose surface normals are oriented roughly towards the device but fails in regions with slanted surface orientations or thin structures such as ropes or narrow edges. These areas often contain exactly the fine details that would be valuable to evaluate, yet they remain unmeasured because LiDAR cannot return valid depth for them. As a consequence, substantial portions of visually important geometry lack ground truth supervision, limiting the ability of any quantitative metric to accurately

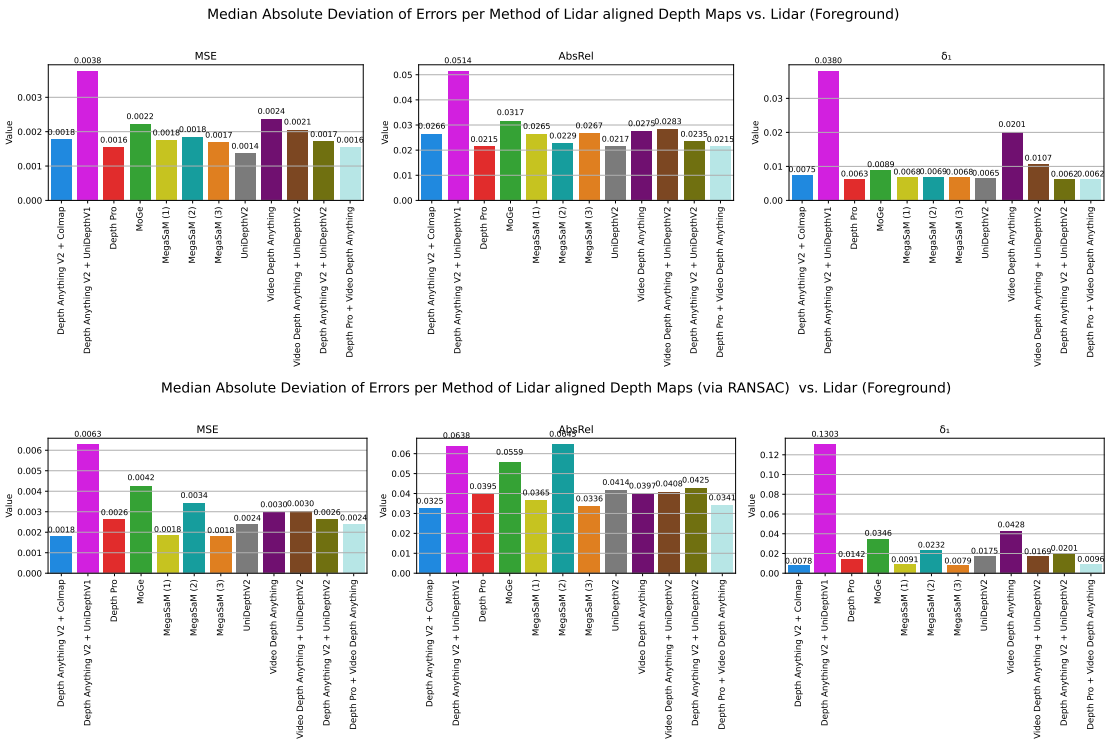


Figure 5.3: *Median absolute deviation of per-method errors on the foreground, shown for the per-image median-based alignment (top) and the RANSAC-based alignment (bottom). The results show that non-temporal depth prediction models do not necessarily have higher variability in median error.*

capture errors in those areas.

5.1.4 Implications for the Integration of Depth Priors

In recent work on neural scene representations, the reconstruction of static environments can be considered as very sophisticated, while the main challenge now lies in accurately recovering the geometry of dynamic foreground regions. This is the reason why the selection of suitable depth priors should primarily be based on their performance on the foreground. Based on the findings from the quantitative analysis, a well-rounded selection of three depth-prediction models has been made for further investigation in the following section:

Depth Pro is selected for its strong overall accuracy, high visual fidelity (see Figure 5.4a), and fast inference time of roughly one second per image. It also performs exceptionally well on the dynamic foreground, ranking as the best single-frame model

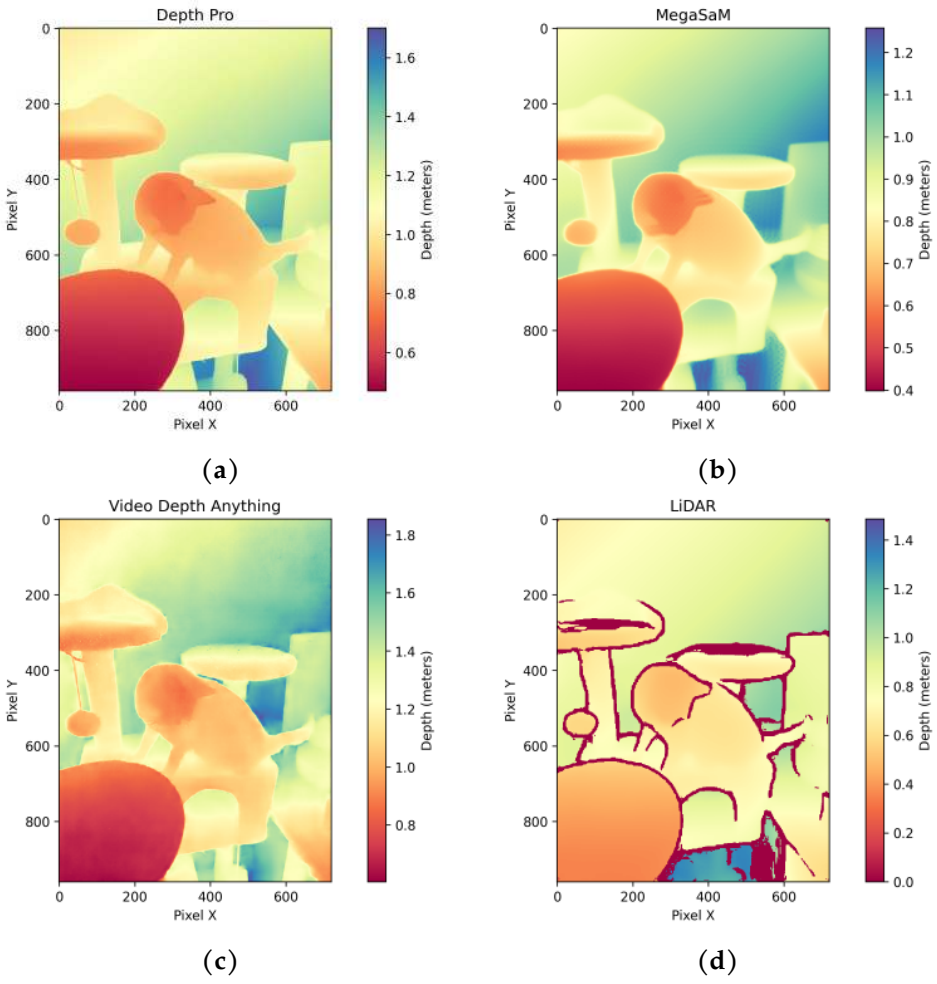


Figure 5.4: *Qualitative comparison of different depth prediction methods: (a) Depth Pro captures fine geometric details, (b) MegaSaM shows occasional dot-like artifacts, (c) Video Depth Anything produces less smooth surfaces at lower resolution, and (d) the ground truth LiDAR, showing its characteristic missing values along edges and fine structures.*

in both AbsRel and MSE under the RANSAC alignment (see Figure 5.1) and as the second-best under per-image alignment (see Figure 5.2).

MegaSaM is chosen because it nearly always achieves the strongest performance across all metrics, alignment strategies, and image regions. These benefits come at the cost of notable drawbacks: its inference pipeline is rather slow, requiring full-depth predictions from *Depth Anything V1* and *UniDepthV2*, followed by a SLAM-based bundle adjustment step. The latter is the reason for its substantial memory consumption, resulting in frequent out-of-memory issues when processing long sequences. As a consequence, the influence of MegaSaM as a depth prior is only evaluated on the iPhone dataset, where it serves as the upper-bound depth prediction reference.

Video Depth Anything is finally included as a lightweight alternative to the latter, providing temporal consistency across frames and making it attractive for video settings, despite its weaker accuracy compared to the two methods above.

Based on the preceding evaluations, for the discussions in both of the following sections, these three depth prediction methods can be ordered by increasing accuracy as follows: *Video Depth Anything* < *Depth Pro* < *MegaSaM*.

5.2 Depth-Based Supervision

This section summarizes the key findings of incorporating depth-based supervision into dynamic 3D reconstruction. The analysis covers all methods and datasets, discussing both consistent trends and method-specific properties. A representative failure case is highlighted in Section 5.2.2.

5.2.1 Key Observations

Table 5.2 reports all quantitative results averaged over the iPhone and the Nerfies dataset. In addition to the figures below, further plots are provided in the supplementary material (see Appendix B).

Influence of Depth-Based Supervision Varies Across Methods and Datasets

The main observation of this section is that the impact of depth-based supervision on dynamic 3D reconstruction pipelines is far from uniform. Its effectiveness depends strongly on both the underlying motion model and the dataset characteristics. As can be seen in Table 5.2, integrating depth priors can lead to improvements, negligible changes, or even significant degradation.

The largest negative effect appears in *DeformableGS*. Here, across all metrics, the performance drops when depth supervision is enabled. While the baseline achieves an

average PSNR of 19.52 across the iPhone and Nerfies datasets, incorporating depth priors reduces it to 19.36 with *Video Depth Anything* and 19.42 with *Depth Pro*. However, a closer look at Figure 5.5 reveals that this decline mainly stems from the iPhone dataset. On Nerfies, *DeformableGS* occasionally shows improvements on selected metrics, while on iPhone, the degradation is consistent.

For *4D-GS*, the effects are mixed. On average, and for both datasets individually, it benefits from the depth-supervised reconstruction pipeline in terms of PSNR and LPIPS. In contrast, improvements in SSIM and MS-SSIM are small and vary with the choice of depth prediction model.

EffGS is the most reliable winner of depth-based supervision. Independent of the dataset, metric, or depth prediction model, its performance consistently increases when depth priors are incorporated into training.

Foreground Regions Particularly Benefit from Depth Supervision

As dynamic 3D reconstruction methods typically struggle the most with accurately modeling the dynamic foreground, assessing the performance of this region is of particular importance. As shown in Table 5.2 and Table 5.3, introducing depth supervision boosts reconstruction quality in the foreground to a much greater extent than on the full image.

For *EffGS*, the average PSNR gain on the full image is 0.14 and 0.16 for *Video Depth Anything* and *Depth Pro*, respectively. However, when restricting the evaluation to the foreground, these improvements rise to 0.25 and 0.29, proving that a significant portion

Table 5.2: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across the Nerfies and iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	19.52	0.531	0.651	0.347	10.6	10038
DeformableGS + Depth Supervision (VideoDA)	19.36	0.523	0.631	0.415	13.2	9165
DeformableGS + Depth Supervision (Depth Pro)	19.42	0.519	0.636	0.383	13.4	9120
EffGS	20.46	0.556	0.671	0.356	97.4	**2278**
EffGS + Depth Supervision (VideoDA)	20.60	0.562	**0.678**	0.348	101.4	2643
EffGS + Depth Supervision (Depth Pro)	**20.62**	**0.563**	0.677	**0.347**	**103.5**	2470
4D-GS	19.31	0.553	0.659	0.362	36.4	7436
4D-GS + Depth Supervision (VideoDA)	19.60	0.558	0.668	0.351	35.0	7827
4D-GS + Depth Supervision (Depth Pro)	19.46	0.542	0.652	0.364	34.7	8065

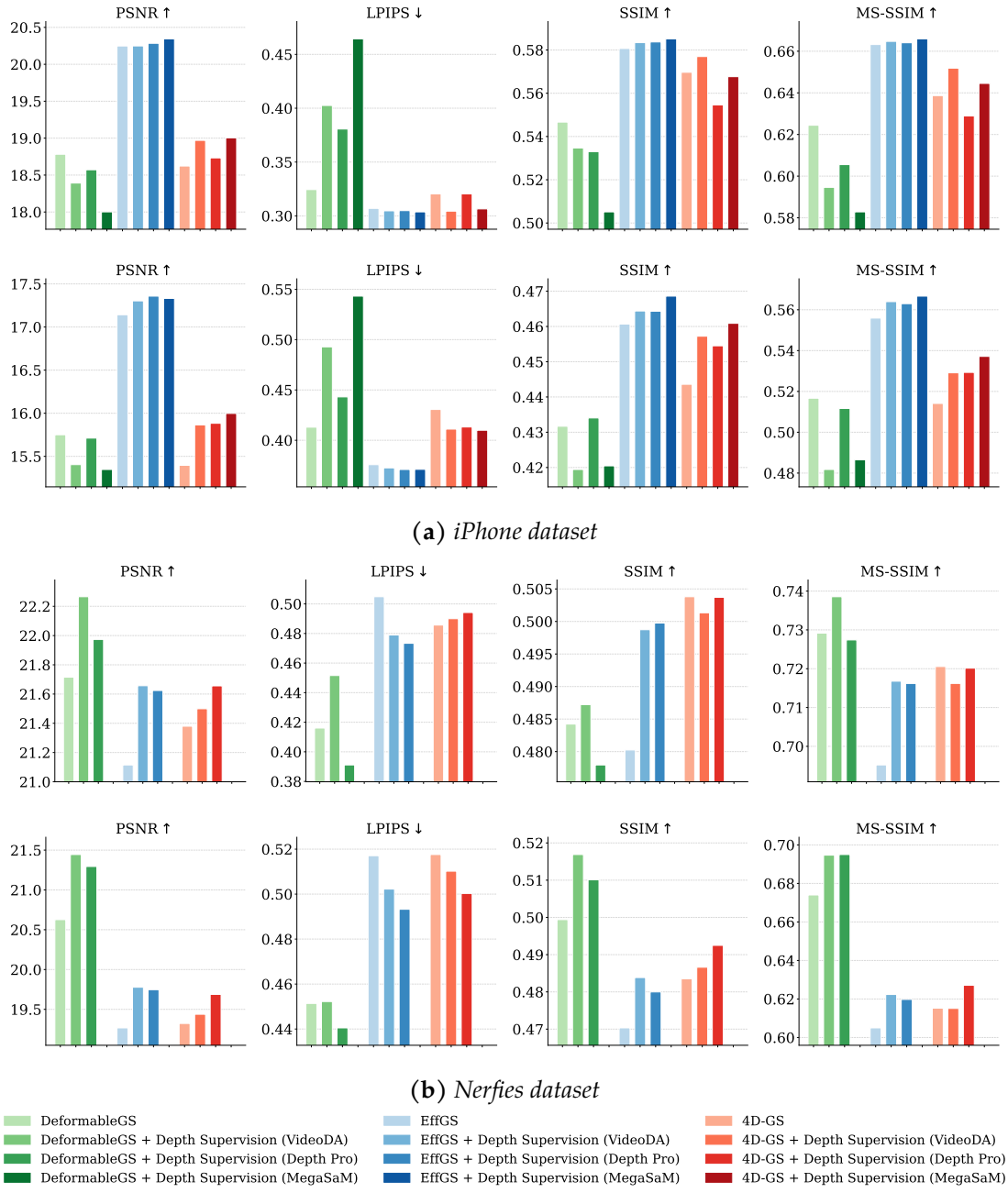
Table 5.3: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the Nerfies and iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	16.97	0.449	0.556	0.423	10.6	10038
DeformableGS + Depth Supervision (VideoDA)	16.91	0.444	0.535	0.483	14.1	9165
DeformableGS + Depth Supervision (Depth Pro)	17.11	0.453	0.557	0.442	13.6	9120
EffGS	17.67	0.463	0.568	0.411	100.1	**2278**
EffGS + Depth Supervision (VideoDA)	17.92	**0.469**	**0.579**	0.405	100.9	2643
EffGS + Depth Supervision (Depth Pro)	**17.96**	0.468	0.577	**0.402**	**105.0**	2470
4D-GS	16.38	0.454	0.539	0.452	37.1	7436
4D-GS + Depth Supervision (VideoDA)	16.76	0.465	0.551	0.436	35.3	7827
4D-GS + Depth Supervision (Depth Pro)	16.83	0.464	0.554	0.435	36.1	8065

of the overall improvement is caused specifically by dynamic regions.

A similar trend holds for *4D-GS*. While on the full-image PSNR improves by 0.29 and 0.15, restricting to the foreground reveals an improvement by 0.38 and 0.45, tripling the effect for *Depth Pro*. As the foreground regions typically cover less than half of the image, these results demonstrate that the improvements seen in full-image metrics cannot be attributed solely to the dynamic foreground regions.

Interestingly, *DeformableGS*, which generally degrades when trained with depth su-



(a) *iPhone dataset*

(b) *Nerfies dataset*

Figure 5.5: *All metrics for (a) iPhone and (b) Nerfies, reported for the full image (top) and the foreground region (bottom). The results highlight that the effect of depth-based supervision varies substantially across methods and datasets. Additionally, improvements in the foreground are much larger than those observed in the full image.*

pervision, shows a reduced drop and even a slight improvement in reconstruction quality when evaluated only on the foreground. While full-image PSNR decreases by 0.16 and 0.1, the reductions in the foreground are much smaller with 0.06 and a small improvement of 0.14. Similar observations can be made for LPIPS, indicating that even when overall performance decreases, the dynamic regions are affected less strongly. The core problem is that the poor performance on the iPhone dataset obfuscates the substantial gains on Nerfies. In fact, when evaluated exclusively on the foreground *DeformableGS* excels, by significantly outperforming all other methods across all metrics, achieving a PSNR gain of nearly 1 dB. The qualitative results on Nerfies in Figure 5.6 (middle) reinforce this observation: for the broom scene (left), *DeformableGS* is the only method that successfully learns to reconstruct the swinging broom under depth supervision. Similarly, for the *toby-sit* scene (right), it is the only method capable of modelling the dog's head motion.

Consistent with these observations, looking at the results in Figure 5.5 shows that *EffGS* and *4D-GS* exhibit improvements across all metrics and datasets when focusing on the foreground. This behavior is different from the full image evaluation, where only *EffGS* consistently benefits from depth supervision. The qualitative examples in Figure 5.6 also illustrate this effect for the iPhone scenes *pillow* (top) and *mochi-high-five* (bottom). More specifically, *4D-GS* shows much stronger reconstruction of the dynamic foreground, like the right hand in pillow or the cat's moving head in mochi-high-five, when depth supervision is applied.

EffGS Leads in Reconstruction Quality, Training Time, and Rendering Speed

When looking at the quantitative results across both the full-image metrics in Table 5.2, as well as the dynamic foreground evaluations in Table 5.3, *EffGS* and its variants using depth priors consistently outperform all of the other methods. With the exception of a single metric in each table (LPIPS for the full image and SSIM for the foreground), it always ranks as the best and second-best method across all metrics.

Qualitative results in Figure 5.7 show that the increased performance in quantitative metrics can also be observed visually. Finer details of the enclosure in the background, as well as the shape of the tail in the foreground, are reconstructed far more precisely, proving the positive influence of supervising the training with depth. At the same time, depth renderings of surfaces, such as the dog's body and the floor, become noticeably smoother.

This is especially impressive when looking at the training time and rendering speed: *EffGS* achieves renderings at frame rates three times and even ten times higher compared to *4D-GS* and *DeformableGS*, respectively. This fact directly translates into substantially shorter training times, which are roughly three times faster than those of the other two methods. Although its Gaussian count falls between the ones of *4D-GS* and *DeformableGS*, which can be seen in Figure 5.8, the faster rendering capabilities of *EffGS* stem from its lightweight motion model, which, as presented in Section 3.5, replaces

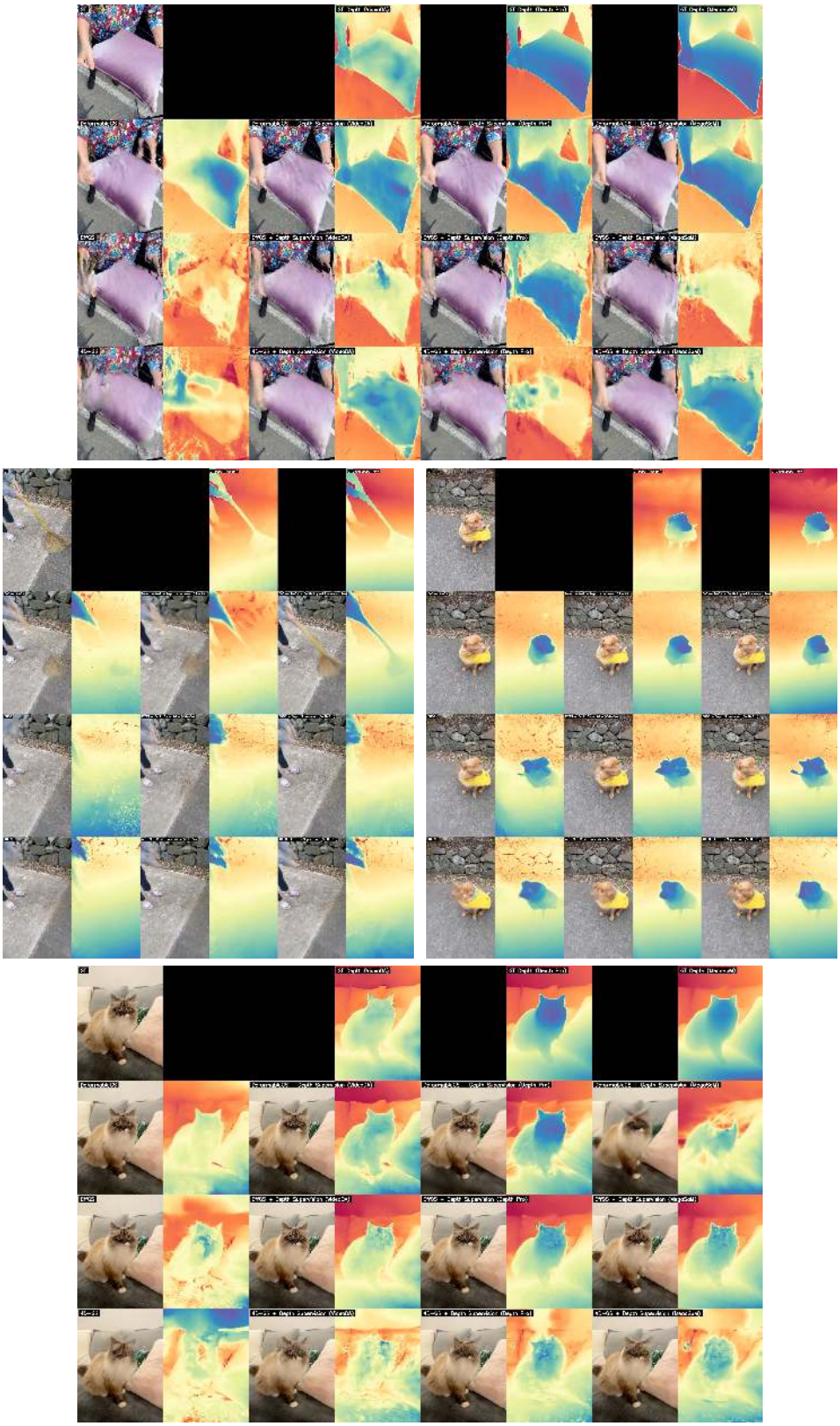


Figure 5.6: *Best-case qualitative results for the pillow (top) and mochi-high-five (bottom) scenes from the iPhone dataset, and the broom (middle-left) and the toby-sit (middle-right) scenes from the Nerfies dataset. Each visualization is structured consistently from left to right: in the first row, RGB ground truth is followed by pseudo ground truth depth predictions from Video Depth Anything (VideoDA), and Depth Pro (and MegaSaM). For each reconstruction method, DeformableGS, EffGS, and 4D-GS, the corresponding baseline RGB and depth renderings are shown, followed by their depth-supervised counterparts using the same depth prediction models. Fine-grained differences can be observed best when zooming in.*

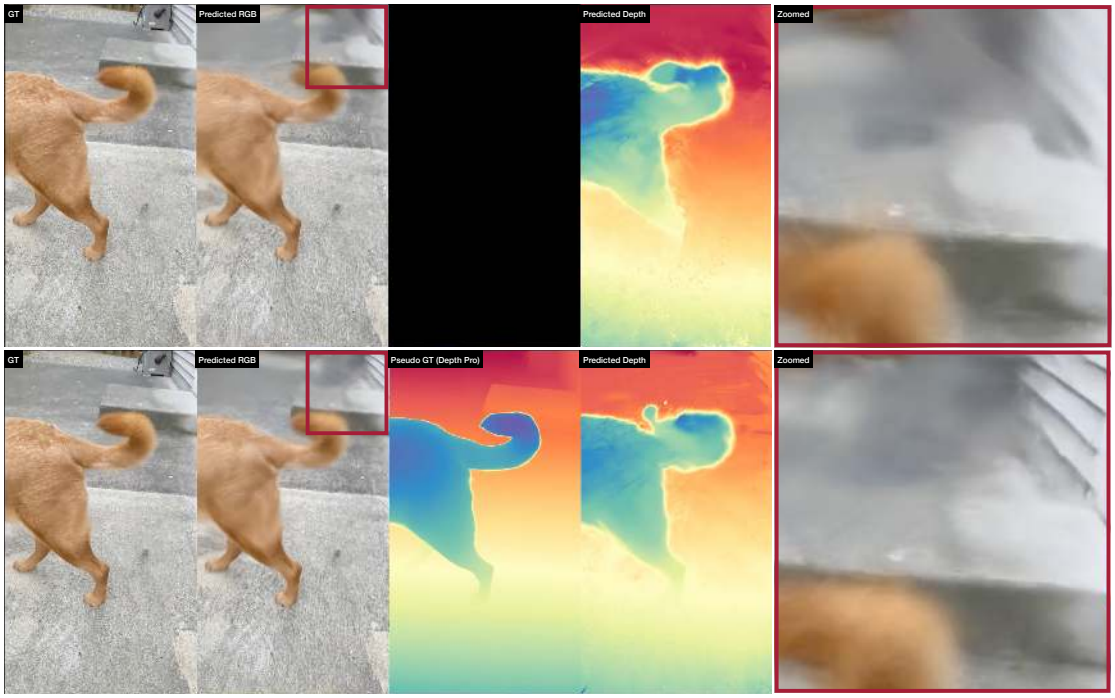


Figure 5.7: *Qualitative results on the tail scene from the Nerfies dataset, comparing the EffGS baseline (top) with its depth-supervised variant using Depth Pro (bottom). From left to right: RGB ground truth, RGB rendering, (pseudo) ground truth depth, predicted depth, and a zoomed-in view indicated by the red bounding box. Depth supervision enables significantly finer reconstruction details, both for the enclosure in the background and for the dog's body.*

the MLP-based deformation networks of the other two with a compact Fourier- and linear-based approximation functions, dramatically reducing computational cost.

Limited Impact of Depth Supervision on Gaussian Count

Although one might expect that improved geometry reconstruction through depth supervision might simplify the geometry and, consequently, reduce the number of Gaussian primitives, this does not generally hold. As shown in Figure 5.8, the only model that produces notably fewer Gaussian counts is *DeformableGS*. However, this reduction is also concomitant with a consistent drop in performance across all metrics, approximately 0.5 dB in PSNR, for instance (see Table 5.2). For the remaining methods, incorporating depth supervision leads to only minor changes in the number of Gaussians, with slight increases or decreases depending on the dataset.

In EffGS Reconstruction Quality Largely Follows the Quality of the Depth Prior

From the averaged quantitative results in Table 5.2, *Depth Pro* consistently outperforms supervision with *Video Depth Anything* across all metrics except MS-SSIM. A closer look at the per-dataset results with exact numerical values in Figure 5.9 confirms that this trend largely continues across each individual dataset. In other words, when using a more accurate depth prediction method, like *Depth Pro*, the corresponding depth-supervised reconstruction improves accordingly, compared to the lower accu-

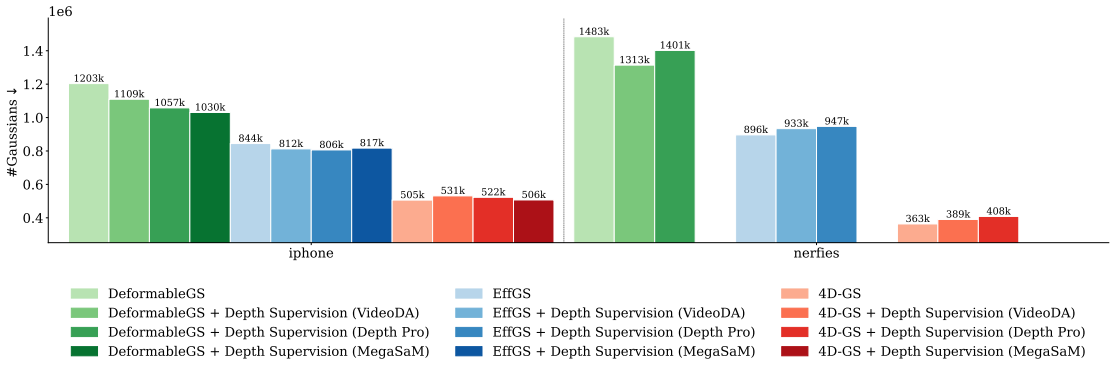


Figure 5.8: *Per-method Gaussian counts on the iPhone and the Nerfies datasets. With the exception of DeformableGS on iPhone, incorporating depth-based supervision has a negligible effect on the number of Gaussians required for reconstruction.*

racy model, *Video Depth Anything*. Similar observations can be made for *MegaSaM*, as it always yields better reconstruction scores than *Video Depth Anything* across all metrics, even for the rendering frame rate.

Interestingly, this relationship does not hold for the two remaining dynamic reconstruction models, which motivates the discussion in the following subsection.

High-Quality Depth Does Not Necessarily Yield Better Reconstructions

A more accurate depth prediction method does not guarantee better reconstruction quality than lower accuracy models. Again, this can be proven by examining Figure 5.9: for *4D-GS*, reconstructions on the iPhone dataset are constantly worse when using *Depth Pro* compared to *Video Depth Anything*, despite the latter having poorer depth estimates. A similar, though less pronounced, observation can be made when comparing *MegaSaM* against *Video Depth Anything*.

On the Nerfies dataset, the situation is quite ambivalent as some metrics, such as PSNR, SSIM, MS-SSIM, tend to improve when using better depth prediction models, while LPIPS measures an opposite trend.

For *DeformableGS*, which generally fails to benefit from additional depth supervision information (see Section 5.2.1), the outcome is expected to some extent: using higher quality depth prediction models on the iPhone dataset often leads to worse reconstruction results. This effect also partly extends to the Nerfies dataset.

Limited Impact of Video Consistent Depth Prediction Models

Another key question addressed in this thesis is whether enforcing temporal consistency in depth predictions used as priors leads to quantifiable gains in dynamic reconstruction quality. Figure 5.9 indicates that *EffGS* is not sensitive to temporal consistency as the reconstruction quality correlates primarily with the accuracy of the depth estimates rather than their temporal stability.

For *4D-GS*, the influence of video-consistent depth is similarly small on the Nerfies dataset. However, on the iPhone dataset, both *Video Depth Anything* and *Depth Pro*,

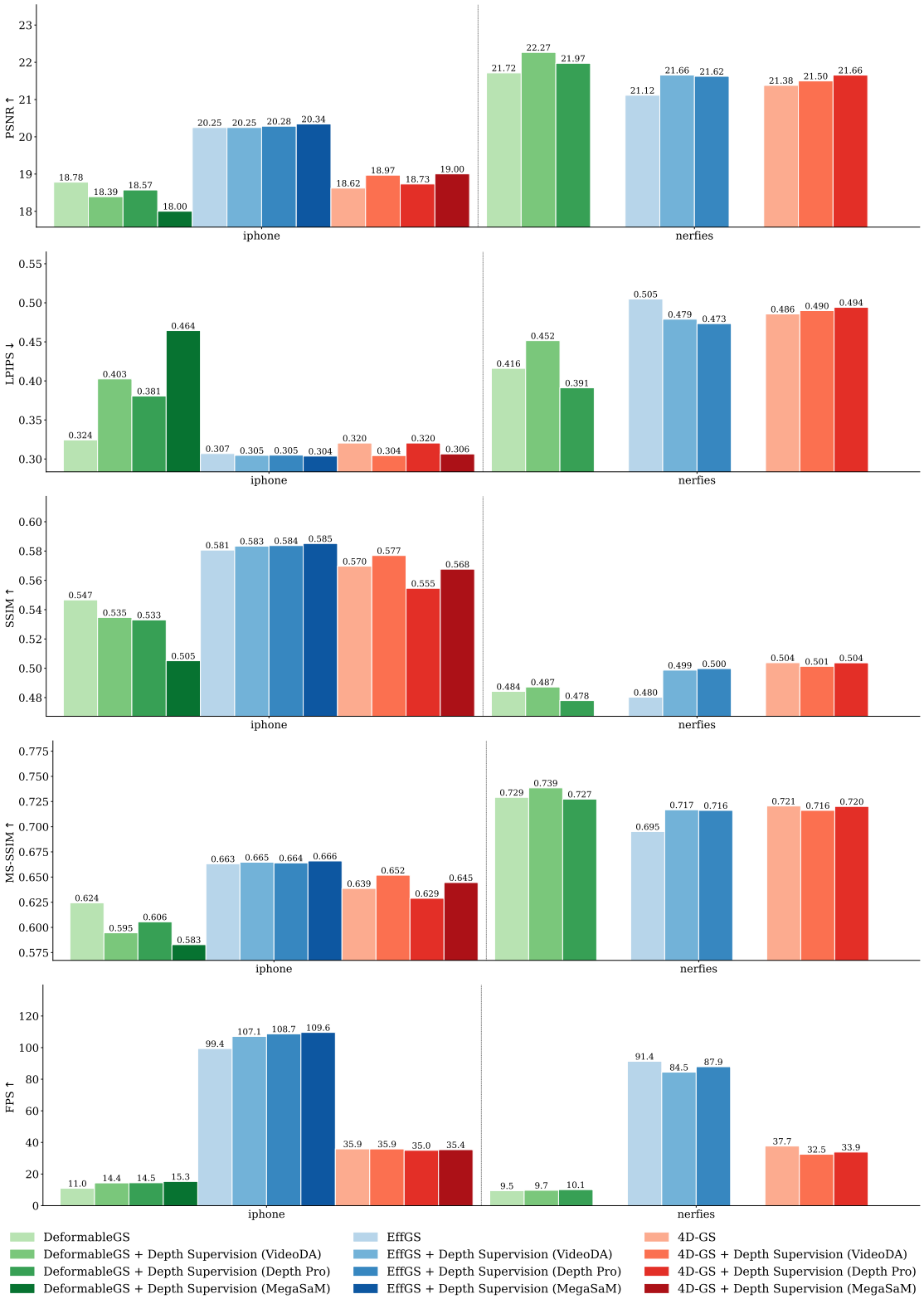


Figure 5.9: *Per-method quantitative results on the iPhone and the Nerfies datasets for PSNR, LPIPS, SSIM, MS-SSIM, and FPS evaluated on the full image. Corresponding numerical values are plotted above each bar. It indicates that EffGS strongly reflects the quality of the underlying depth prior, which does not always hold for the other methods. The plots also show that video-consistent depth prediction provides only a marginal additional benefit, if any.*

the two methods providing video-consistent depths, achieve consistently higher scores across all metrics.

As in previous sections, the behaviour of *DeformableGS* is counterintuitive as it deviates from the general trend. On the iPhone dataset, degradation is smaller when using *Depth Pro*, even though it does not provide video-consistent depth predictions. For the Nerfies on the other hand, *Video Depth Anything* outperforms *Depth Pro* in PSNR, SSIM, and MS-SSIM, indicating that temporal consistency can be beneficial under certain motion patterns or scene characteristics.

Depth Supervision Acts as a Regularizer

One of the initial hypotheses was that depth supervision would have a regularizing influence on the reconstruction process. The results in Figure 5.10 indeed show that across most settings, the gap between training and test performance decreases once depth priors are incorporated. In line with the general performance gains in reconstruction quality, *EffGS* consistently benefits from depth-based supervision, showing a small reduction in the train-test gap on the iPhone dataset and a substantial reduction for Nerfies across all metrics.

For *4D-GS*, again, the situation is mixed, as some metrics show reduced gaps on the iPhone dataset but not on Nerfies, while others reveal the opposite. However, it is

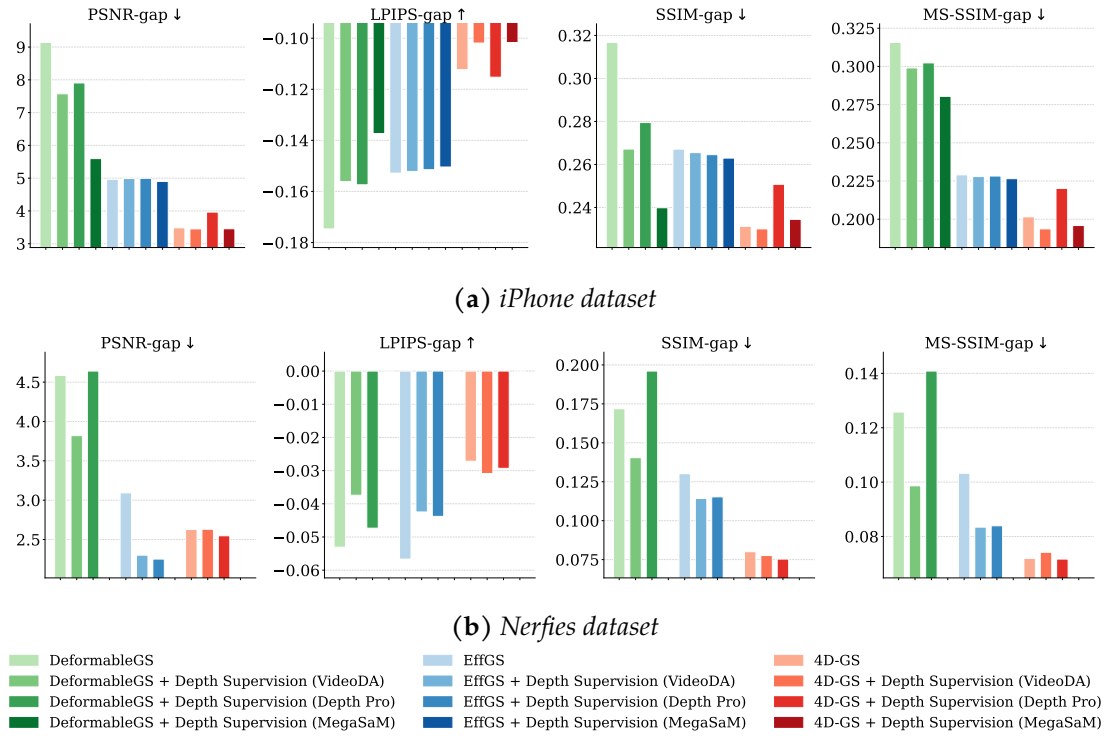
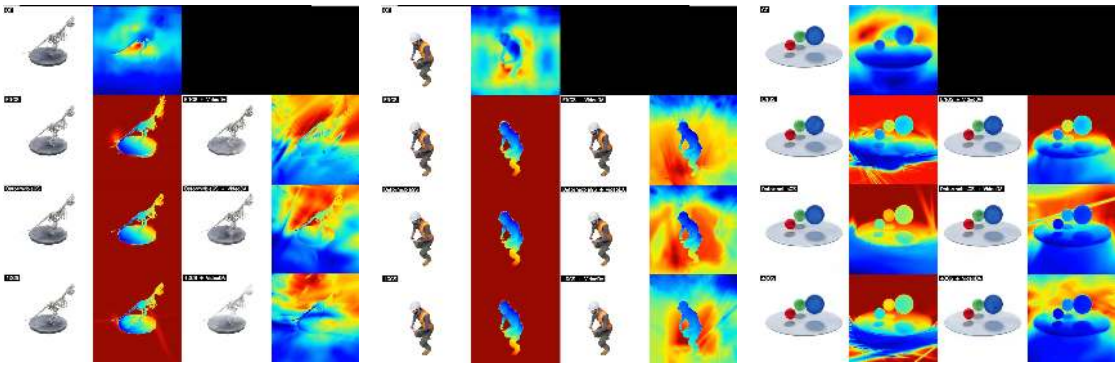


Figure 5.10: *Train-test gaps for all metrics on (a) iPhone and (b) Nerfies, reported for the full image. The results highlight the regularizing effect of depth-based supervision across the different methods. For most of them, the quantitative difference between train and test set renderings is reduced, indicating that depth supervision helps partially to reduce overfitting and improve generalization.*



(a) (b) (c)

Figure 5.11: *Qualitative results of incorporating depth priors via supervision into the dynamic reconstruction pipeline for the (a) trex, (b) standup, and (c) bouncing balls scenes from the D-NeRF dataset. In each visualization, the first row shows the ground truth RGB image and the corresponding pseudo ground truth depth prediction. The subsequent rows show for each method (EffGS, DeformableGS, 4D-GS), the baseline RGB and depth renderings on the left, followed by the equivalent renderings with Video Depth Anything depth supervision on the right. The results clearly show that both the depth-supervised depth renderings and even the pseudo ground truth depth predictions look significantly worse than the baseline renderings.*

noticeable that, similar to the regular performance, the version using *Depth Pro* always tends to produce much larger gaps than both the baseline and the other supervised variants on the iPhone dataset.

Although *DeformableGS* generally suffers in overall reconstruction quality when depth supervision is used, it shows a clear reduction in overfitting. On the iPhone dataset, the train-test gap decreases significantly, with the largest improvement observed for PSNR, reducing from 9 in the baseline to 5.5 when using *MegaSaM*. Similar reductions are observed on Nerfies as well, with the exception of the *Depth Pro* variant.

5.2.2 Failure Cases

Integrating depth priors into the reconstruction pipeline does not always improve results. This is particularly evident when the scenes are already densely observed and well captured, but at the same time, the pseudo ground truth depth predictions are noisy and inaccurate. For instance, in the D-NeRF dataset Pumarola et al., 2021, the reconstruction objects are captured from many viewpoints at different time steps, providing good coverage for learning accurate geometry and appearance. As a result, baseline models already achieve excellent performance as reported by Liang et al., 2025, with *EffGS*, *4D-GS*, and *DeformableGS* achieving PSNR values of 30.52, 33.27, and 37.14, respectively.

The qualitative results in Figure 5.11 show the effect of adding depth supervision in such cases. Firstly, the baseline depth renderings are visibly more accurate than the pseudo ground truth prediction provided by *Video Depth Anything* (*VideoDA*). Secondly, incorporating these inaccurate depth priors can actually harm the geometry reconstruction quality, introducing errors and artifacts that would not be present if the

model relied solely on the numerous RGB multiview observations.

5.3 Depth-Based Initialization of Gaussians

This section summarizes the key outcomes of the depth-based Gaussian initialization experiments. In Section 5.3.1, different configurations of the progressive dataset scheduler introduced in Section 4.2.3 are evaluated. Subsequently, Section 5.3.2 presents the central observations on how the depth-guided Gaussian initialization process influences reconstruction quality across methods and datasets. The section concludes with a discussion of the main limitations identified during experimentation in Section 5.3.3.

5.3.1 Evaluating Progressive Dataset Scheduling

In Section 4.2.3, a *Progressive Dataset Scheduling* strategy was proposed to help the model in the early training phase after initialization by gradually increasing the number of training images used for optimization. To estimate the effect of the custom scheduler on reconstruction, four different configurations are examined: the first starts with a single training image at iteration 0 and adds one additional training view every 10 iterations to the set of training images used for random sampling. Secondly, a scheduler with the same variant but a slower expansion rate, adding one new image every 20 iterations. A third setup follows a fraction-based progression in which the number of images in the training set grows linearly starting also with a single image at iteration 0 and reaching the full dataset by iteration 3,000. Lastly, for comparison, results are provided for the baseline configuration where no custom scheduling is applied, and the model relies solely on the standard random sampling strategy.

Each variant introduces its own trade-offs. The fraction-based version is sensitive to scenes of different sizes, as the growing rate is dependent on the number of training images. For scenes with only dozens of training views, the slowly growing rate can lead to overfitting, while for large scenes, the effect of scheduling might be negligible. The fixed-step schedules mitigate this issue by explicitly controlling the expansion rate, yet at the cost of potentially delaying exposure to the full set of views in scenes that contain several hundred images.

All configurations are evaluated using *EffGS* and *4D-GS* on the *teddy* and *haru-sit* scenes of the iPhone dataset. The reason why these two scenes were chosen is that they differ substantially in the number of training images, containing 472 images for *teddy* but only 50 for *haru-sit*.

As expected, the experiments indicate that the scheduling procedure implants strong overfitting to the training views. This can be seen in Figure 5.12, where the train-test gap widens when depth-based Gaussian initialization is used. However, the standard scheduling strategy generally yields a comparable or sometimes even smaller gap, reducing the overfitting to some extent. More importantly, disabling the progressive dataset scheduling altogether yields the best quantitative results for LPIPS, SSIM, and

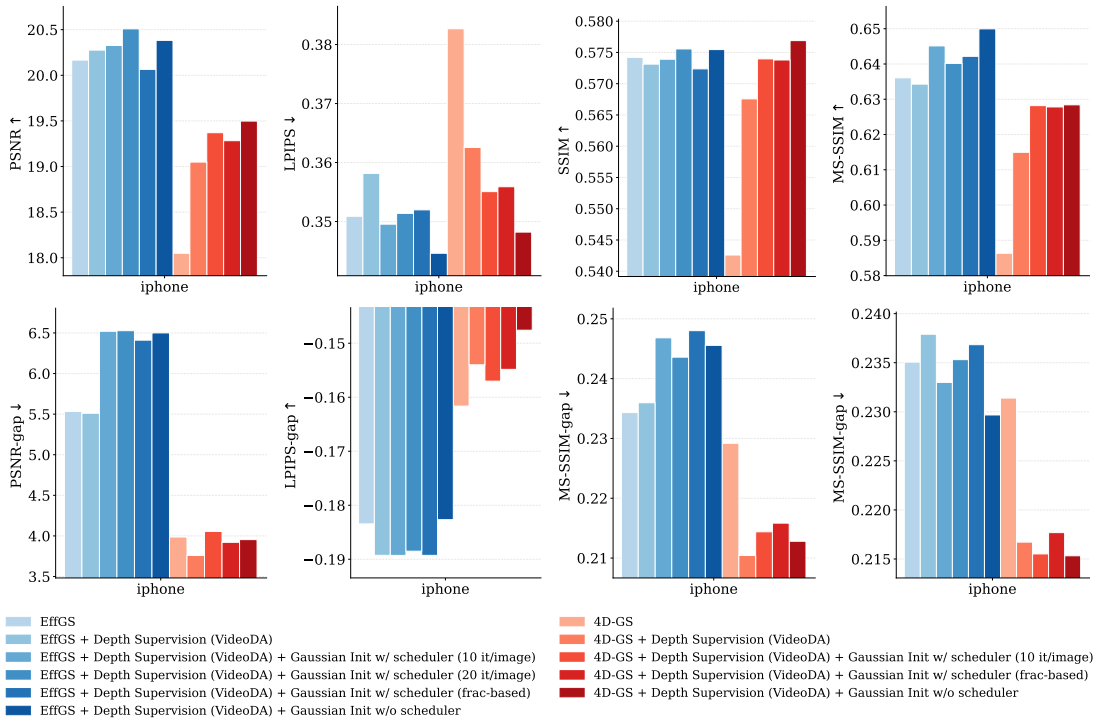


Figure 5.12: *Test metrics performance (top) and corresponding train-test gaps (bottom) for different sampler configurations on EffGS and 4D-GS, averaged over the haru-sit and teddy scenes of the iPhone dataset. Aside from the baseline and the depth supervision setting, the comparison includes several variants of progressive dataset scheduling: one configuration that adds an image to the set of randomly sampled training images every 10 iterations, a second that uses 20 iterations per image, a fraction-based variant that increases the proportion of available images linearly until the full set is reached at iteration 3,000, and the standard version which always samples images randomly from the full set.*

MS-SSIM, and the second-best results for PSNR for the *EffGS* method. Similarly, it excels across all four metrics for the *4D-GS* method.

5.3.2 Key Observations

Table 5.4 summarizes the quantitative results averaged over both the iPhone and the Nerfies datasets. The following sections discuss the most relevant trends and method-specific behaviours revealed by these experiments. Additional figures and extended plots are provided in the supplementary (see Appendix C).

Influence of Depth-Based Initialization Varies Across Methods and Datasets

Similarly to Section 4.2, the main findings of this section are that the impact of depth-based initialization on dynamic 3D reconstruction pipelines is highly method- and dataset-dependent. Its effects range from clear improvements to substantial performance degradation and everything in between. As shown in Table 5.2, integrating depth-guided Gaussian initialization as well does not provide a universally positive contribution.

The largest negative effect is again observed for *DeformableGS*. Here, enabling depth-

Table 5.4: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across Nerfies and iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	19.52	0.531	0.651	0.347	10.6	10038
DeformableGS + Depth Supervision (VideoDA)	19.36	0.523	0.631	0.415	13.2	9165
DeformableGS + Depth Supervision (Depth Pro)	19.42	0.519	0.636	0.383	13.4	9120
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	19.31	0.490	0.610	0.495	15.5	7997
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	18.86	0.476	0.589	0.545	18.3	7014
EffGS	20.46	0.556	0.671	0.356	97.4	**2278**
EffGS + Depth Supervision (VideoDA)	20.60	0.562	0.678	0.348	101.4	2643
EffGS + Depth Supervision (Depth Pro)	20.62	0.563	0.677	0.347	**103.5**	2470
EffGS + Depth Supervision + Gaussian Init (VideoDA)	**20.69**	**0.564**	**0.684**	**0.337**	101.8	2665
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	20.67	0.560	0.676	0.339	102.1	2573
4D-GS	19.31	0.553	0.659	0.362	36.4	7436
4D-GS + Depth Supervision (VideoDA)	19.60	0.558	0.668	0.351	35.0	7827
4D-GS + Depth Supervision (Depth Pro)	19.46	0.542	0.652	0.364	34.7	8065
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	19.87	0.559	0.675	0.366	36.5	8200
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	20.12	0.561	0.675	0.362	37.2	8027

based initialization further decreases performance across all metrics compared to using depth supervision alone. While the baseline and *Depth Pro*-supervised model achieve average PSNR values of 19.52 and 19.42 across the iPhone and Nerfies datasets, incorporating depth-based initialization reduces the score to 18.86. Unlike the supervision setting, this decline is consistent across both datasets, as illustrated in Figure 5.5.

The outcome for *4D-GS* again is double-sided. As the gains on the iPhone dataset are so large, they conceal the poor results on the Nerfies dataset, dominating the overall average. In terms of PSNR, *4D-GS* benefits the most from depth-based initialization, far more than from depth supervision alone. Averaged across both datasets, the highest improvement can be seen for *Depth Pro* where PSNR increases by 0.66 and 0.81 compared to depth supervision and the baseline, respectively. In contrast, gains in LPIPS, SSIM, and MS-SSIM are rather small and vary with the choice of the depth prediction model. In fact, when considering only the iPhone dataset, Figure 5.13, *4D-GS* shows the strongest improvement of the entire experiment: PSNR rises from 18.62 for the baseline to 19.80 using *Depth Pro* based Gaussian initialization. That is an increase of roughly 6 % or 1.18 dB.

Table 5.5: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the Nerfies and iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	16.97	0.449	0.556	0.423	10.6	10038
DeformableGS + Depth Supervision (VideoDA)	16.91	0.444	0.535	0.483	14.1	9165
DeformableGS + Depth Supervision (Depth Pro)	17.11	0.453	0.557	0.442	13.6	9120
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	17.11	0.451	0.549	0.481	16.2	7997
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	16.77	0.438	0.533	0.536	18.7	7014
EffGS	17.67	0.463	0.568	0.411	100.1	**2278**
EffGS + Depth Supervision (VideoDA)	17.92	0.469	0.579	0.405	100.9	2643
EffGS + Depth Supervision (Depth Pro)	17.96	0.468	0.577	0.402	105.0	2470
EffGS + Depth Supervision + Gaussian Init (VideoDA)	17.98	0.471	**0.583**	**0.391**	99.5	2665
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	**17.99**	0.463	0.573	0.394	**105.5**	2573
4D-GS	16.38	0.454	0.539	0.452	37.1	7436
4D-GS + Depth Supervision (VideoDA)	16.76	0.465	0.551	0.436	35.3	7827
4D-GS + Depth Supervision (Depth Pro)	16.83	0.464	0.554	0.435	36.1	8065
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	17.18	**0.481**	0.580	0.421	37.4	8200
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	17.27	0.479	0.578	0.419	37.7	8027

On the other hand, *EffGS* cannot continue its trend of reliably increasing results independent of dataset, metric, and depth prediction model. While depth-based initialization still provides clear benefits on the iPhone dataset, this is no longer true for Nerfies, where performance stagnates or drops for all metrics except LPIPS.

Returning to *4D-GS*, the qualitative results in Figure 5.14 reveal a completely different behavior on the Nerfies dataset. With the exception of the *curls* scene, depth-based initialization leads to a substantial increase in foreground detail across the remaining scenes. This qualitative improvement is consistent with the per-scene quantitative analysis, which reports foreground PSNR gains of approximately $1 - 2$ dB.

However, the *curls* scene illustrates a remarkable failure case, where reconstruction quality suffers extremely, resulting in a drop of roughly 4 dB in PSNR. As a result, this degradation outweighs the gains observed in the other scenes, which is the reason

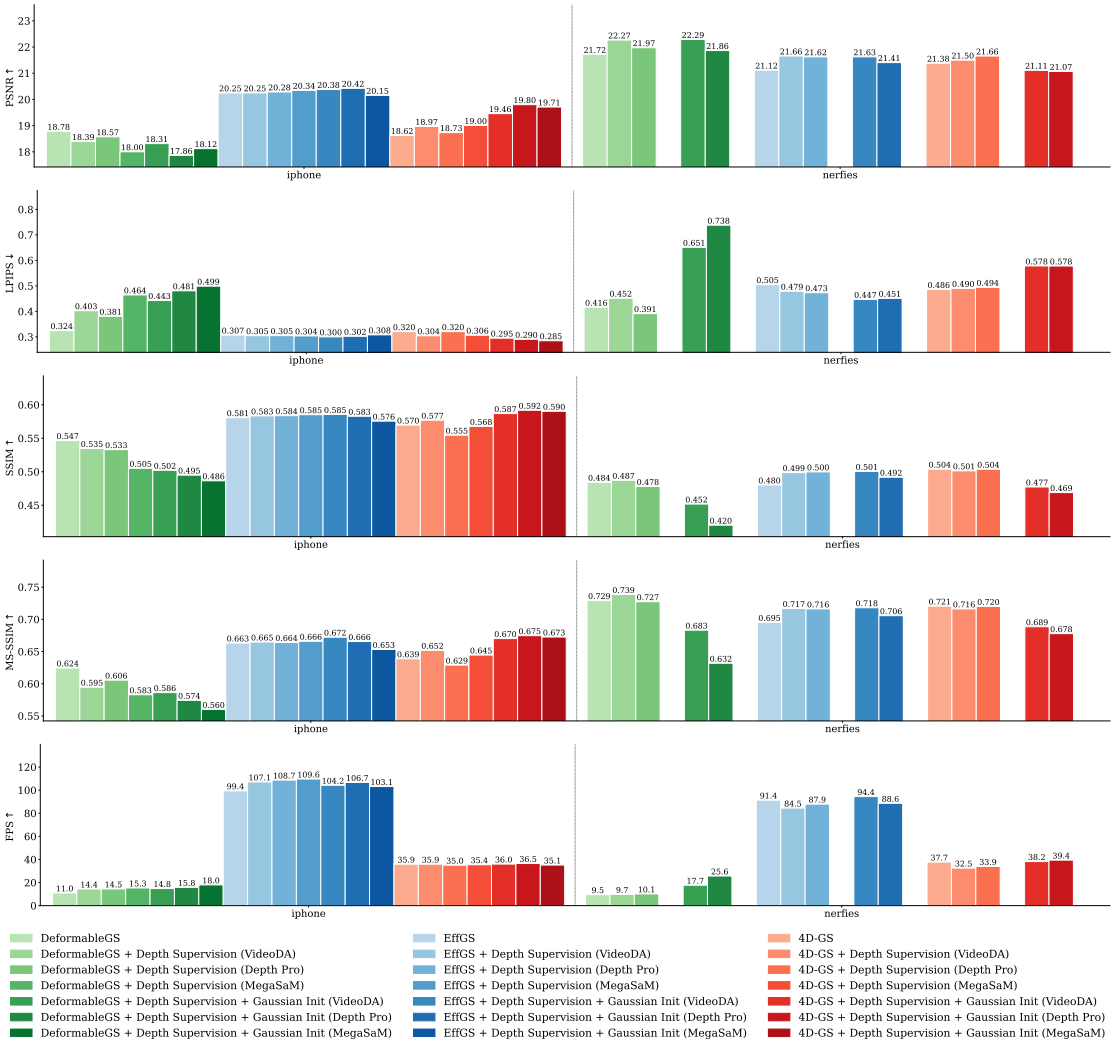


Figure 5.13: *Per-method quantitative results on the iPhone and the Nerfies datasets for PSNR, LPIPS, SSIM, MS-SSIM, and FPS evaluated on the full image. Corresponding numerical values are plotted above each bar. It indicates that depth priors with higher accuracy do not automatically result in better reconstructions. Additionally, choosing a video consistent depth prediction method provides no noteworthy benefit.*

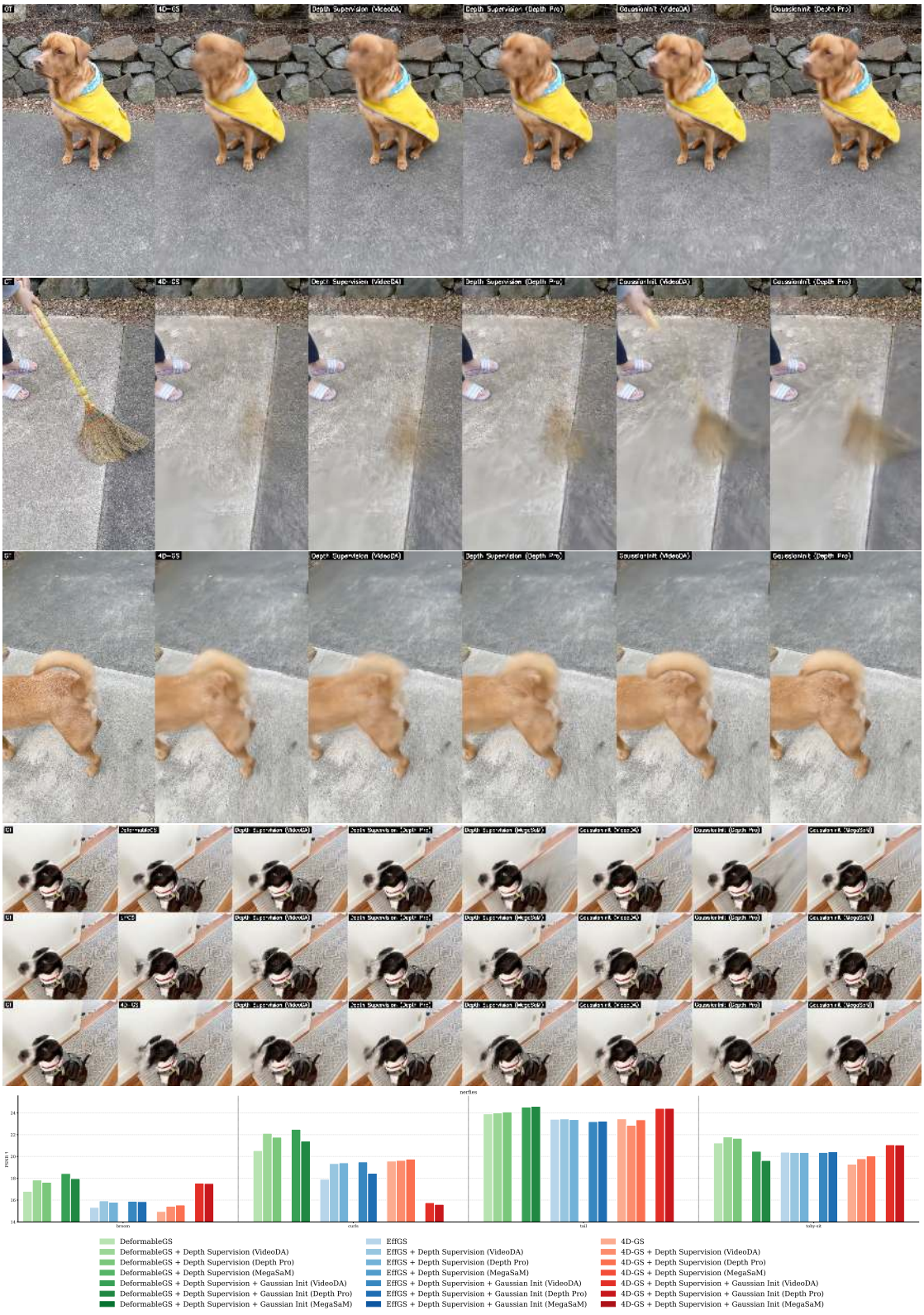


Figure 5.14: *Quantitative and qualitative best-case results for depth-based initialization using 4D-GS on the toby-sit (first row), broom (second row), and tail (third row) scenes from the Nerfies dataset, as well as the haru-sit scene from the iPhone dataset (fourth row). Each row is ordered from left to right as follows: RGB ground truth, reconstructions using depth supervision only with Video Depth Anything (VideoDA), Depth Pro (and MegaSaM), followed by reconstructions that additionally apply depth-based initialization with the same depth prediction methods. For haru-sit, corresponding results for DeformableGS and EffGS are also included. Across all scenes, depth-based initialization leads to visibly improved foreground reconstructions. This is supported by the quantitative results in the last row reporting per Nerfies scene PSNR values for each method. Fine-grained differences can be observed best when zooming in.*

why the improvement across the dataset vanishes. Therefore, one should conclude that *4D-GS* benefits from depth-based initialization across all datasets, except for the *curls* scene.

Foreground Regions Do Not Disproportionally Benefit from Depth-Based Initialization

Averaged across all datasets, depth-based initialization only yields low gains on foreground regions and only surpasses the depth-supervised variant in the case of *4D-GS*, as shown in Table 5.4 and Table 5.5. Concretely, using *Depth Pro*, the increase in PSNR from depth supervision to depth-based initialization is 0.66 on the full image but only 0.44 on the foreground.

Interestingly, further investigations of Figure 5.15 demonstrate that injecting depth information during Gaussian initialization can even harm background reconstruction on Nerfies. Although all metrics report a significant decline in the background when enabling initialization, the dynamic foreground remains largely unaffected, showing comparable or slightly improved results across all metrics, indicating that the degradation is mostly caused by static regions of the scene.

Limited Impact of Depth-Based Initialization on Gaussian Count

Similarly to the depth supervision experiment, one might expect that providing a high-quality Gaussian initialization would reduce the number of primitives required during training, since the model could just refine and already have a reasonable distribution instead of discovering less accurate geometry from scratch. In practice, this forecast turns out to be false.

As illustrated in Figure 5.16, the only method producing notably fewer Gaussian counts is *DeformableGS*. However, this comes at a high cost as the initialization step further degrades reconstruction quality across all metrics. In fact, compared to the depth-supervision-only variant, depth-based initialization introduces an additional drop of approximately 0.5 dB in PSNR (see Table 5.2). The effect is particularly drastic on the Nerfies dataset, where *DeformableGS* uses only about a quarter of the Gaussians of the baseline while almost doubling the LPIPS (see Figure 5.15).

For the remaining methods, incorporating depth-based initialization has only a minor influence on the Gaussian count. Depending on the model, either small increases as for *EffGS* or slight reductions for *4D-GS* can be observed.

Gaussian Initialization Degrades Performance in DeformableGS

Building on the previously detected performance drop of *DeformableGS* under depth supervision, this trend aggravates across all metrics and datasets when depth is additionally used for Gaussian initialization. However, it is conspicuous that PSNR deteriorates far less than the perceptual measures. For instance, on Nerfies, the PSNR for

Depth Pro decreases only slightly, from 21.97 under depth supervision to 21.86 with depth-based initialization, while LPIPS nearly doubles, rising from 0.391 to 0.738.

A qualitative inspection reveals that this discrepancy can be attributed to well-known limitations of PSNR, which can report high similarity despite severe distortions. Figure 5.17 shows several occurrences of this effect across the *toby-sit* and *broom* scenes from the Nerfies dataset, as well as the *backpack* and *teddy* scenes from the iPhone

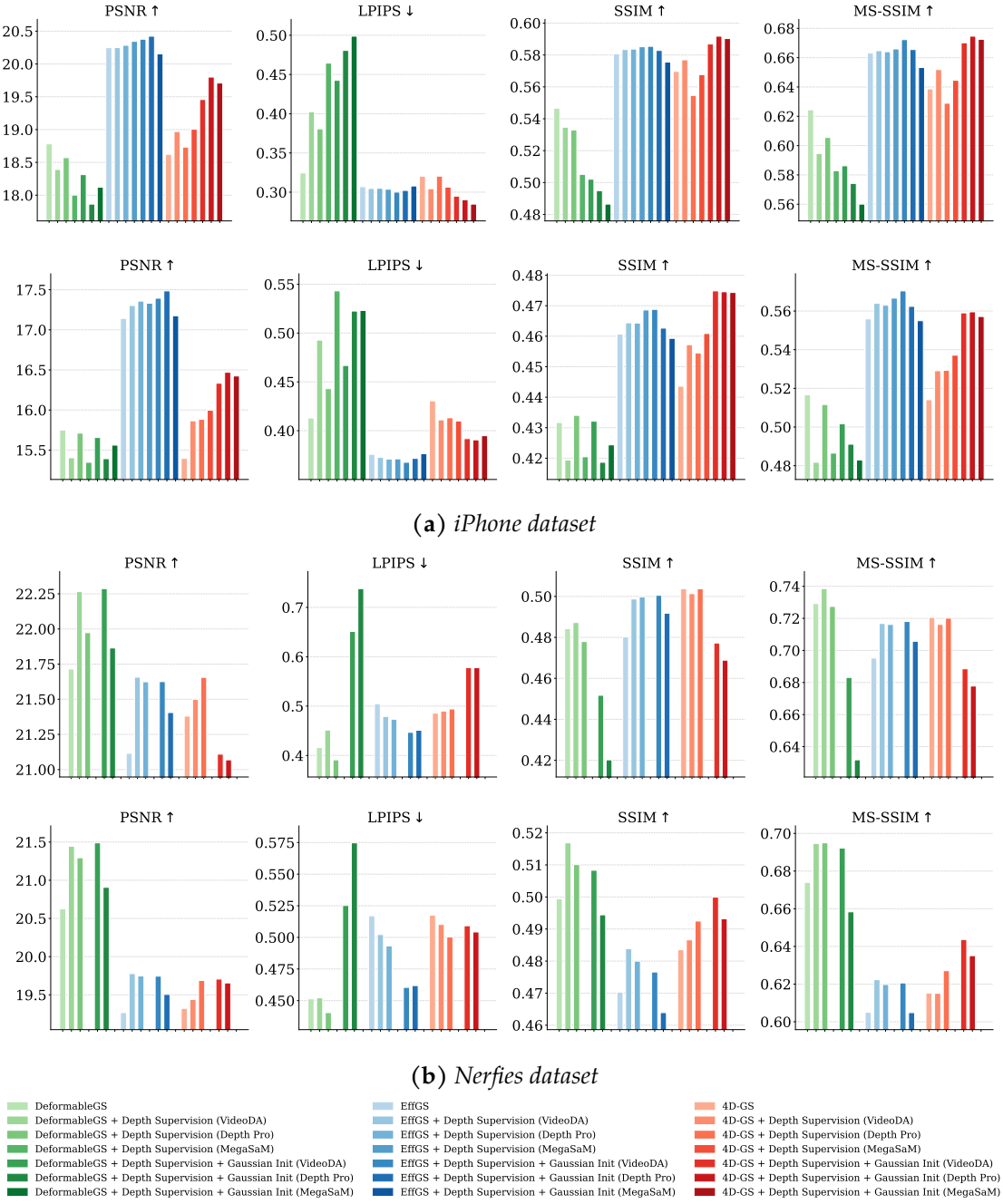


Figure 5.15: *All metrics for (a) iPhone and (b) Nerfies, reported for the full image (top) and the foreground region (bottom). The results highlight that the effect of depth-based supervision varies substantially across methods and datasets. Furthermore, improvements on the dynamic foreground are generally not larger than those measured on the full image.*

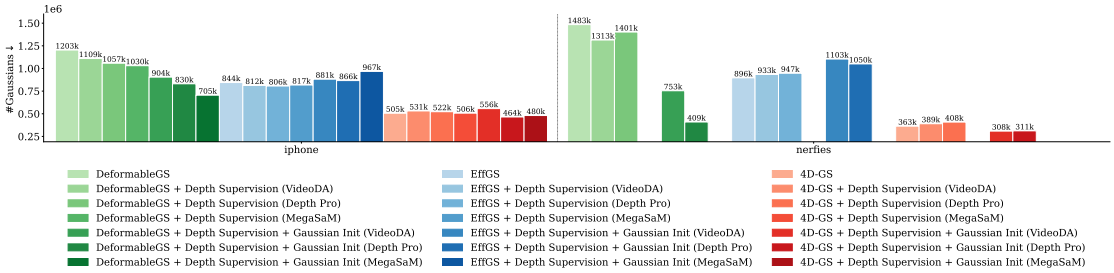


Figure 5.16: *Per-method Gaussian counts on the iPhone and the Nerfies datasets. With the exception of DeformableGS on iPhone, incorporating depth-based Gaussian initialization has a negligible effect on the number of Gaussians required for reconstruction.*

Figure 5.17: *Qualitative comparison of DeformableGS reconstructions for the toby-sit and broom scenes from the Nerfies dataset and the backpack and teddy scenes from the iPhone dataset (top to bottom). For each scene, the following outputs are shown from left to right: RGB ground truth (GT), baseline reconstruction (DeformableGS), depth-supervised reconstructions (Depth Supervision) using Video Depth Anything, Depth Pro (and MegaSaM), and depth-based initialized reconstructions (Gaussian Init) using Video Depth Anything, Depth Pro (and MegaSaM). The degenerated background reconstruction explains the large drops in LPIPS, SSIM, and MS-SSIM, which occur primarily when depth-based initialization is activated.*

dataset. Notably, these artifacts, which are mostly visible in the background, already appear in the depth supervision variant but become stronger when depth-based initialization is enabled.

High-Quality Depth Does Not Necessarily Yield Better Reconstructions

One of the central questions of this study is whether improving the accuracy of the depth prediction model translates to better reconstruction quality when using depth-based Gaussian initialization. The results in Figure 5.13 show that this relationship does not hold in general. Only *EffGS* on the iPhone dataset reliably benefits from higher-quality depth, with *MegaSaM*-based initialization consistently outperforming *Depth Pro*-based variant across all metrics.

For all remaining methods and datasets, no such pattern can be observed. Higher accuracy depth predictions do not systematically improve reconstruction; instead, in most cases, they achieve negligible gains or even degradations. This indicates that the effectiveness of depth-based initialization is strongly method-dependent, meaning that it does not directly correlate with the quality of the depth prior.

Video Consistency of Depth Prediction Models Provides No Advantage

Compared to the Depth supervision-only setting, where *4D-GS* profited from using video-consistent depth predictions, no such pattern can be observed when applying depth-based initialization. As illustrated in Figure 5.13, the choice between video-consistent and per-frame depth prediction does not reliably influence reconstruction quality in this scenario.

Depth-Based Initialization Increases Overfitting Rather than Regularizing

Contrary to the intuition that a better geometric initialization might reduce overfitting to the training views, the results in Figure 5.18 show the opposite. With the exception of *DeformableGS*, depth-based initialization consistently increases the train-test gap for *EffGS* across all metrics on both the iPhone and the Nerfies datasets.

For *4D-GS* this behavior can also be perceived for the Nerfies dataset, whereas for the iPhone dataset it depends on the chosen metric. Relative to the depth supervision case, SSIM and MS-SSIM report smaller gaps, while PSNR and LPIPS gaps remain close to their baseline and depth-supervised counterparts. More specifically, for these two metrics, higher-accuracy depth priors tend to exhibit smaller gaps than their lower-quality competitors.

5.3.3 Limitations

What all the quantitative metrics above fail to capture is the fact that *DeformableGS* is the only method capable of producing meaningful dynamic reconstructions for scenes containing true novel-view synthesis test images like *apple*, *block*, *spin*, and *teddy*. In

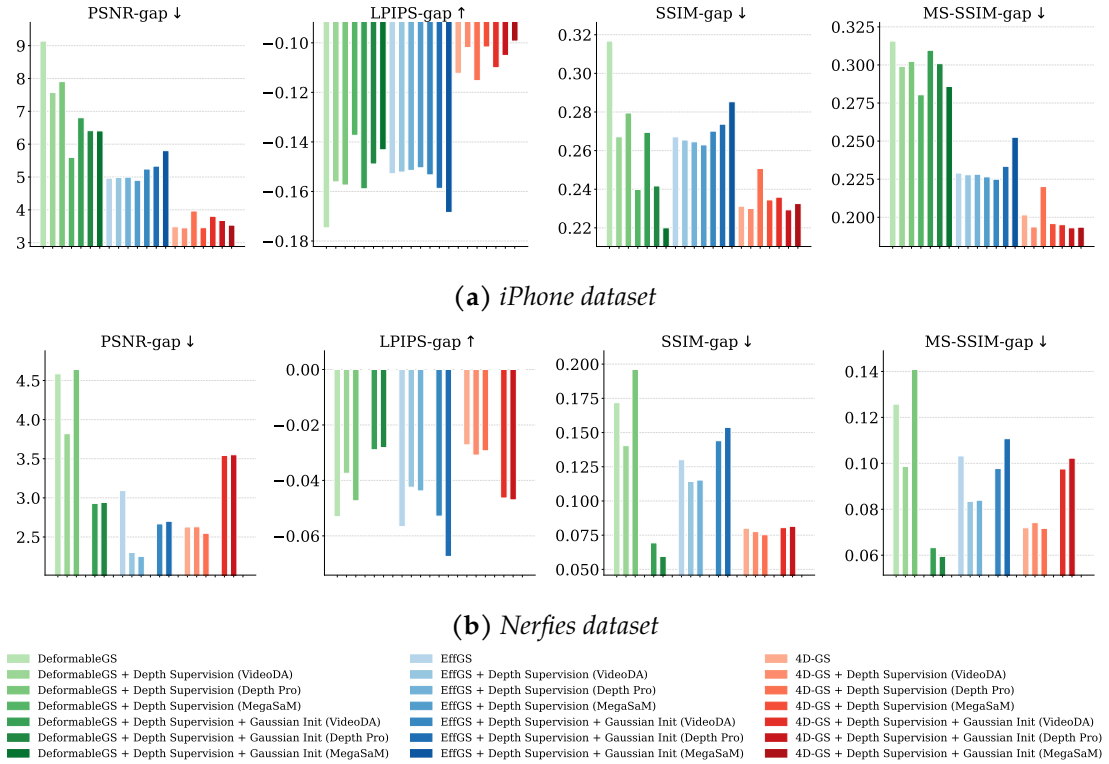
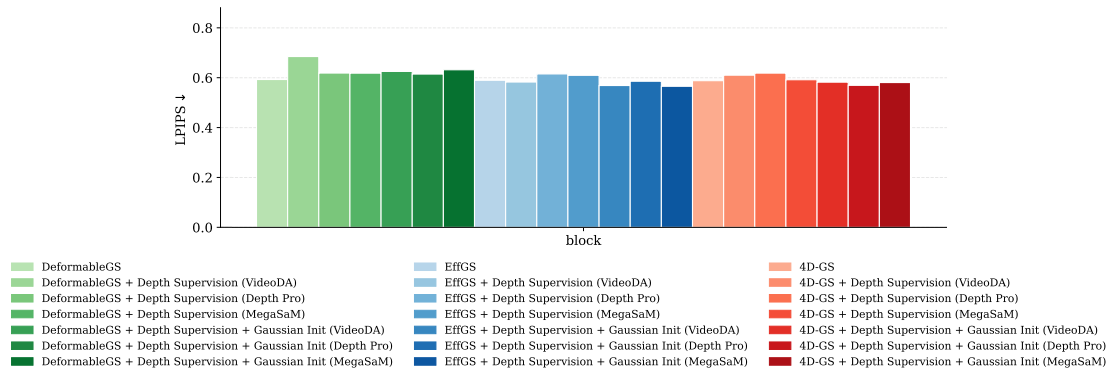


Figure 5.18: *Train-test gaps for all metrics on (a) iPhone and (b) Nerfies, reported for the full image. The results highlight the overfitting effect of depth-based initialization across the different methods. For most methods, the difference between training and test set performance increases, indicating that in most cases, depth-based initialization injects overfitting and does not improve generalization.*

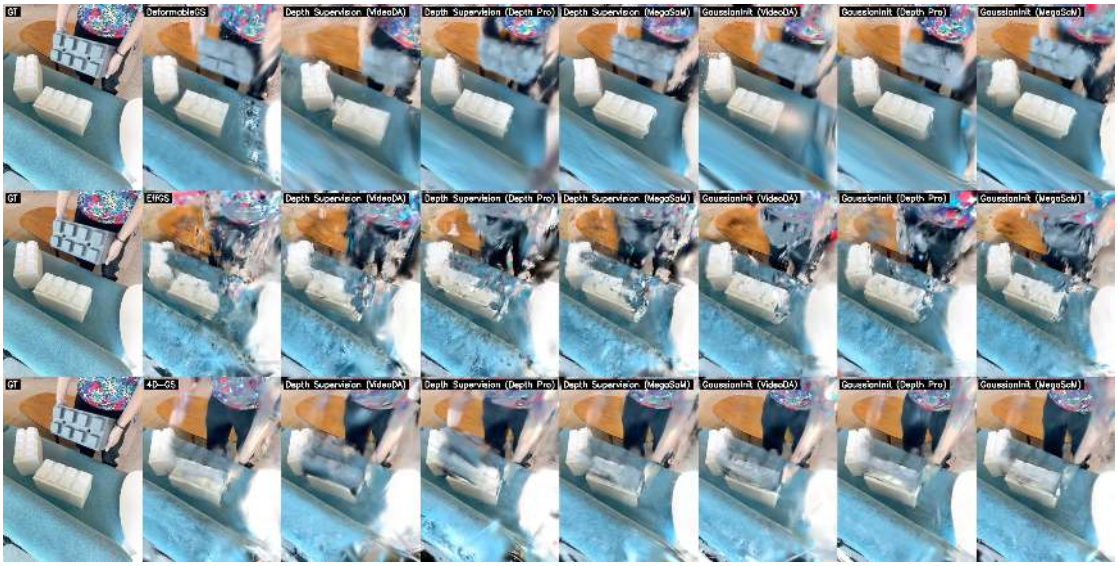
these cases, the model is able to capture the motion and deformation of objects for previously unseen viewpoints. However, *DeformableGS* suffers significantly from scene drift over time, which leads to inconsistencies in the foreground and background as well as misalignment across frames. This drift is the primary reason why the reported standard metrics do not reflect its superior motion reconstruction.

Figure 5.19 tries to illustrate this phenomenon through a set of static visualizations. Figure 5.19a indicates that LPIPS generally reports higher (worse) values for *DeformableGS* compared to the other methods. At the same time, its visual reconstruction is closest to the ground truth as DeformableGS is the only method that places the block near the true position (see Figure 5.19b. In contrast, EffGS fails to reconstruct the block completely, while 4D-GS renders it as a translucent, static structure at the final position (on top of the other block). However, the supremacy of DefromableGS' motion model is not reflected in the metrics due to scene drifting issues, which are shown in Figure 5.19c. Although the block itself is reconstructed accurately, the person holding it is significantly shifted to the right, such that the right arm falls outside the mask and the left arm is already out of frame.

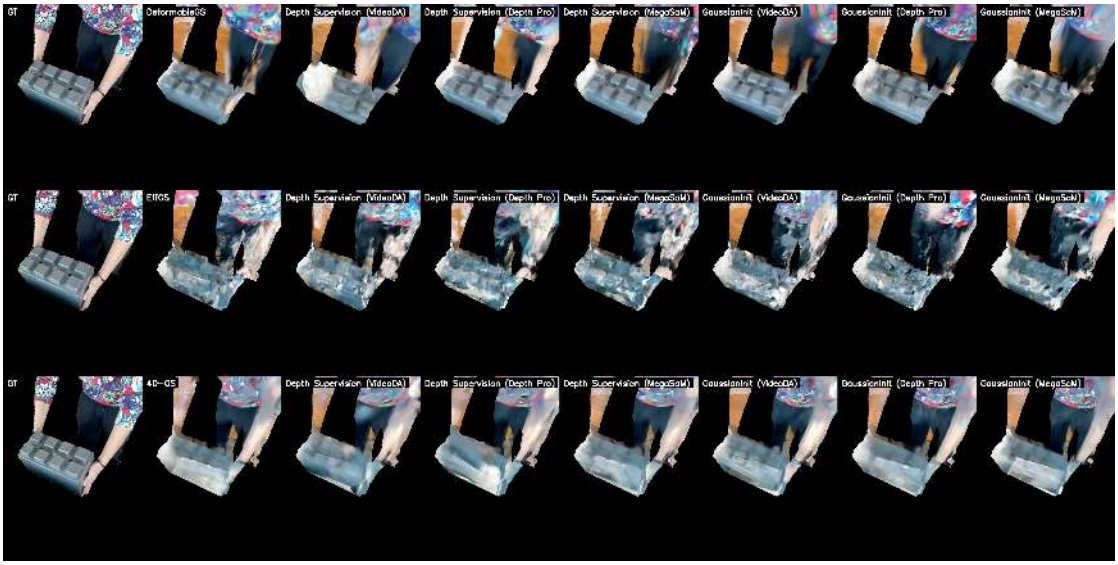
In fact, for these scenes, while *EffGS* and *4D-GS* are able to capture the motion of the objects quite decently in the training views, they fail to generalize to novel viewpoints. As a result, their reconstructions tend to appear mostly static.



(a)



(b)



(c)

Figure 5.19: *Comparison of (a) quantitative foreground evaluated results using the LPIPS metric and qualitative reconstructions of DeformableGS, EffGS, and 4D-GS for the block scene from the iPhone dataset (top to bottom) on (b) the full image as well as (c) the masked foreground. For each scene, the following outputs are shown from left to right: RGB ground truth (GT), baseline reconstruction (DeformableGS), depth-supervised reconstructions (Depth Supervision) using Video Depth Anything, Depth Pro (and MegaSaM), and depth-based initialized reconstructions (Gaussian Init) using Video Depth Anything, Depth Pro (and MegaSaM). It can be seen that DeformableGS reconstructs the block accurately, but positions the person partially outside the frame.*

6 Conclusion

This thesis presented an extensive study of how incorporating depth priors influences Gaussian Splatting-based dynamic 3D reconstruction. It began with a detailed evaluation of state-of-the-art monocular depth estimation methods, where *Video Depth Anything*, *Depth Pro*, and *MegaSaM* emerged as the most suitable candidates for depth-based enhancement of the reconstruction pipeline. This evaluation not only benchmarked their accuracy on LiDAR ground truth data but also explored their capabilities of predicting temporal consistency depth maps.

Building upon these outcomes, the integration of depth-based supervision into dynamic 3D reconstruction pipelines showed that depth priors can slightly enhance reconstruction quality for *EffGS* and *4D-GS*, by consistently outperforming their respective baselines on nearly all metrics. These improvements were especially pronounced in dynamic regions, where current dynamic reconstruction models often struggle to perform effectively.

Thereafter, we investigated depth-based Gaussian initialization as a second mechanism for injecting depth priors. Results indicate that, for *EffGS* and *4D-GS*, the performance gains achieved through depth supervision can be improved even further. While the overall improvement for *EffGS* is roughly 0.3 dB, *4D-GS* excels this, achieving an overall enhancement of nearly 1 dB in PSNR. Other metrics yield similar results. Although the quantitative metrics suggest that *DeformableGS* performs the worst, most probably due to its scene drifting issue, they do not show the full picture. When it comes to predicting motion from novel viewpoints, it massively outperforms both *EffGS* and *4D-GS*, demonstrating more accurate reconstruction of the dynamic foreground motion. Moreover, the experiments also reveal that these benefits are far from universally valid. In fact, the impact of depth priors is highly dependent on the underlying motion model, the dataset, and the nature of the depth prediction itself. These findings underscore that depth integration is not a one-size-fits-all solution, but rather requires careful and model-specific assessment considering both the prior quality and the model's representation of motion.

Several interesting directions for future work exist. The most natural extension would be to expand the framework to additional dynamic 3D reconstruction datasets, such as NeRF-DS or HyperNeRF, in order to improve the generality of the findings. Additionally, a more fine-grained analysis of the depth-based initialization procedure could be performed, including ablations of the warm-up phase, Gaussian coloring, scale parameters, and motion model weight initialization. Such studies would provide deeper insights by disentangling the contribution of each component.

Newly released methods such as *Depth Anything 3* (Lin et al., 2025), which predicts parameters for pixel-aligned 3D Gaussian primitives, also present an interesting opportunity to ease the initialization procedure. Moreover, combining additional monocular priors, such as depth and optical flow, could also act as a regularizer, potentially enhancing performance in synthesizing novel views.

Bibliography

Baranowski, Bartłomiej et al. (2026). "ConeGS: Error-Guided Densification Using Pixel Cones for Improved Reconstruction with Fewer Primitives". In: *2026 International Conference on 3D Vision (3DV)*.

Bochkovskii, Aleksei et al. (2025). "Depth Pro: Sharp Monocular Metric Depth in Less Than a Second". In: *International Conference on Learning Representations*. URL: https://arxiv.org/abs/2410.02073.

Cao, Ang and Justin Johnson (2023). "Hexplane: A Fast Representation for Dynamic Scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 130–141.

Chen, Sili et al. (2025). "Video Depth Anything: Consistent Depth Estimation for Super-Long Videos". In: *arXiv:2501.12375*.

Fang, Jiemin et al. (2022). "Fast Dynamic Radiance Fields with Time-Aware Neural Voxels". In: *SIGGRAPH Asia 2022 Conference Papers*.

Fridovich-Keil, Sara et al. (2022). "Plenoxels: Radiance Fields Without Neural Networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5501–5510.

Fridovich-Keil, Sara et al. (2023). "K-planes: Explicit radiance fields in space, time, and appearance". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12479–12488.

Gao, Hang et al. (2022). "Dynamic Novel-View Synthesis: A Reality Check". In: *NeurIPS*.

Glorot, Xavier and Yoshua Bengio (May 2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Horn, Berthold and Michael Brooks (Jan. 1989). *Shape from Shading*. Vol. 2. MIT Press.

Jiawei, Xu et al. (2025). "AD-GS: Object-Aware B-Spline Gaussian Splatting for Self-Supervised Autonomous Driving". In: *International Conference on Computer Vision*.

Katsumata, Kai, Duc Minh Vo, and Hideki Nakayama (2024). "A Compact Dynamic 3D Gaussian Representation for Real-Time Dynamic View Synthesis". In: *European Conference on Computer Vision*. Springer, pp. 394–412.

Kerbl, Bernhard et al. (2023). "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4. URL: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

Kopanas, Georgios et al. (2021). "Point-Based Neural Rendering with Per-View Optimization". In: *Computer Graphics Forum*. Vol. 40. 4. Wiley Online Library, pp. 29–43.

Li, Zhan et al. (2023). "Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis". In: *arXiv preprint arXiv:2312.16812*.

Li, Zhengqi et al. (2024). "MegaSaM: Accurate, Fast and Robust Structure and Motion from Casual Dynamic Videos". In: *arxiv*.

Liang, Yiqing et al. (2025). "Monocular Dynamic Gaussian Splatting: Fast, Brittle, and Scene Complexity Rules". In: *Transactions on Machine Learning Research*. Survey Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=fzmw8Joug4.

Lin, Haotong et al. (2025). "Depth Anything 3: recovering the visual space from any views". In: *arXiv preprint arXiv:2511.10647*.

Luiten, Jonathon et al. (2024). "Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis". In: *3DV*.

Mildenhall, Ben et al. (2020). "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*.

Müller, Thomas et al. (2022). "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM transactions on graphics (TOG)* 41.4, pp. 1–15.

Oquab, Maxime et al. (2023). *DINOv2: Learning Robust Visual Features without Supervision*.

Park, Keunhong et al. (Dec. 2021a). "HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields". In: *ACM Trans. Graph.* 40.6.

Park, Keunhong et al. (2021b). "Nerfies: Deformable Neural Radiance Fields". In: *ICCV*.

Piccinelli, Luigi et al. (2024). "UniDepth: Universal Monocular Metric Depth Estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Piccinelli, Luigi et al. (2025). *UniDepthV2: Universal Monocular Metric Depth Estimation Made Simpler*. arXiv: 2502.20110 [cs.CV]. URL: https://arxiv.org/abs/2502.20110.

Pumarola, Albert et al. (June 2021). "D-NeRF: Neural Radiance Fields for Dynamic Scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10318–10327.

Ranftl, René et al. (2022). "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.3.

Schönberger, Johannes Lutz and Jan-Michael Frahm (2016a). "Structure-From-Motion Revisited". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schönberger, Johannes Lutz et al. (2016b). "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*.

Suhoi, Daniil (July 2024). *Metric and Relative Monocular Depth Estimation: An Overview. Fine-Tuning Depth Anything V2*. https://huggingface.co/blog/Isayoften/monocular-depth-estimation-guide. Hugging Face Blog. Last accessed: 2025-11-20.

Teed, Zachary and Jia Deng (2020). "Raft: Recurrent All-Pairs Field Transforms for Optical Flow". In: *European conference on computer vision*. Springer, pp. 402–419.

Teed, Zachary and Jia Deng (2021). "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras". In: *Advances in neural information processing systems*.

Tomasi, Carlo and Takeo Kanade (1992). "Shape and motion from image streams under orthography: a factorization method". In: *International journal of computer vision* 9.2, pp. 137–154.

Wang, Qianqian et al. (2025). "Shape of Motion: 4D Reconstruction from a Single Video". In: *International Conference on Computer Vision (ICCV)*.

Wang, Ruicheng et al. (2025). "Moge: Unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision". In: *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 5261–5271.

Wang, Zhou, Eero P Simoncelli, and Alan C Bovik (2003). "Multiscale structural similarity for image quality assessment". In: *The thrity-seventh asilomar conference on signals, systems & computers, 2003*. Vol. 2. Ieee, pp. 1398–1402.

Wang, Zhou et al. (2004). "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4, pp. 600–612.

Wu, Guanjun et al. (2024). "4D Gaussian Splatting for Real-Time Dynamic Scene Rendering". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 20310–20320.

Yan, Zhiwen, Chen Li, and Gim Hee Lee (2023). "NeRF-DS: Neural Radiance Fields for Dynamic Specular Objects". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8285–8295.

Yang, Lihe et al. (2024a). "Depth Anything V2". In: *arXiv:2406.09414*.

Yang, Lihe et al. (2024b). "Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data". In: *CVPR*.

Yang, Zeyu et al. (2024). "Real-time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting". In: *International Conference on Learning Representations (ICLR)*.

Yang, Ziyi et al. (2024). "Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 20331–20341.

Ye, Vickie et al. (2025). "gsplat: An open-source library for Gaussian splatting". In: *Journal of Machine Learning Research* 26.34, pp. 1–17.

Yifan, Wang et al. (2019). "Differentiable surface splatting for point-based geometry processing". In: *ACM Transactions On Graphics (TOG)* 38.6, pp. 1–14.

Yu, Zehao et al. (2022). "MonoSDF: Exploring Monocular Geometric Cues for Neural Implicit Surface Reconstruction". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, Richard et al. (2018). "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595.

Zwicker, Matthias et al. (2001). "EWA Volume Splatting". In: *Proceedings Visualization, 2001. VIS'01.* IEEE, pp. 29–538.

Appendices

A

Additional Results: Evaluation of Depth Prediction Models

A.1 Quantitative Result Plots Per Image

A.1.1 Absolute Relative Error (AbsRel)

Foreground Evaluation

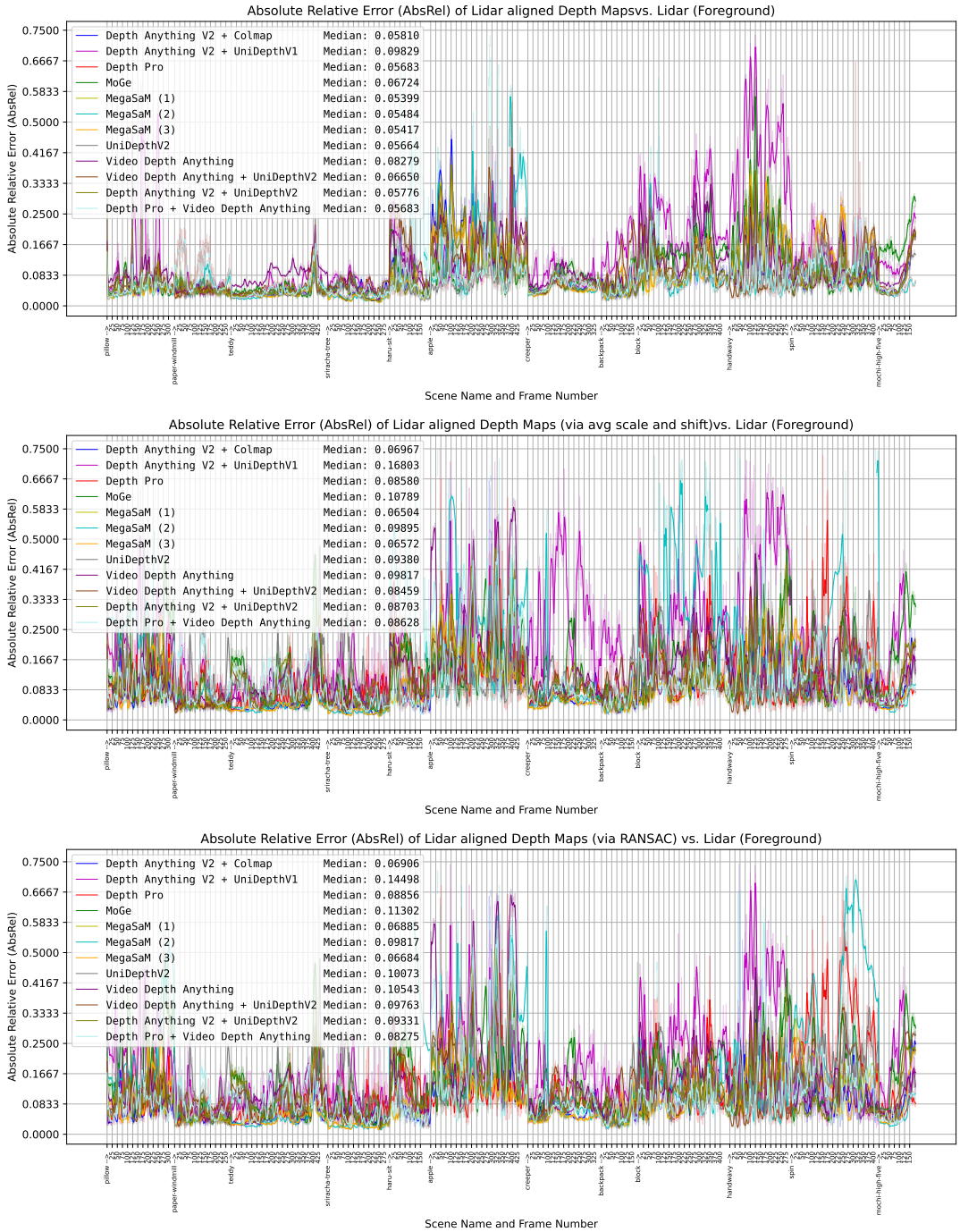


Figure A.1: *Gaussian-smoothed per-image Absolute Relative Error (AbsRel) on the foreground across the three alignment strategies: per-image median-based (top), per-scene median-based using the mean scale and shift (middle), and RANSAC (bottom). It is notable that per-scene alignment strategies have generally higher AbsRel errors compared to the per-image approach.*

Full-Image Evaluation

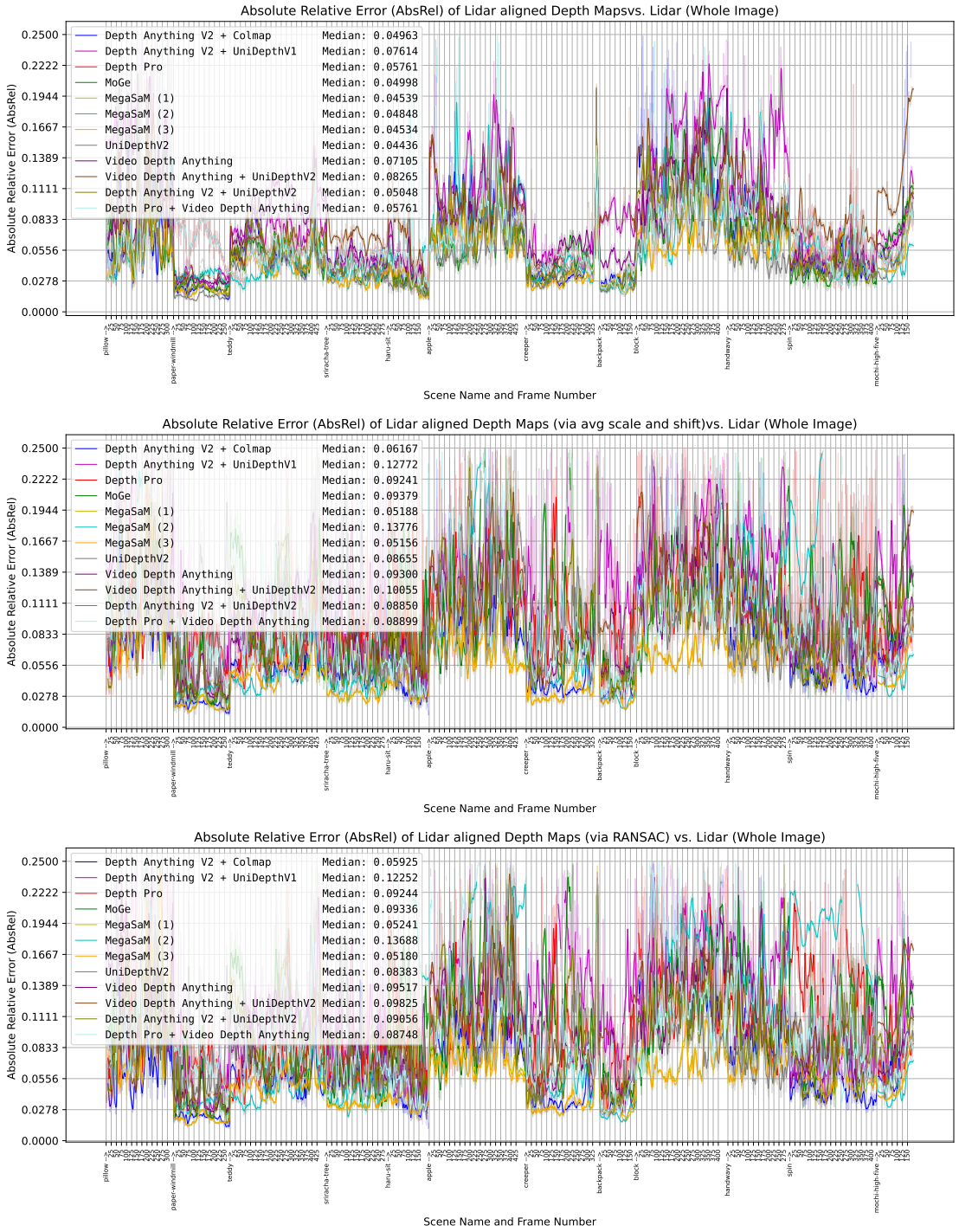


Figure A.2: *Gaussian-smoothed per-image Absolute Relative Error (AbsRel) on the whole image across the three alignment strategies: per-image median-based (top), per-scene median-based using the mean scale and shift (middle), and RANSAC (bottom). The top plot highlights that certain scenes, such as* paper-windmill*, are easier for all methods than more challenging scenes like* block *or* handwavy*. In contrast, the two bottom plots show the strong performance of MegaSaM (1) and (2) (yellow and light green).*

A.2 Per Image Mean Squared Error (MSE)

A.2.1 Foreground Evaluation

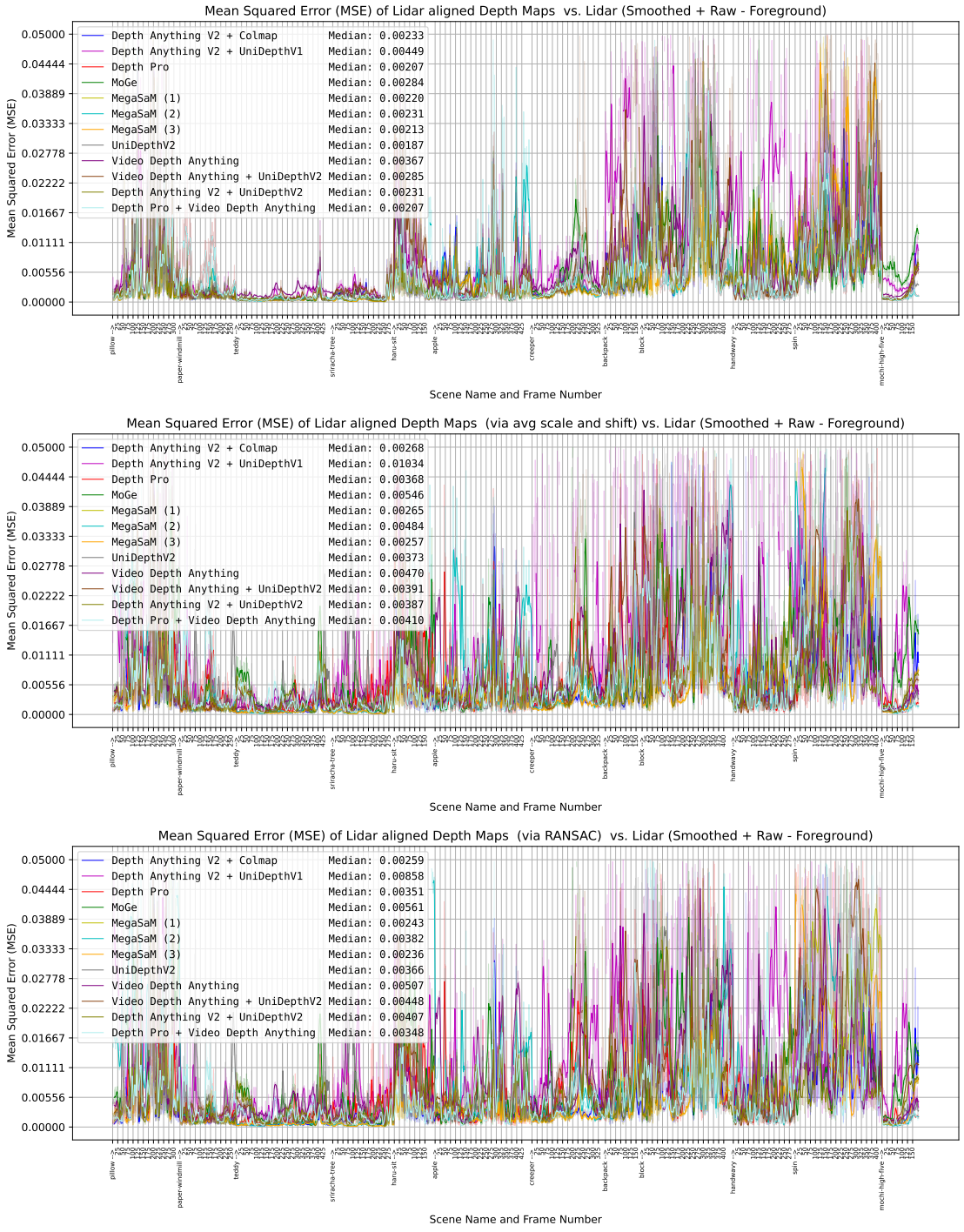


Figure A.3: *Gaussian-smoothed per-image Mean Squared Error (MSE) on the foreground across the three alignment strategies: per-image median-based (top), per-scene median-based using the mean scale and shift (middle), and RANSAC (bottom). Scenes captured at larger distances, such as* spin*, tend to have larger errors than more compact scenes like* apple *and* mochi-high-five *due to the non-relative nature of the MSE metric.*

A.2.2 Full-Image Evaluation

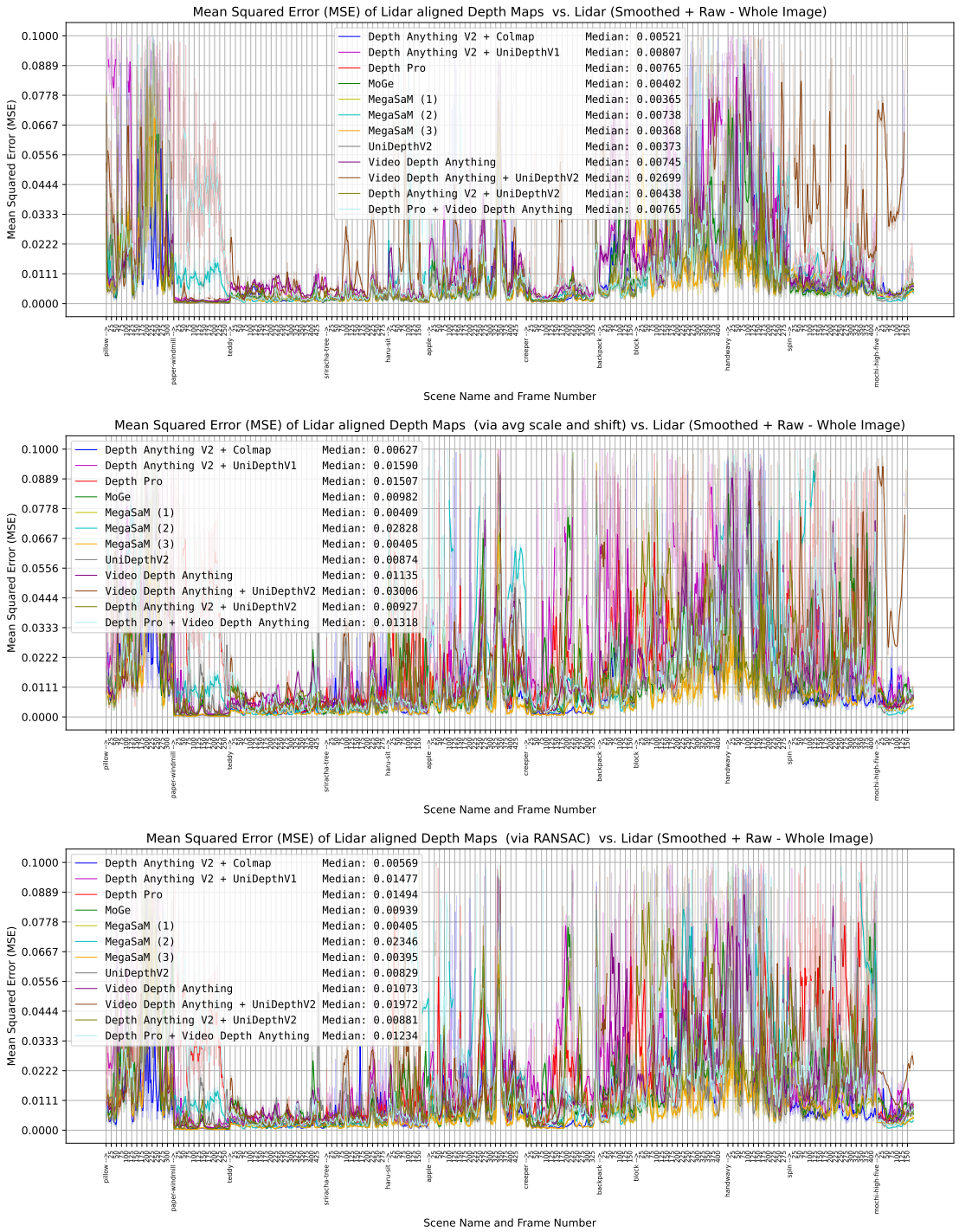


Figure A.4: *Gaussian-smoothed per-image Mean Squared Error (MSE) on the whole image across the three alignment strategies: per-image median-based (top), per-scene median-based using the mean scale and shift (middle), and RANSAC (bottom). Depth prediction methods especially struggle to predict temporal consistent depths for the* creeper *scene, where the MSE noticeably increases when using scene-level alignment (top vs bottom two plots).*

A.3 Per Method Median Absolute Deviation

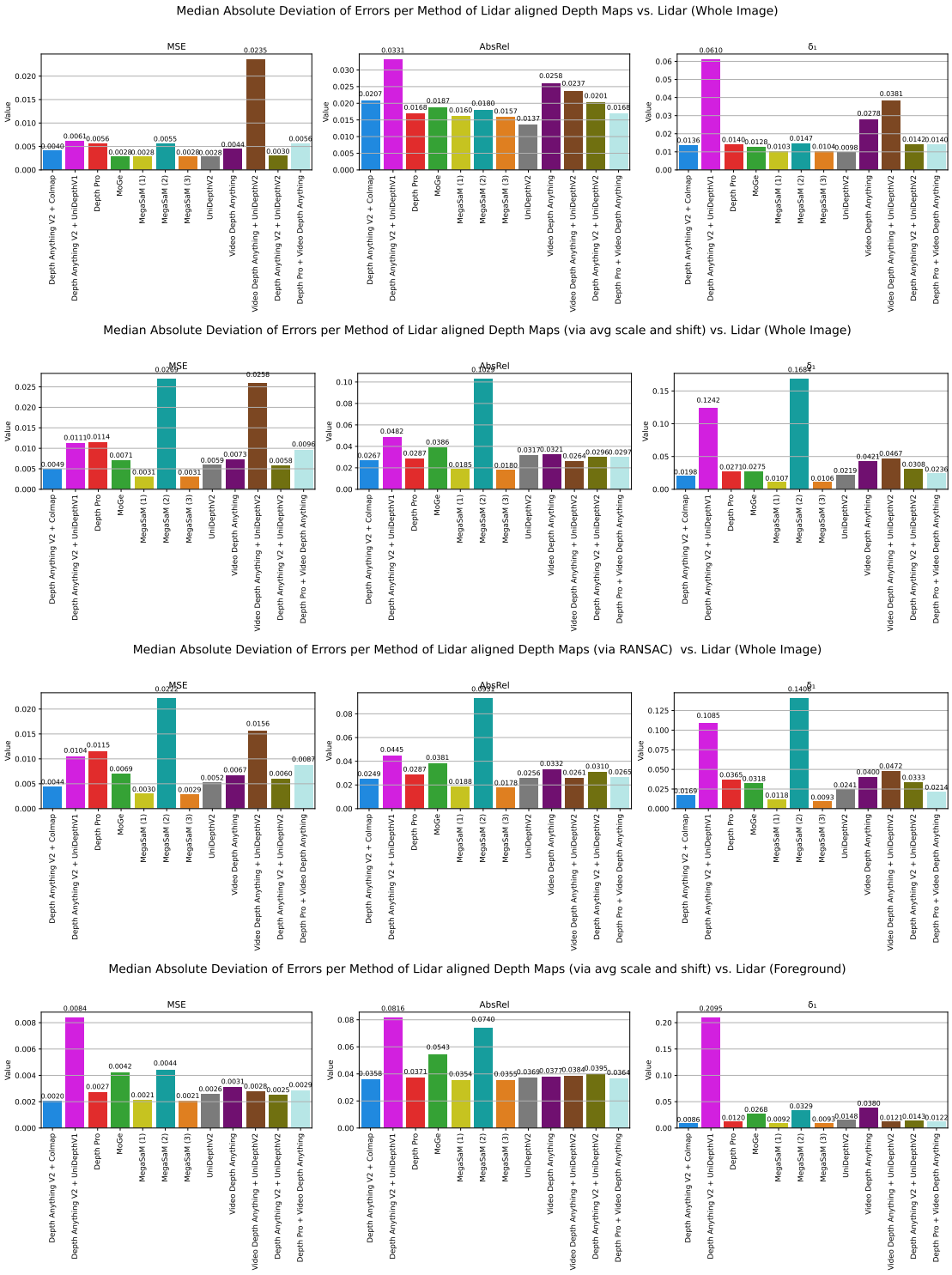


Figure A.5: *Median Absolute Deviation of per method errors on the whole image across the three alignment strategies: per-image median-based (top), per-scene median-based using the mean scale and shift (upper-middle), and RANSAC (lower-middle); per-scene median-based using the mean scale and shift alignment on the foreground (bottom). Notably, MegaSaM (2) has consistently high median deviation across all metrics when using per-scene alignment.*

A.4 Per Scene Mean Squared Error (MSE) Boxplots

A.4.1 Per Image Median-Based Alignment

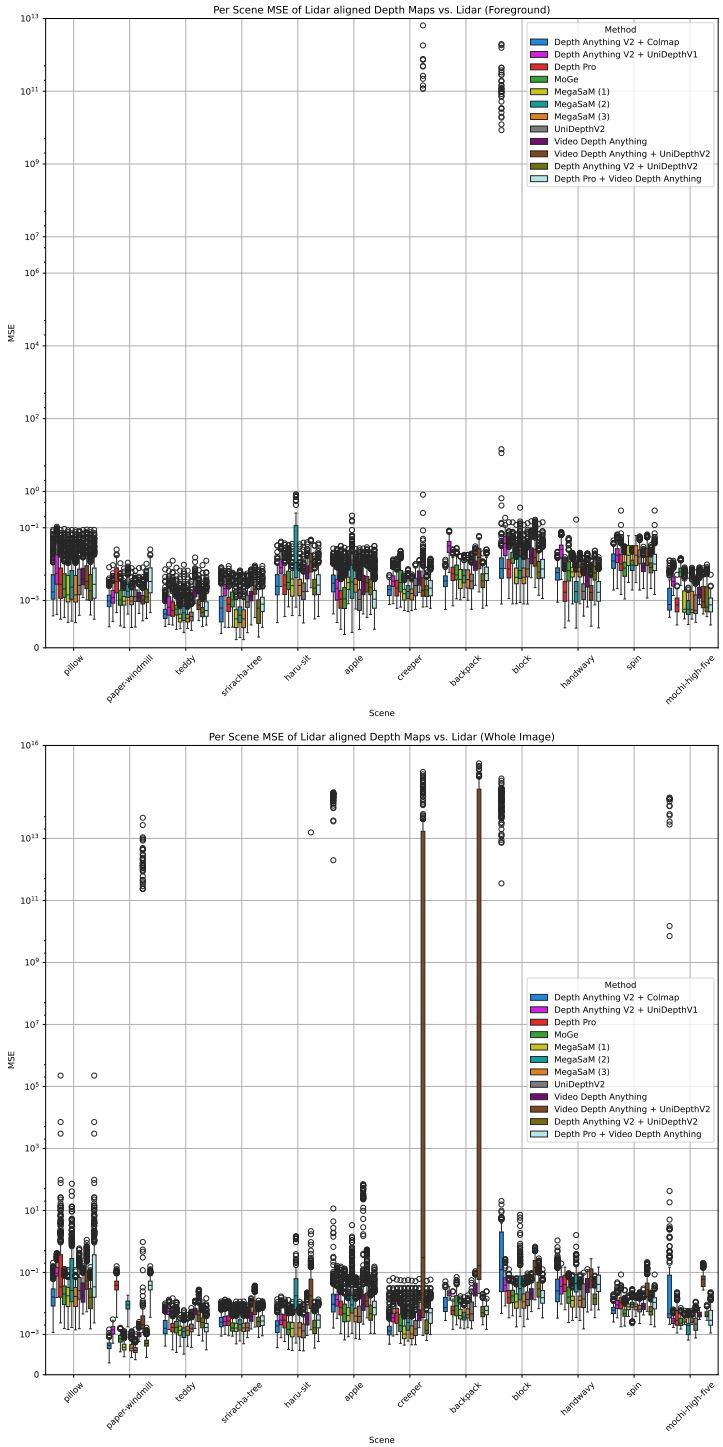


Figure A.6: *Per-scene Mean Squared Error (MSE) boxplots using per-image median-based alignment, shown for the foreground (top) and for the whole image (bottom). It is evident from the plots that nearly all methods generate fewer outliers when predicting the foreground (top) compared to the whole image (bottom).*

A.4.2 Per Scene Median-Based Alignment

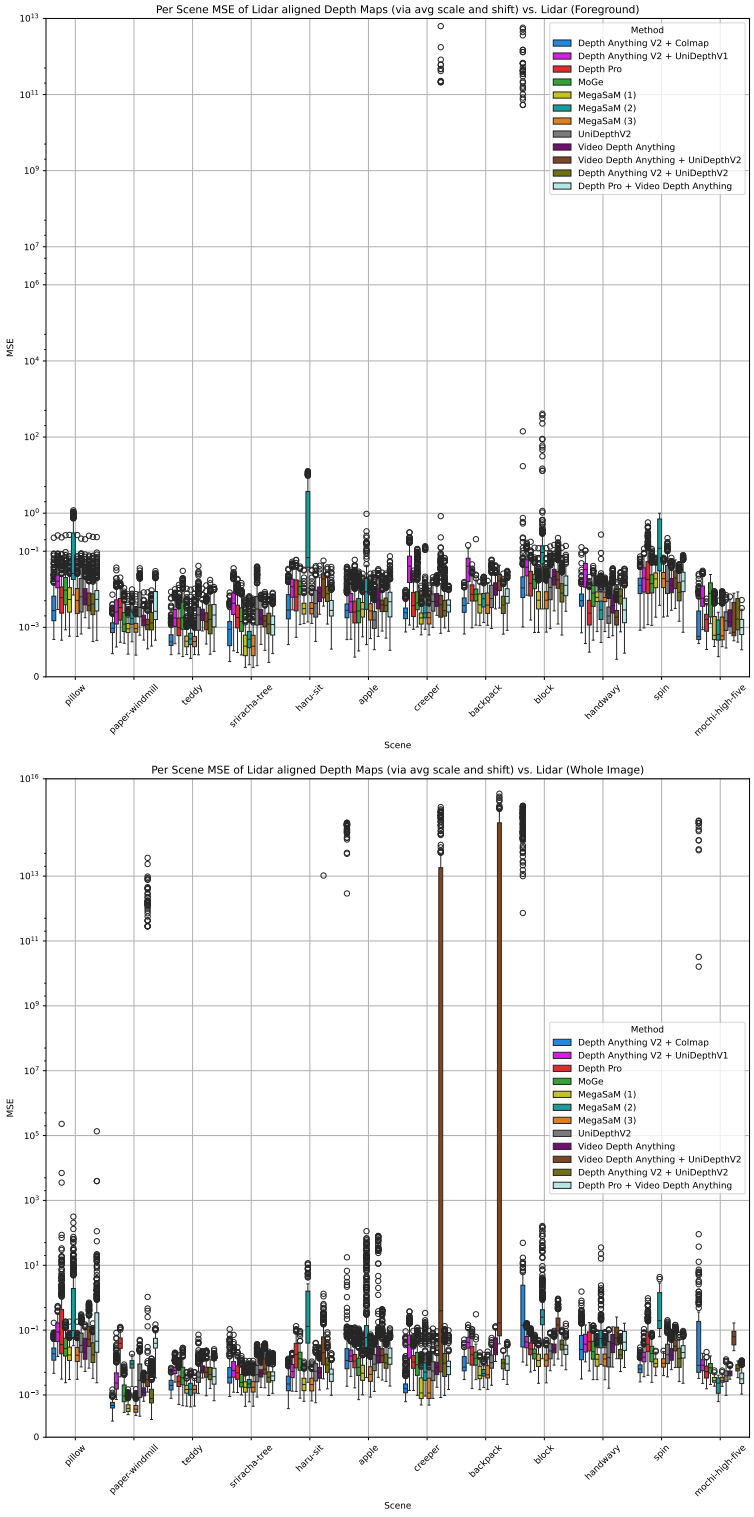


Figure A.7: *Per-scene Mean Squared Error (MSE) boxplots using per-scene median-based alignment using the mean scale and shift, shown for the foreground (top) and for the whole image (bottom). Across all metrics, MegaSaM (1) (yellow) and (2) (orange) excel with very small interquartile ranges, indicating consistently low variance.*

A.4.3 Per Scene RANSAC-Based Alignment

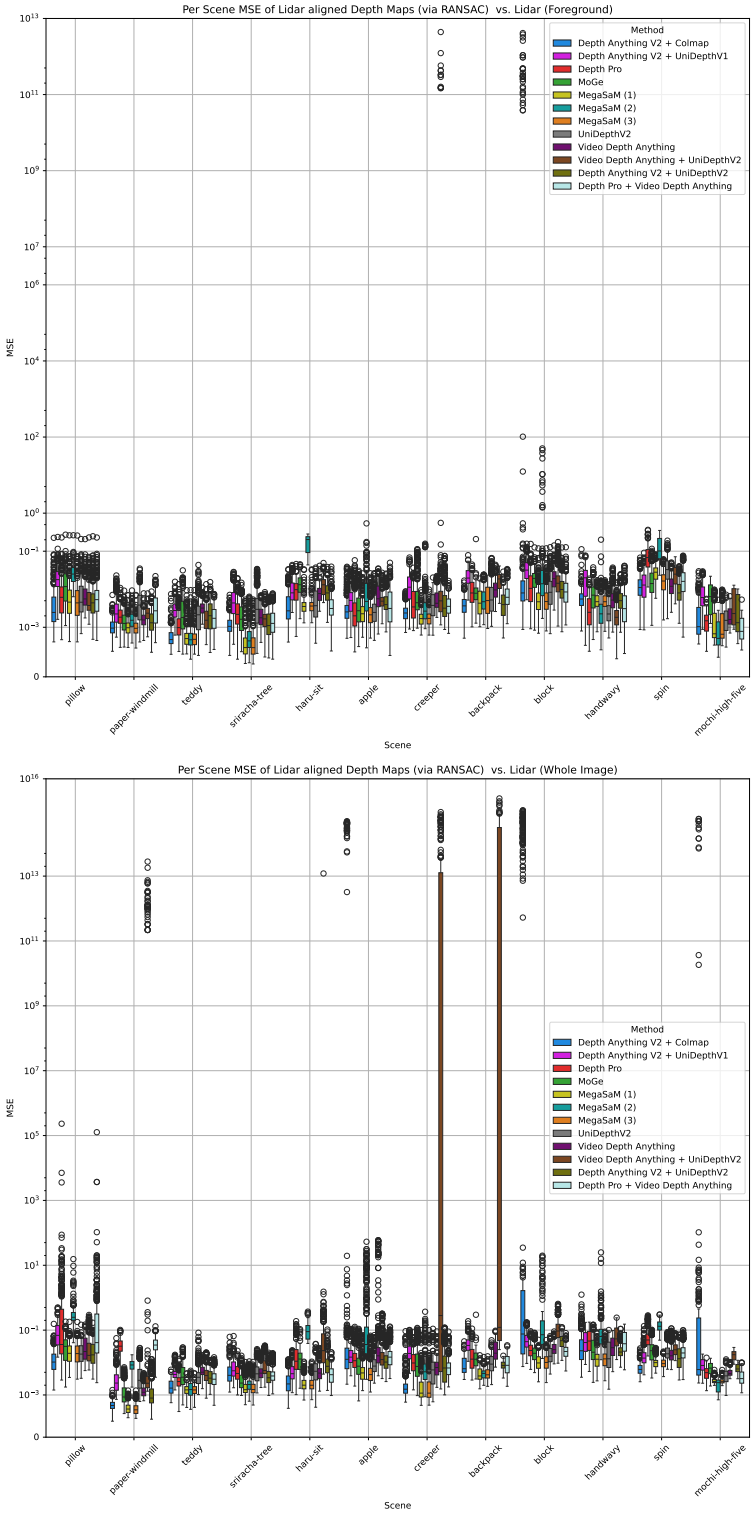


Figure A.8: *Per-scene Mean Squared Error (MSE) boxplots using per-scene RANSAC alignment, shown for the foreground (top) and for the whole image (bottom). Similar to boxplots for other alignment strategies, Video Depth Anything + UniDepthV2 exhibits very high third quartiles on the whole image in some scenes (bottom), suggesting a large number of substantial outliers.*

B

Additional Results: Depth-Based Supervision

B.1 Quantitative Results Per Dataset

B.1.1 Nerfies

Table B.1: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across the Nerfies dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	21.72	0.484	0.729	0.416	9.5	10197
DeformableGS + Depth Supervision (VideoDA)	**22.27**	0.487	**0.739**	0.452	9.7	10054
DeformableGS + Depth Supervision (Depth Pro)	21.97	0.478	0.727	**0.391**	10.1	10539
EffGS	21.12	0.480	0.695	0.505	**91.4**	**2722**
EffGS + Depth Supervision (VideoDA)	21.66	0.499	0.717	0.479	84.5	3091
EffGS + Depth Supervision (Depth Pro)	21.62	0.500	0.716	0.473	87.9	3142
4D-GS	21.38	**0.504**	0.721	0.486	37.7	6360
4D-GS + Depth Supervision (VideoDA)	21.50	0.501	0.716	0.490	32.5	6726
4D-GS + Depth Supervision (Depth Pro)	21.66	0.504	0.720	0.494	33.9	7353

Table B.2: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the Nerfies dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	20.63	0.499	0.674	0.451	9.6	10197
DeformableGS + Depth Supervision (VideoDA)	**21.44**	**0.517**	0.695	0.452	11.8	10054
DeformableGS + Depth Supervision (Depth Pro)	21.29	0.510	**0.695**	**0.440**	10.5	10539
EffGS	19.27	0.470	0.605	0.517	**91.9**	**2722**
EffGS + Depth Supervision (VideoDA)	19.78	0.484	0.622	0.502	89.1	3091
EffGS + Depth Supervision (Depth Pro)	19.75	0.480	0.620	0.493	90.0	3142
4D-GS	19.32	0.484	0.615	0.518	39.9	6360
4D-GS + Depth Supervision (VideoDA)	19.44	0.487	0.615	0.510	33.7	6726
4D-GS + Depth Supervision (Depth Pro)	19.69	0.492	0.627	0.500	36.3	7353

B.1.2 iPhone

Table B.3: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across the iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	18.78	0.547	0.624	0.324	11.0	9986
DeformableGS + Depth Supervision (VideoDA)	18.39	0.535	0.595	0.403	14.4	8868
DeformableGS + Depth Supervision (Depth Pro)	18.57	0.533	0.606	0.381	14.5	8647
DeformableGS + Depth Supervision (MegaSaM)	18.00	0.505	0.583	0.464	15.3	7515
EffGS	20.25	0.581	0.663	0.307	99.4	**2131**
EffGS + Depth Supervision (VideoDA)	20.25	0.583	0.665	0.305	107.1	2493
EffGS + Depth Supervision (Depth Pro)	20.28	0.584	0.664	0.305	108.7	2246
EffGS + Depth Supervision (MegaSaM)	**20.34**	**0.585**	**0.666**	**0.304**	**109.6**	2325
4D-GS	18.62	0.570	0.639	0.320	35.9	7795
4D-GS + Depth Supervision (VideoDA)	18.97	0.577	0.652	0.304	35.9	8194
4D-GS + Depth Supervision (Depth Pro)	18.73	0.555	0.629	0.320	35.0	8302
4D-GS + Depth Supervision (MegaSaM)	19.00	0.568	0.645	0.306	35.4	8222

Table B.4: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	15.75	0.432	0.517	0.413	10.9	9986
DeformableGS + Depth Supervision (VideoDA)	15.40	0.419	0.482	0.493	14.9	8868
DeformableGS + Depth Supervision (Depth Pro)	15.71	0.434	0.512	0.443	14.6	8647
DeformableGS + Depth Supervision (MegaSaM)	15.35	0.420	0.486	0.543	15.5	7515
EffGS	17.14	0.461	0.556	0.376	102.8	**2131**
EffGS + Depth Supervision (VideoDA)	17.30	0.464	0.564	0.373	104.9	2493
EffGS + Depth Supervision (Depth Pro)	**17.36**	0.464	0.563	**0.371**	110.0	2246
EffGS + Depth Supervision (MegaSaM)	17.33	**0.469**	**0.567**	0.371	**110.7**	2325
4D-GS	15.40	0.444	0.514	0.431	36.1	7795
4D-GS + Depth Supervision (VideoDA)	15.86	0.457	0.529	0.411	35.8	8194
4D-GS + Depth Supervision (Depth Pro)	15.88	0.455	0.529	0.413	36.1	8302
4D-GS + Depth Supervision (MegaSaM)	16.00	0.461	0.537	0.410	36.5	8222

B.2 Per Metric Quantitative Result Plots

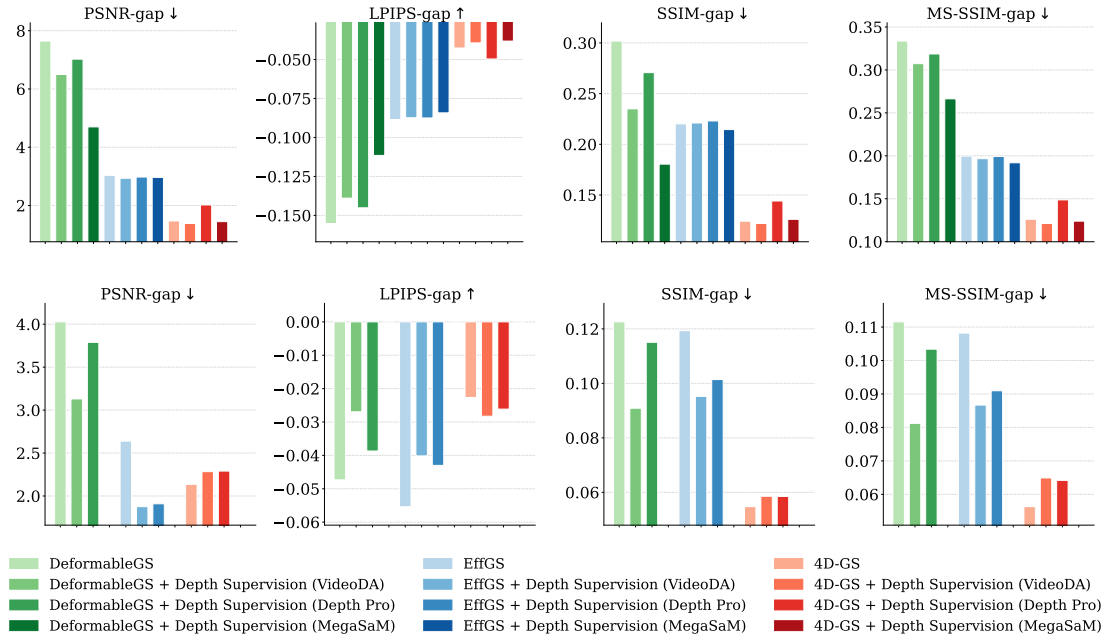


Figure B.1: *Train-test gaps for all metrics on the iPhone (top) and the Nerfies dataset (bottom), reported for the foreground image showing the limited regularizing effect of depth-based supervision across the different methods.*

B.3 Per Dataset Quantitative Result Plots

B.3.1 Foreground Evaluation

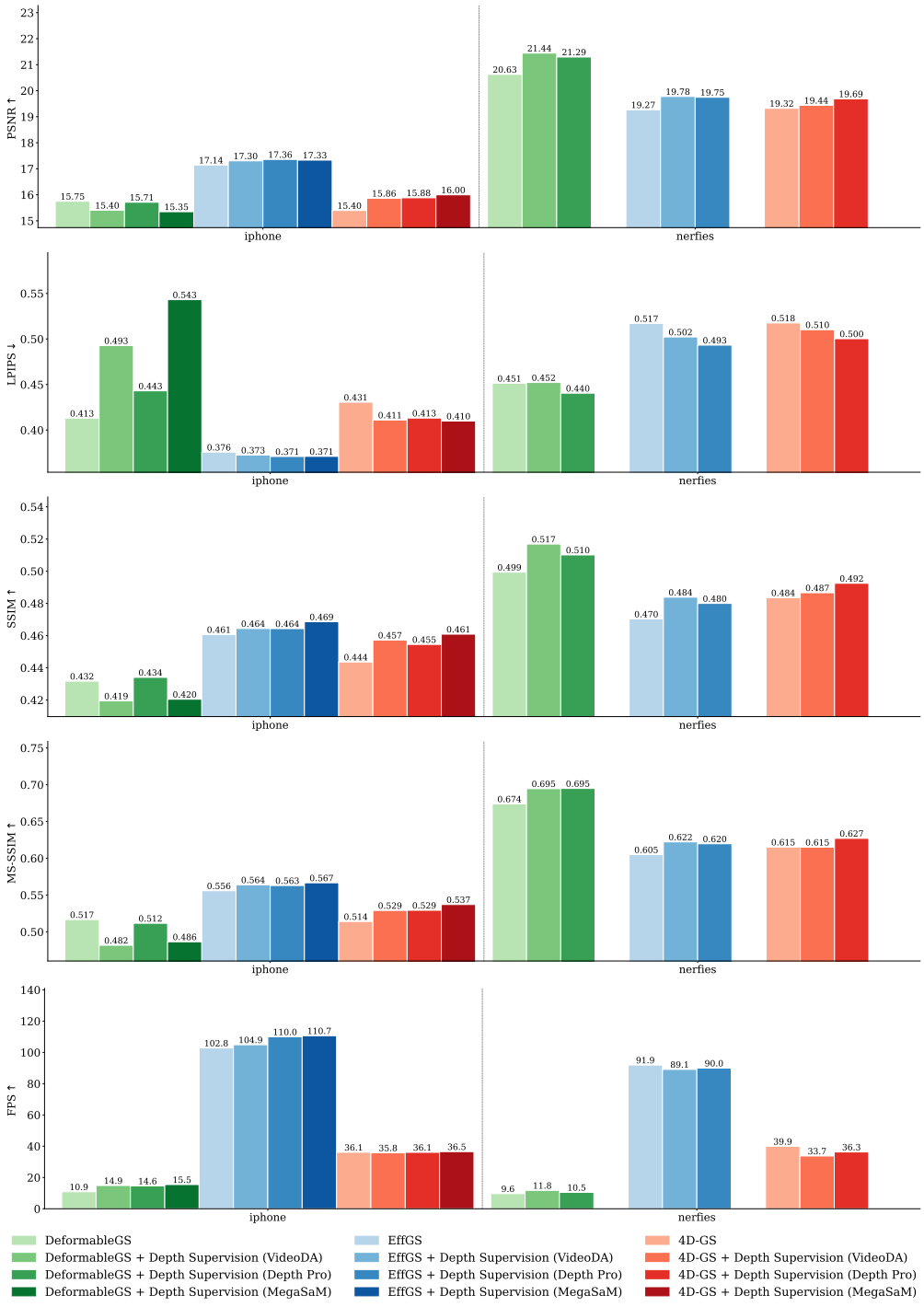


Figure B.2: *Per method quantitative results on the iPhone and the Nerfies datasets evaluated on the foreground. Corresponding numerical values are plotted above each bar. The results indicate that, with the exception of DeformableGS on the iPhone dataset, integrating depth-based supervision generally leads to improved reconstructions.*

B.4 Per Scene Quantitative Result Plots

B.4.1 Nerfies

Full-Image Evaluation

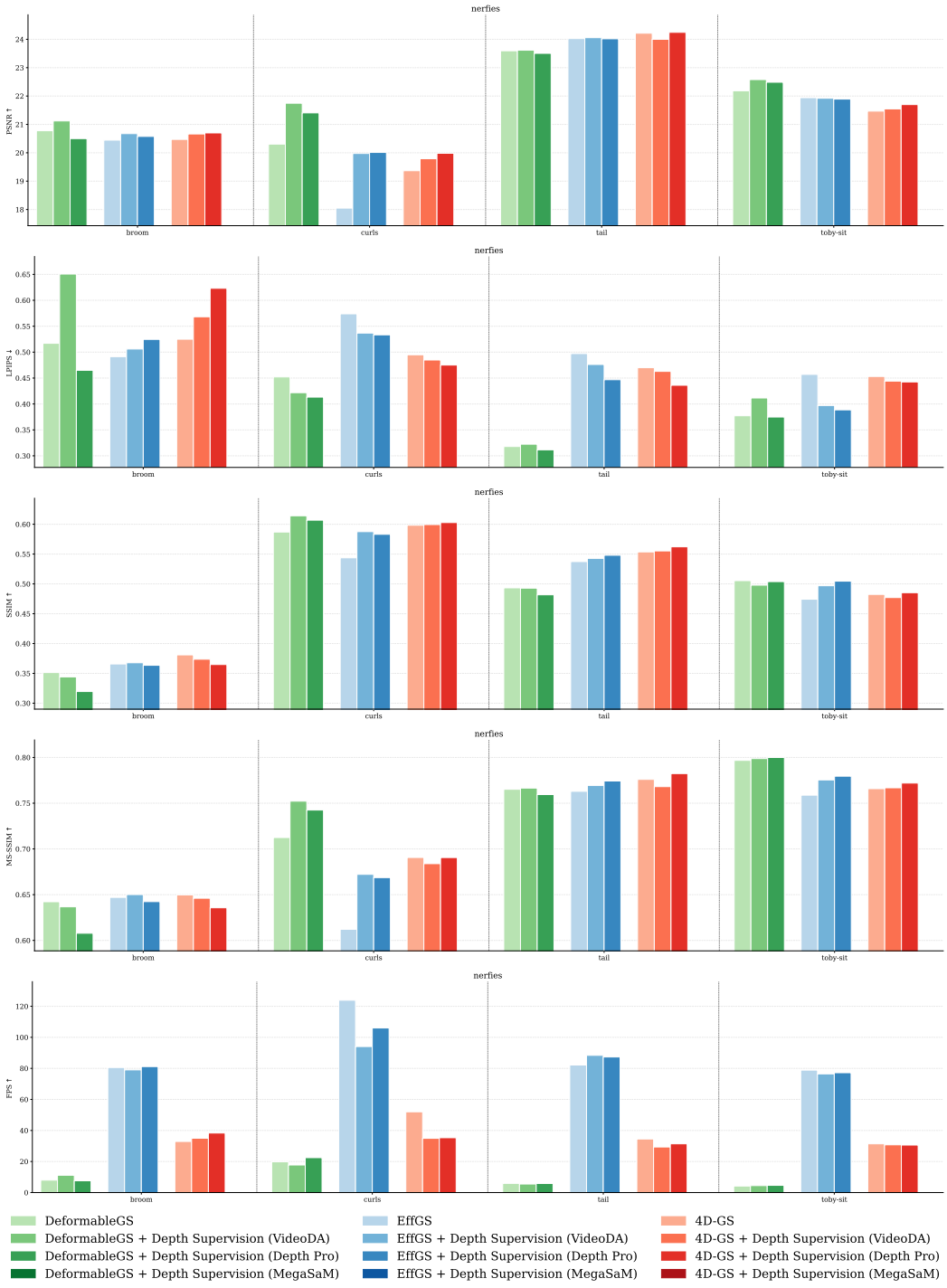


Figure B.3: ***Reconstruction Metrics.*** *Per scene quantitative results on the Nerfies dataset evaluated on the full image. It shows that all methods achieve a significant boost in performance on the curls scene when integrating depth-based supervision.*

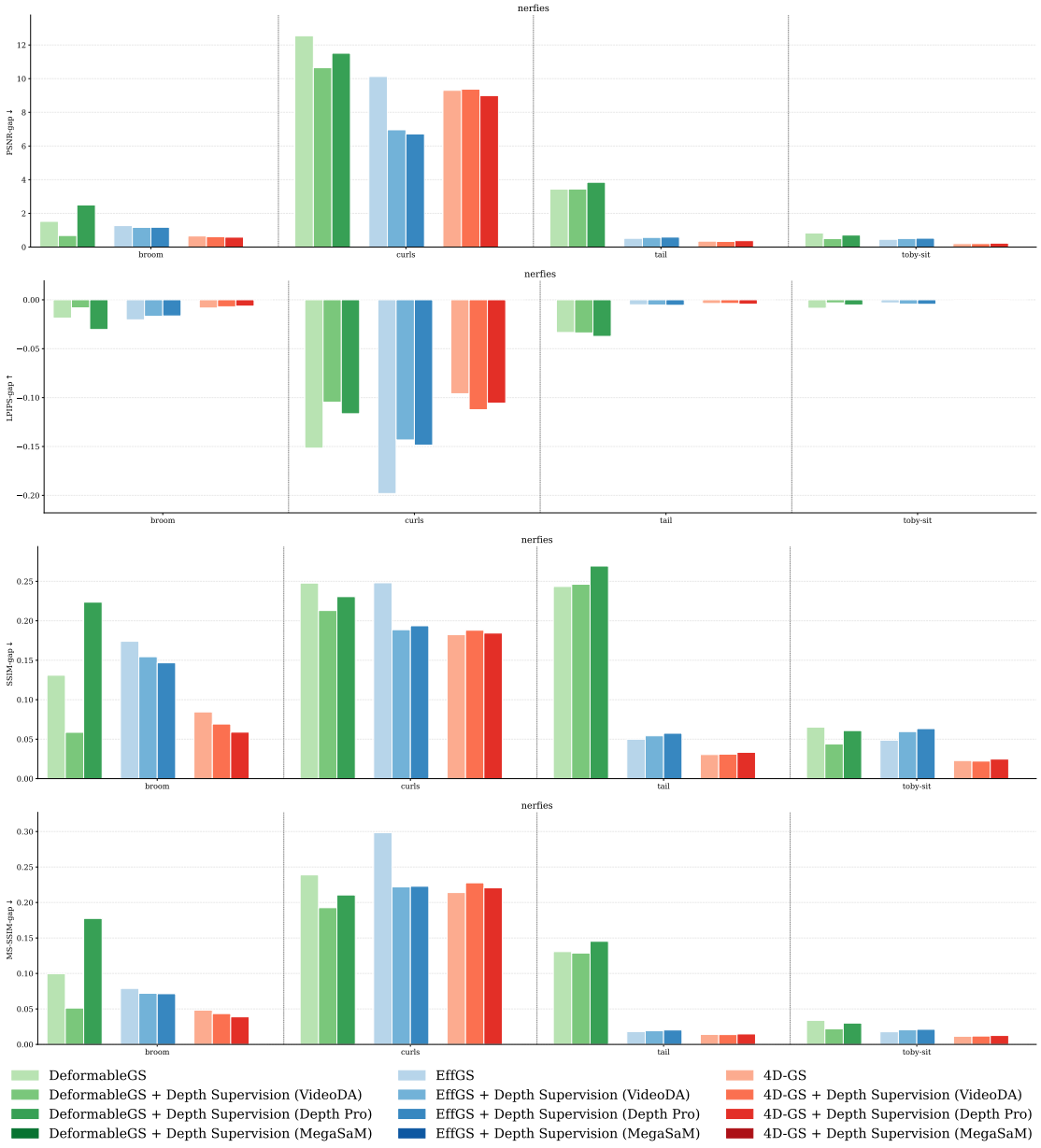


Figure B.4: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the Nerfies dataset evaluated on the full image. It is striking that all methods overfit on the curls scene, with PSNR gaps of up to* 12.

Foreground Evaluation

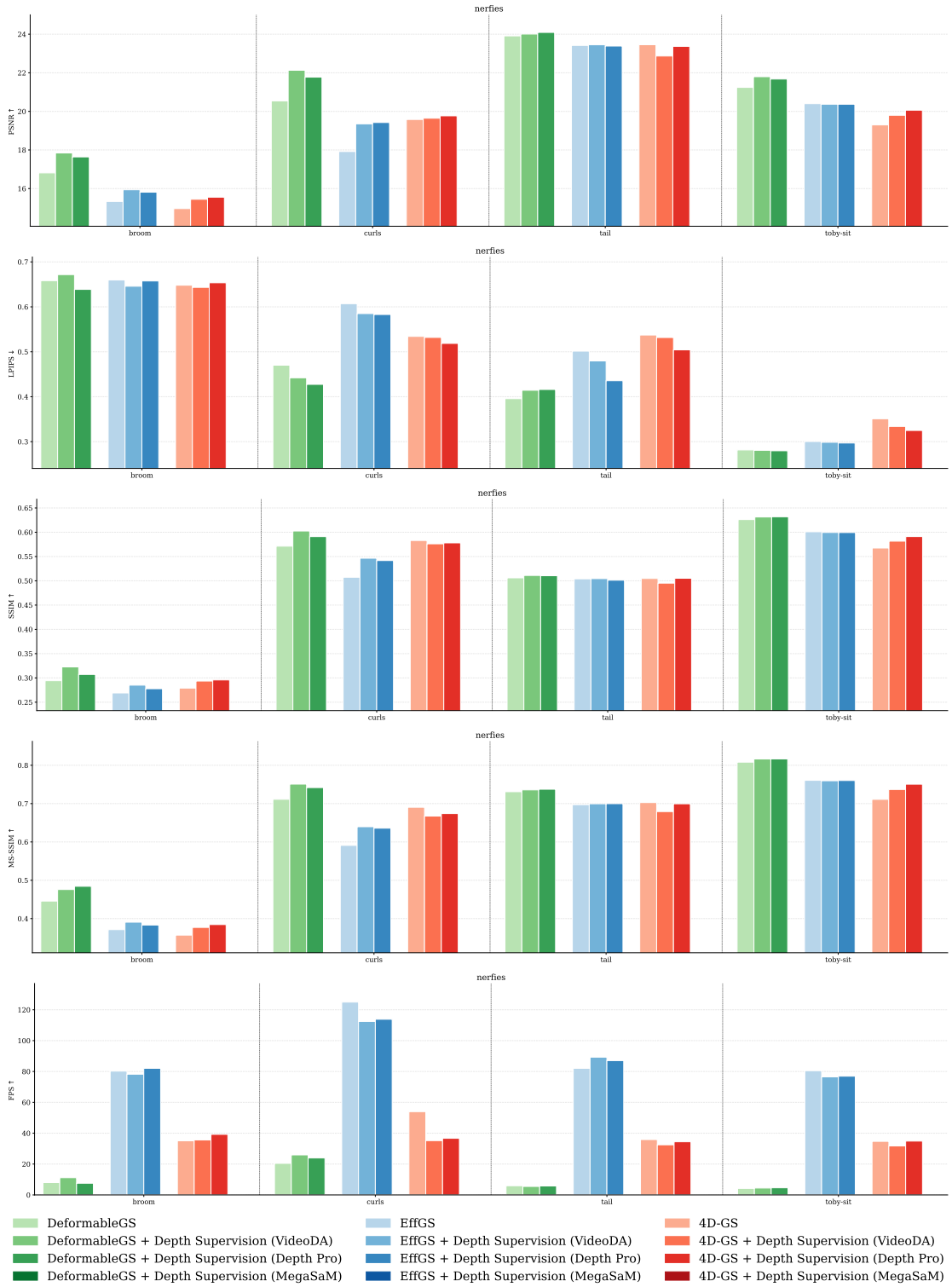


Figure B.5: ***Reconstruction Metrics.*** *Per scene quantitative results on the Nerfies dataset evaluated on the foreground. The results demonstrate that the broom scene has a particularly challenging foreground.*

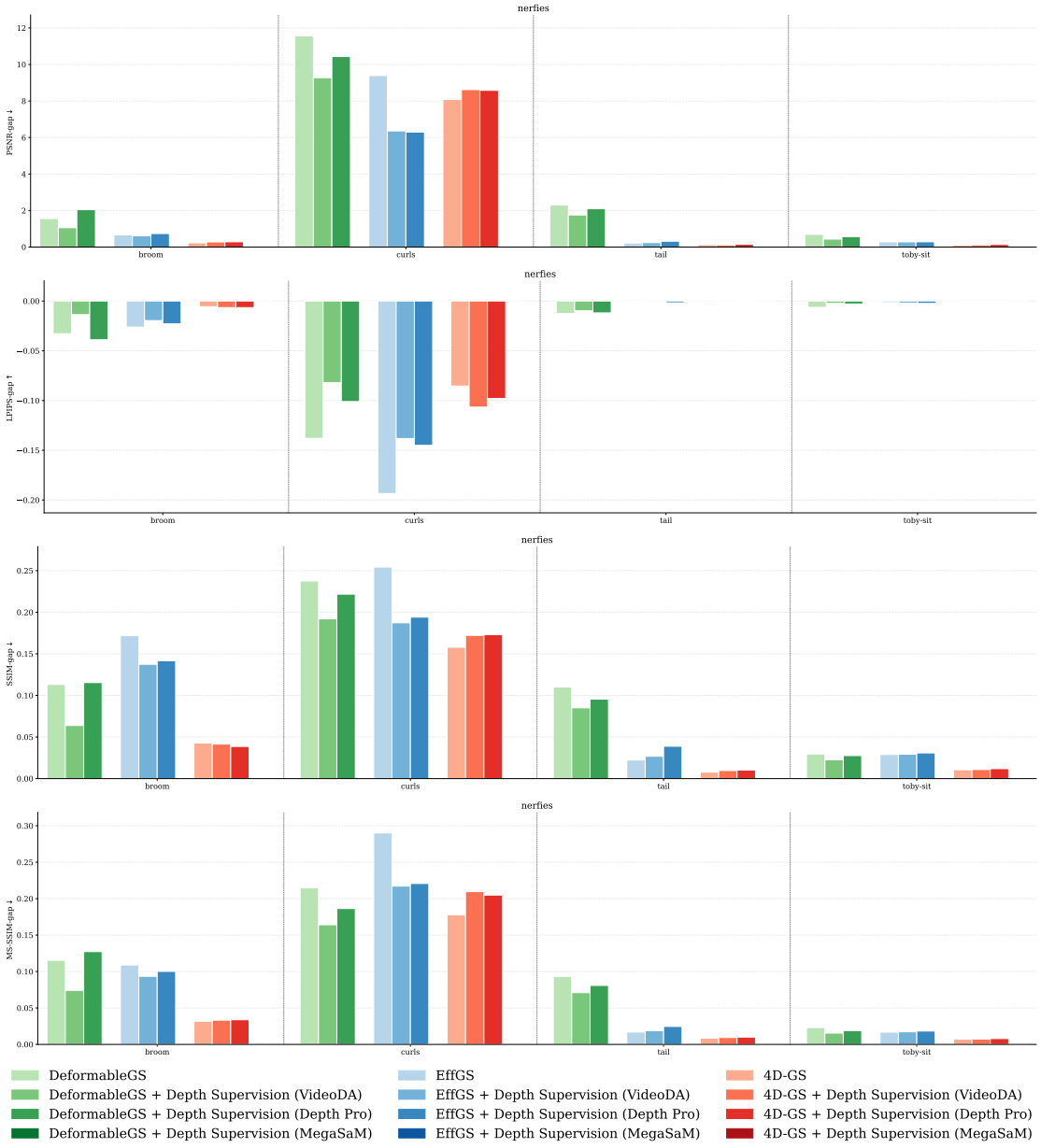


Figure B.6: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the Nerfies dataset evaluated on the foreground. Remarkably, the curls scene reports smaller gaps for EffGS, implying depth-based supervision improves regularization.*

B.4.2 iPhone

Full-Image Evaluation

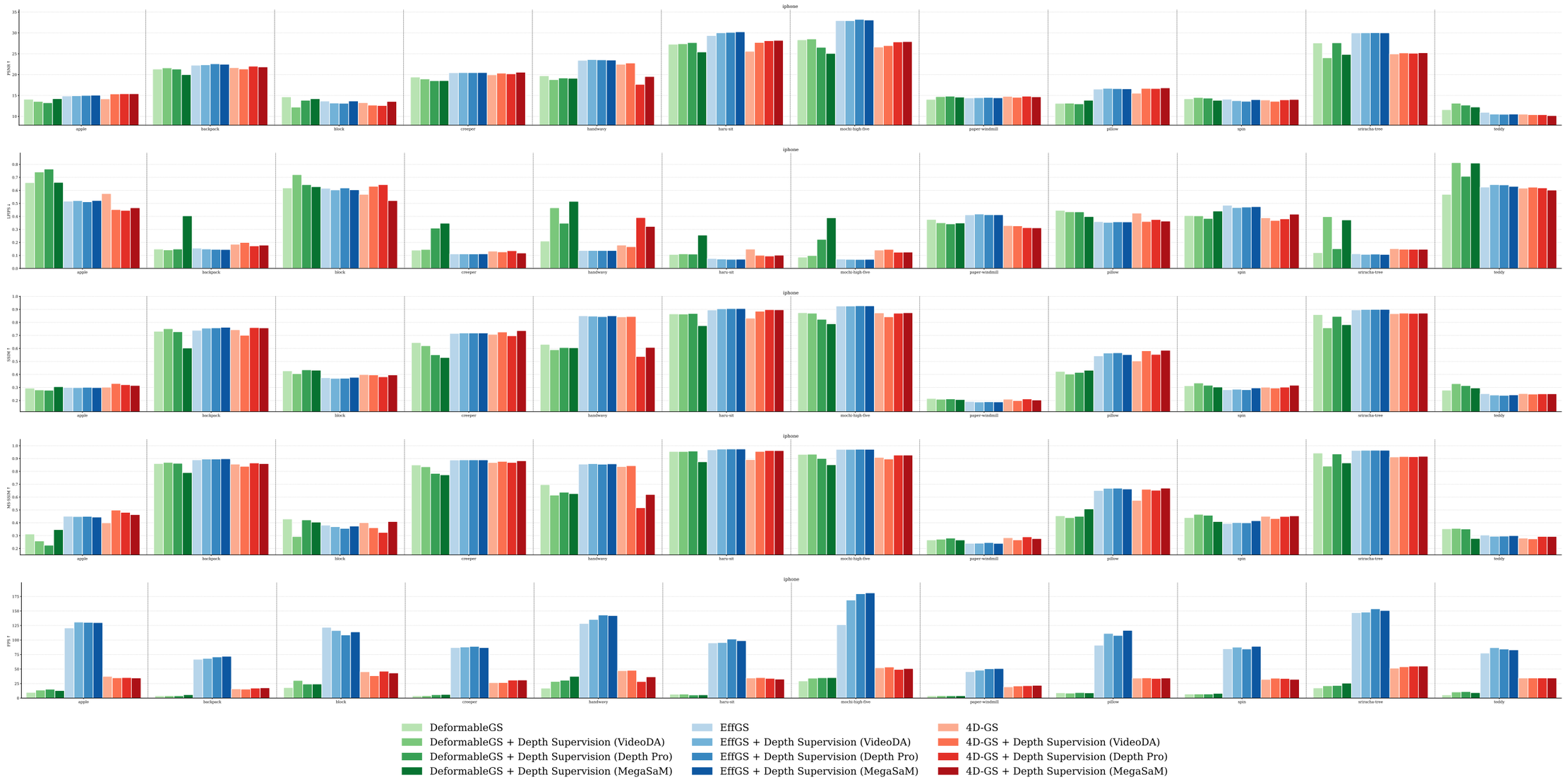


Figure B.7: ***Reconstruction Metrics.*** *Per scene quantitative results on the iPhone dataset evaluated on the full image. The results reveal that mochi-high-five is the easiest scene, achieving PSNR values above 25 dB for all methods, with EffGS reaching approximately 32 dB.*

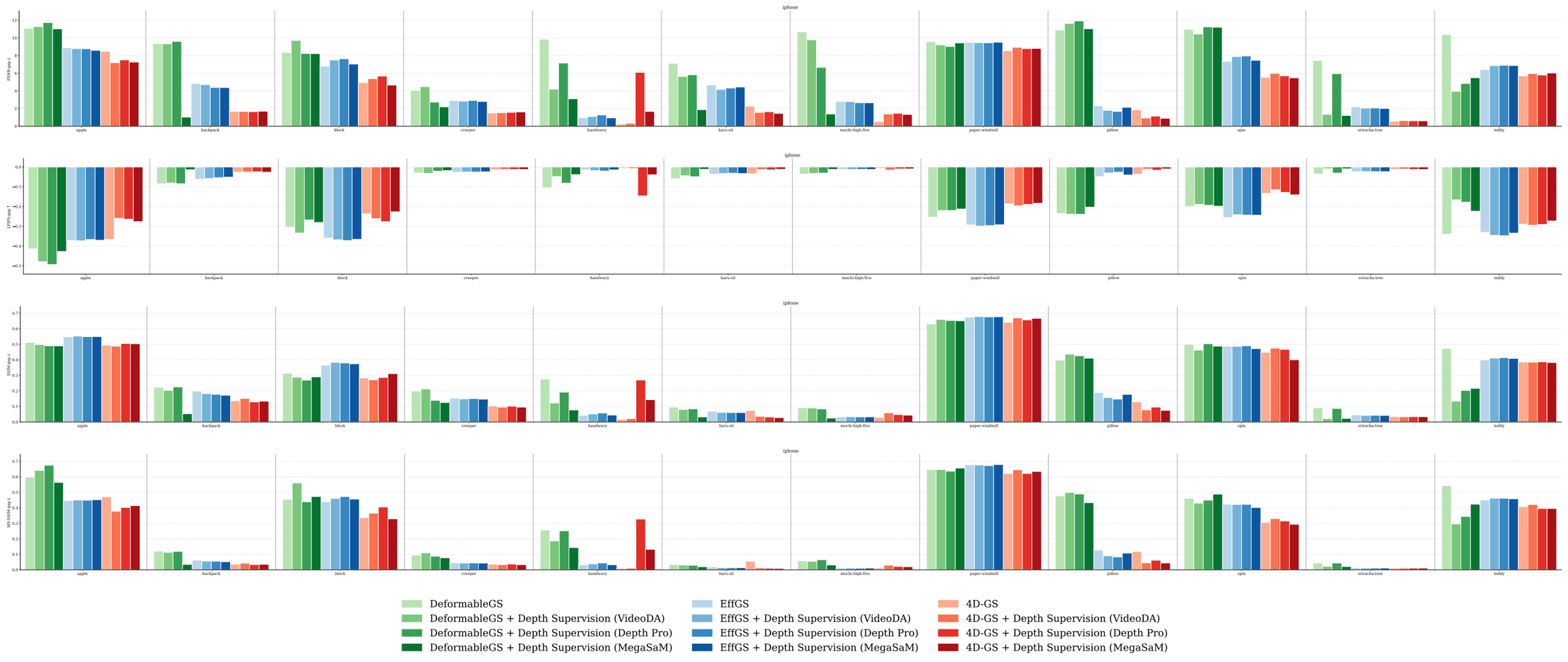


Figure B.8: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the iPhone dataset evaluated on the full image. It is notable that all methods overfit heavily on the paper-windmill scene, with MS-SSIM gaps of up to* 0.67.

Foreground Evaluation

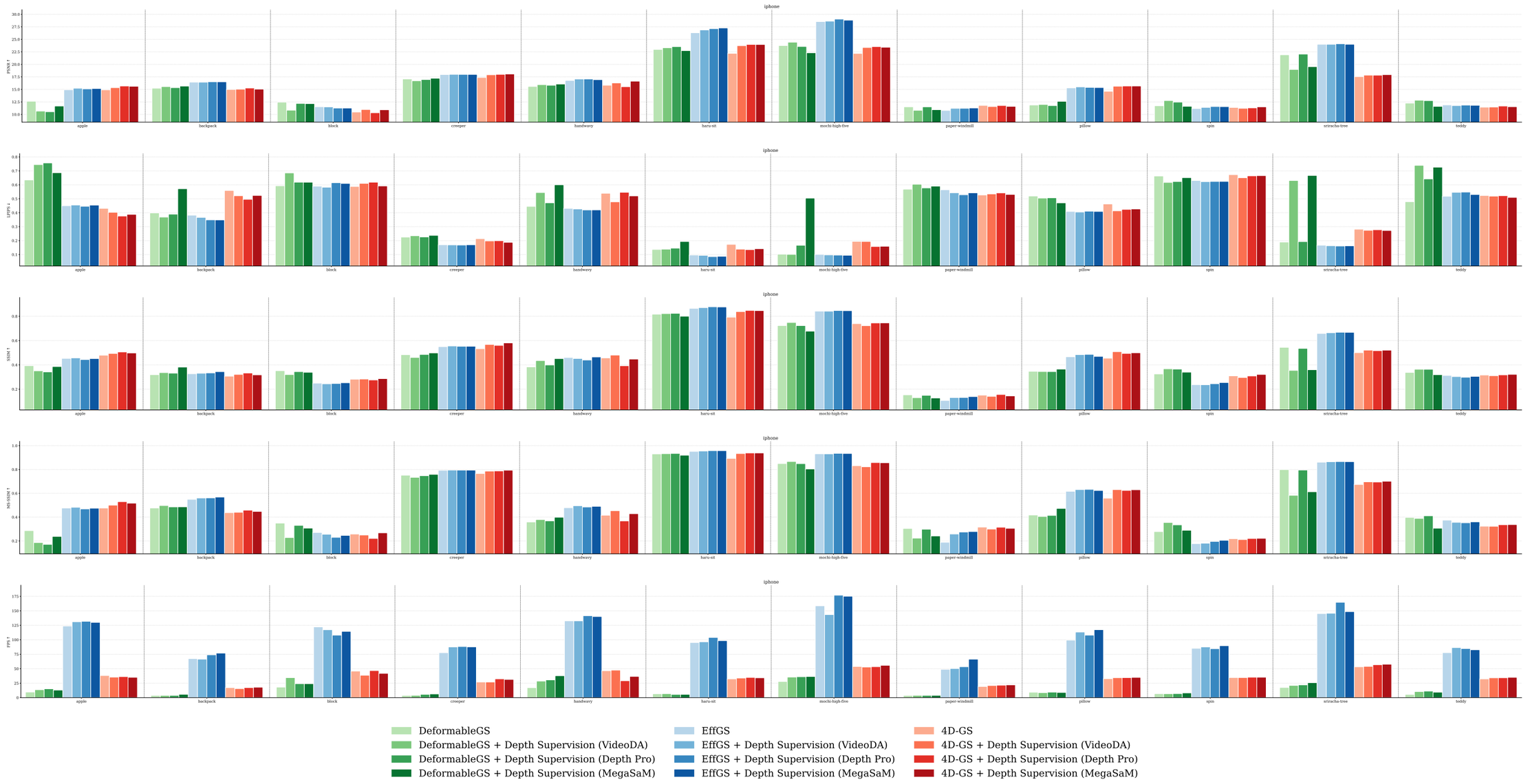


Figure B.9: ***Reconstruction Metrics.*** *Per scene quantitative results on the iPhone dataset evaluated on the foreground. The results reveal that the paper-windmill scene has a particularly challenging foreground.*

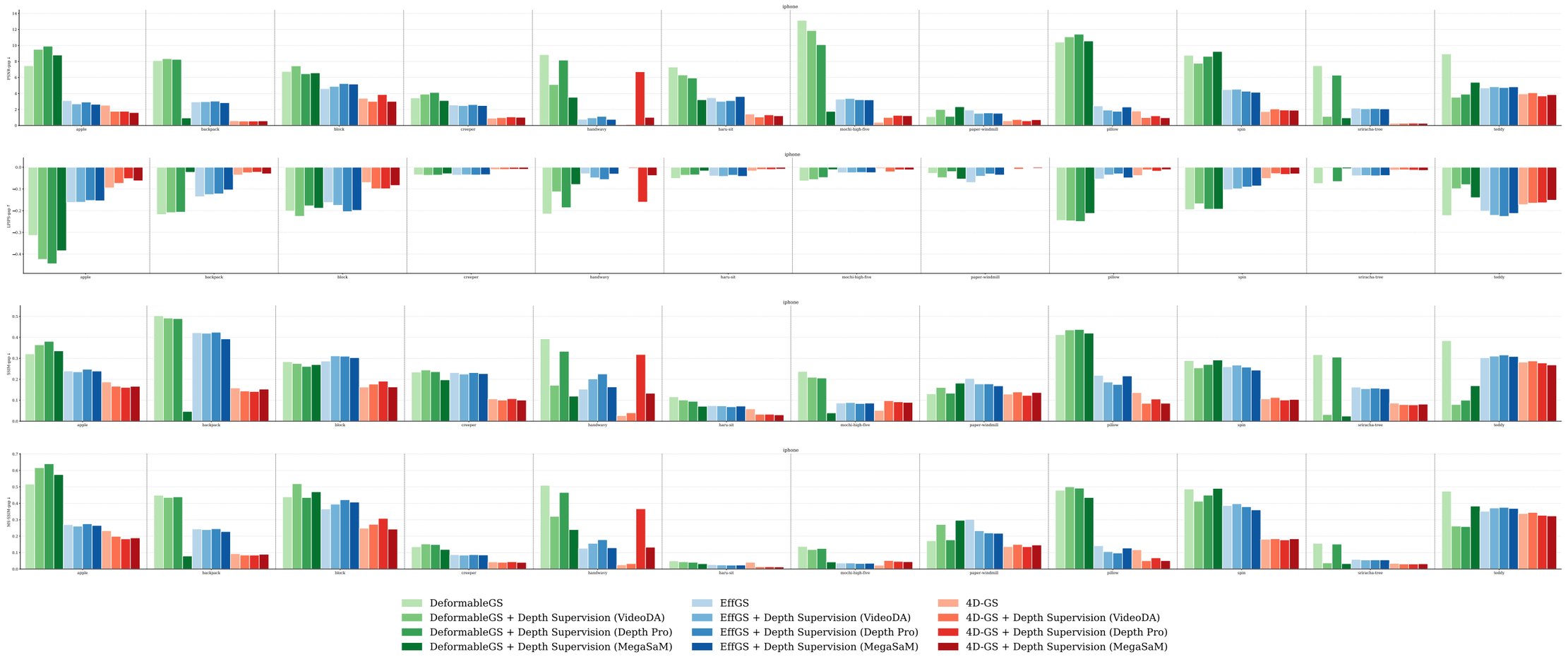


Figure B.10: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the iPhone dataset evaluated on the foreground. Remarkably, DeformableGS shows overfitting in the foreground for almost all scenes.*

C

Additional Results: Depth-Based Gaussian Initialization

C.1 Quantitative Results Per Dataset

C.1.1 Nerfies

Table C.1: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across the Nerfies dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	21.72	0.484	0.729	0.416	9.5	10197
DeformableGS + Depth Supervision (VideoDA)	22.27	0.487	**0.739**	0.452	9.7	10054
DeformableGS + Depth Supervision (Depth Pro)	21.97	0.478	0.727	**0.391**	10.1	10539
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	**22.29**	0.452	0.683	0.651	17.7	7272
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	21.86	0.420	0.632	0.738	25.6	5442
EffGS	21.12	0.480	0.695	0.505	91.4	**2722**
EffGS + Depth Supervision (VideoDA)	21.66	0.499	0.717	0.479	84.5	3091
EffGS + Depth Supervision (Depth Pro)	21.62	0.500	0.716	0.473	87.9	3142
EffGS + Depth Supervision + Gaussian Init (VideoDA)	21.63	0.501	0.718	0.447	**94.4**	3132
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	21.41	0.492	0.706	0.451	88.6	3235
4D-GS	21.38	**0.504**	0.721	0.486	37.7	6360
4D-GS + Depth Supervision (VideoDA)	21.50	0.501	0.716	0.490	32.5	6726
4D-GS + Depth Supervision (Depth Pro)	21.66	0.504	0.720	0.494	33.9	7353
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	21.11	0.477	0.689	0.578	38.2	7550
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	21.07	0.469	0.678	0.578	39.4	7654

Table C.2: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the Nerfies dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	20.63	0.499	0.674	0.451	9.6	10197
DeformableGS + Depth Supervision (VideoDA)	21.44	**0.517**	0.695	0.452	11.8	10054
DeformableGS + Depth Supervision (Depth Pro)	21.29	0.510	**0.695**	**0.440**	10.5	10539
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	**21.49**	0.508	0.692	0.525	18.8	7272
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	20.91	0.494	0.658	0.575	26.3	5442
EffGS	19.27	0.470	0.605	0.517	91.9	**2722**
EffGS + Depth Supervision (VideoDA)	19.78	0.484	0.622	0.502	89.1	3091
EffGS + Depth Supervision (Depth Pro)	19.75	0.480	0.620	0.493	90.0	3142
EffGS + Depth Supervision + Gaussian Init (VideoDA)	19.74	0.477	0.621	0.460	90.1	3132
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	19.51	0.464	0.605	0.462	**96.6**	3235
4D-GS	19.32	0.484	0.615	0.518	39.9	6360
4D-GS + Depth Supervision (VideoDA)	19.44	0.487	0.615	0.510	33.7	6726
4D-GS + Depth Supervision (Depth Pro)	19.69	0.492	0.627	0.500	36.3	7353
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	19.71	0.500	0.644	0.509	40.2	7550
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	19.66	0.493	0.635	0.504	40.3	7654

C.1.2 iPhone

Table C.3: ***Summary of Quantitative Results.*** *The table shows a summarized quantitative evaluation of all methods averaged across the iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	18.78	0.547	0.624	0.324	11.0	9986
DeformableGS + Depth Supervision (VideoDA)	18.39	0.535	0.595	0.403	14.4	8868
DeformableGS + Depth Supervision (Depth Pro)	18.57	0.533	0.606	0.381	14.5	8647
DeformableGS + Depth Supervision (MegaSaM)	18.00	0.505	0.583	0.464	15.3	7515
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	18.31	0.502	0.586	0.443	14.8	8239
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	17.86	0.495	0.574	0.481	15.8	7538
DeformableGS + Depth Supervision + Gaussian Init (MegaSaM)	18.12	0.486	0.560	0.499	18.0	6624
EffGS	20.25	0.581	0.663	0.307	99.4	**2131**
EffGS + Depth Supervision (VideoDA)	20.25	0.583	0.665	0.305	107.1	2493
EffGS + Depth Supervision (Depth Pro)	20.28	0.584	0.664	0.305	108.7	2246
EffGS + Depth Supervision (MegaSaM)	20.34	0.585	0.666	0.304	**109.6**	2325
EffGS + Depth Supervision + Gaussian Init (VideoDA)	20.38	0.585	0.672	0.300	104.2	2509
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	**20.42**	0.583	0.666	0.302	106.7	2352
EffGS + Depth Supervision + Gaussian Init (MegaSaM)	20.15	0.576	0.653	0.308	103.1	2495
4D-GS	18.62	0.570	0.639	0.320	35.9	7795
4D-GS + Depth Supervision (VideoDA)	18.97	0.577	0.652	0.304	35.9	8194
4D-GS + Depth Supervision (Depth Pro)	18.73	0.555	0.629	0.320	35.0	8302
4D-GS + Depth Supervision (MegaSaM)	19.00	0.568	0.645	0.306	35.4	8222
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	19.46	0.587	0.670	0.295	36.0	8417
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	19.80	**0.592**	**0.675**	0.290	36.5	8151
4D-GS + Depth Supervision + Gaussian Init (MegaSaM)	19.71	0.590	0.673	**0.285**	35.1	8326

Table C.4: ***Summary of Quantitative Results on Foreground.*** *The table shows a summarized quantitative evaluation of all methods averaged across foreground regions of the iPhone dataset.*

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
DeformableGS	15.75	0.432	0.517	0.413	10.9	9986
DeformableGS + Depth Supervision (VideoDA)	15.40	0.419	0.482	0.493	14.9	8868
DeformableGS + Depth Supervision (Depth Pro)	15.71	0.434	0.512	0.443	14.6	8647
DeformableGS + Depth Supervision (MegaSaM)	15.35	0.420	0.486	0.543	15.5	7515
DeformableGS + Depth Supervision + Gaussian Init (VideoDA)	15.66	0.432	0.502	0.467	15.4	8239
DeformableGS + Depth Supervision + Gaussian Init (Depth Pro)	15.39	0.419	0.491	0.523	16.2	7538
DeformableGS + Depth Supervision + Gaussian Init (MegaSaM)	15.56	0.424	0.483	0.523	18.2	6624
EffGS	17.14	0.461	0.556	0.376	102.8	**2131**
EffGS + Depth Supervision (VideoDA)	17.30	0.464	0.564	0.373	104.9	2493
EffGS + Depth Supervision (Depth Pro)	17.36	0.464	0.563	0.371	110.0	2246
EffGS + Depth Supervision (MegaSaM)	17.33	0.469	0.567	0.371	**110.7**	2325
EffGS + Depth Supervision + Gaussian Init (VideoDA)	17.39	0.469	**0.570**	**0.368**	102.7	2509
EffGS + Depth Supervision + Gaussian Init (Depth Pro)	**17.49**	0.463	0.562	0.372	108.5	2352
EffGS + Depth Supervision + Gaussian Init (MegaSaM)	17.18	0.459	0.555	0.376	103.1	2495
4D-GS	15.40	0.444	0.514	0.431	36.1	7795
4D-GS + Depth Supervision (VideoDA)	15.86	0.457	0.529	0.411	35.8	8194
4D-GS + Depth Supervision (Depth Pro)	15.88	0.455	0.529	0.413	36.1	8302
4D-GS + Depth Supervision (MegaSaM)	16.00	0.461	0.537	0.410	36.5	8222
4D-GS + Depth Supervision + Gaussian Init (VideoDA)	16.33	**0.475**	0.559	0.392	36.5	8417
4D-GS + Depth Supervision + Gaussian Init (Depth Pro)	16.47	0.475	0.560	0.391	36.9	8151
4D-GS + Depth Supervision + Gaussian Init (MegaSaM)	16.42	0.474	0.557	0.395	38.0	8326

C.2 Per Metric Quantitative Result Plots

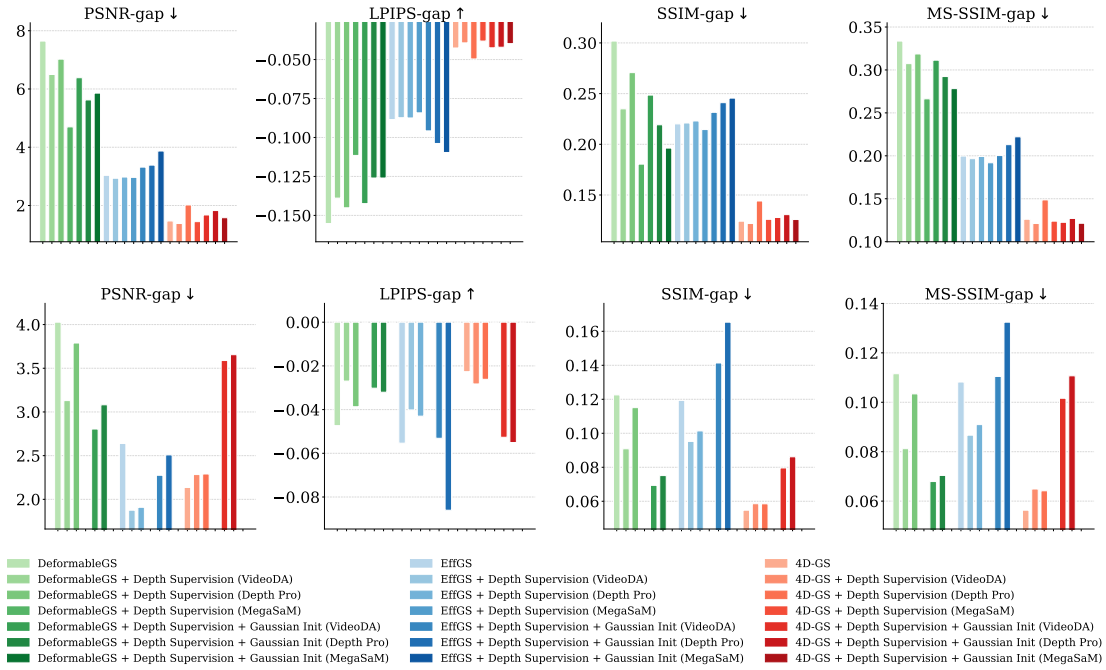


Figure C.1: *Train-test gaps for all metrics on the iPhone (top) and the Nerfies dataset (bottom), reported for the foreground image showing the non-regularizing effect of depth-based Gaussian initialization across the different methods, with the tendency towards stronger overfitting.*

C.3 Per Dataset Quantitative Result Plots

C.3.1 Foreground Evaluation

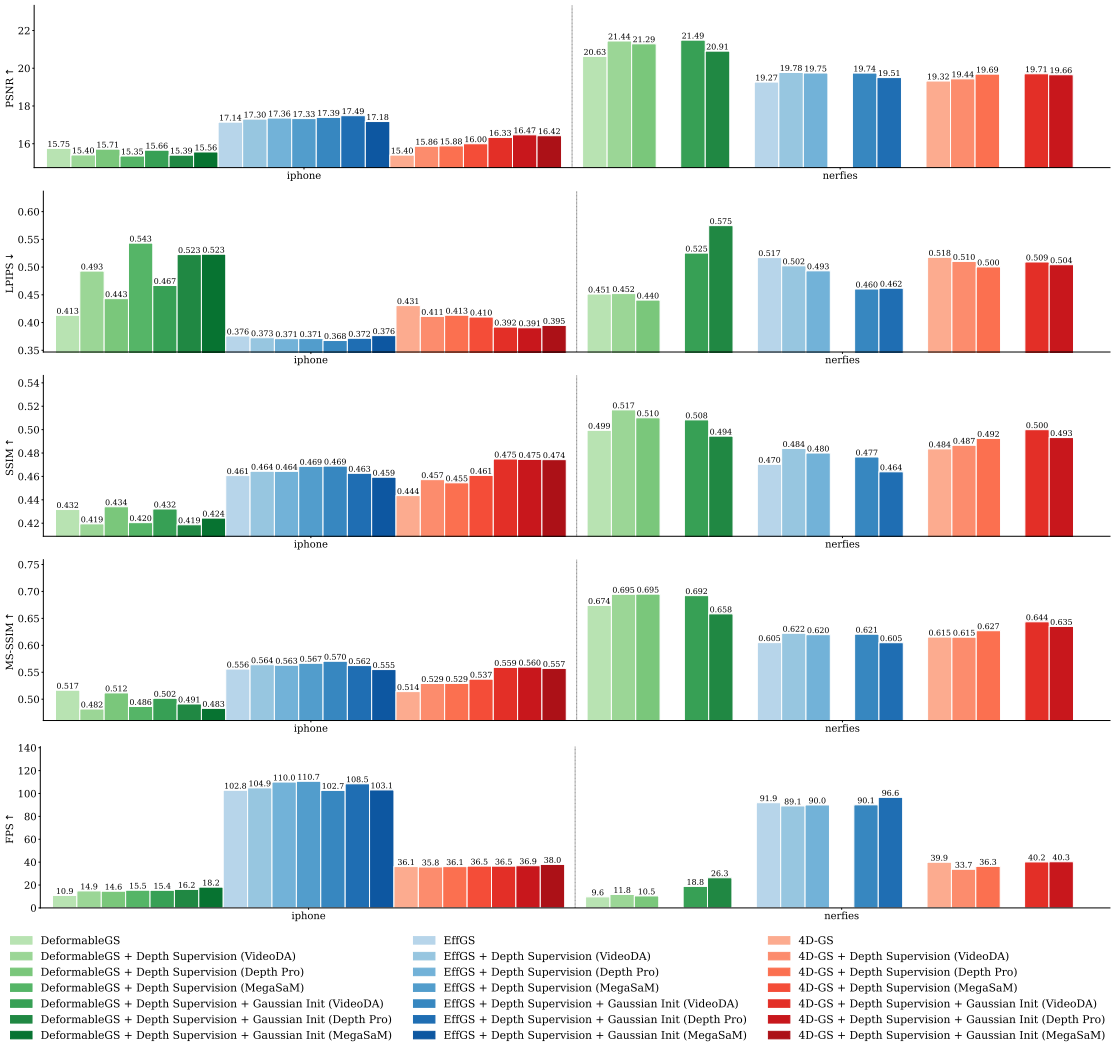


Figure C.2: *Per method quantitative results on the iPhone and the Nerfies datasets evaluated on the foreground. Corresponding numerical values are plotted above each bar. The results indicate that, with especially 4D-GS on the iPhone dataset benefits of integrating depth-based Gaussian initialization.*

C.4 Per Scene Quantitative Result Plots

C.4.1 Nerfies

Full-Image Evaluation

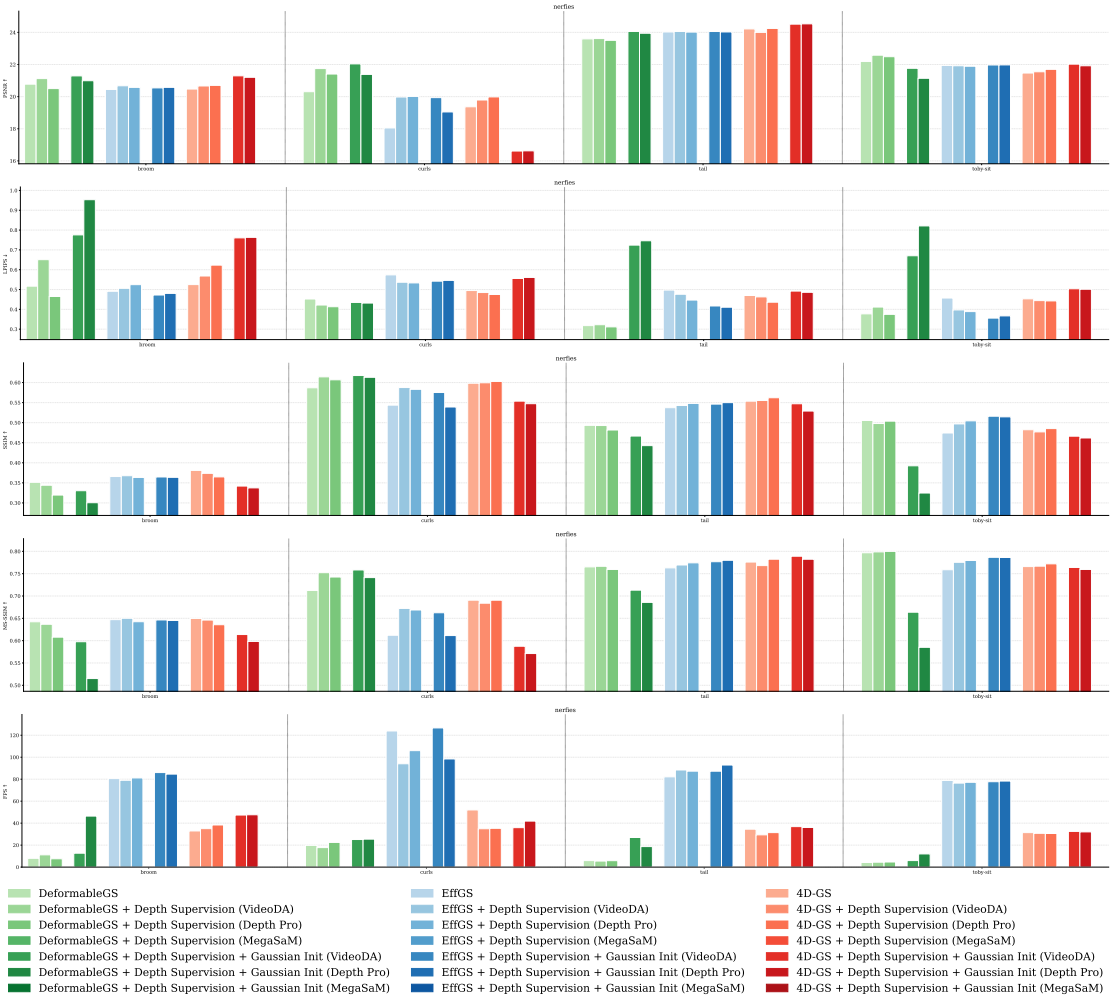


Figure C.3: ***Reconstruction Metrics.*** *Per scene quantitative results on the Nerfies dataset evaluated on the full image. It shows that integrating depth-based Gaussian initialization notably degrades the reconstruction quality of 4D-GS on the curls scene.*

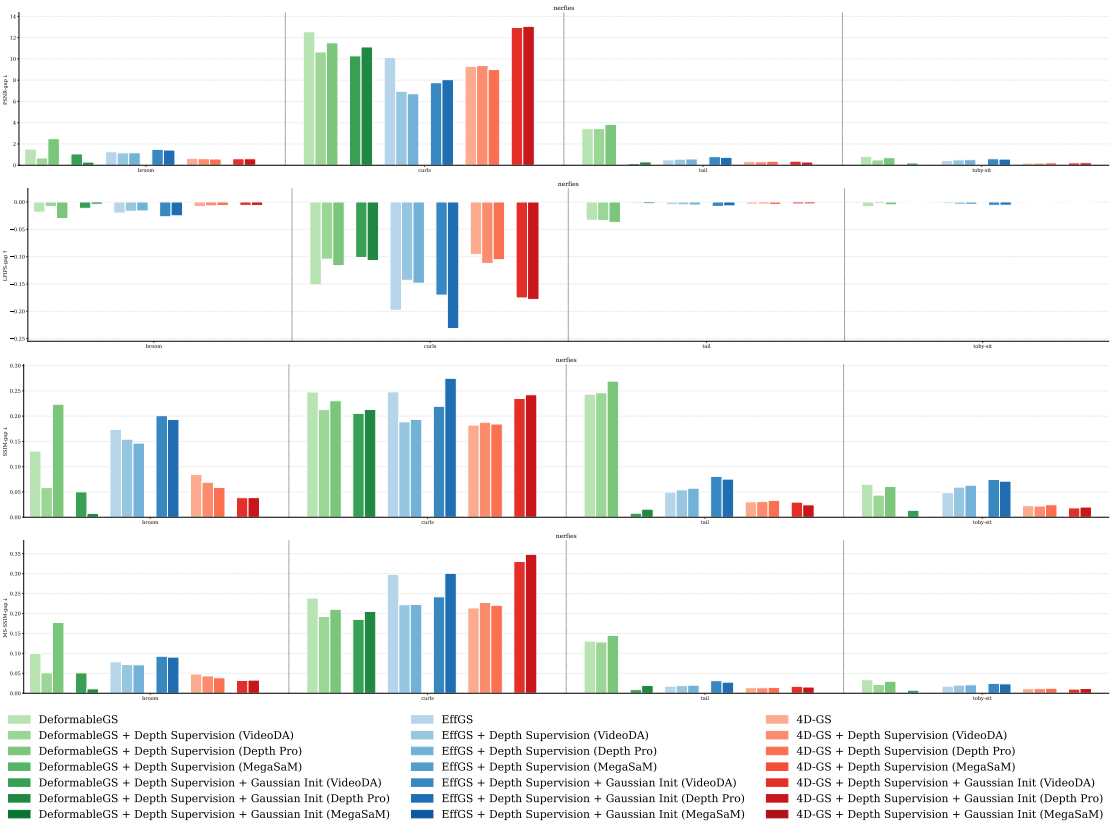


Figure C.4: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the Nerfies dataset evaluated on the full image. It is striking that integrating depth-based Gaussian initialization leads to much stronger overfitting on the curls scene when using 4D-GS, as the PSNR gap increases by 4 dB compared to the baseline.*

Foreground Evaluation

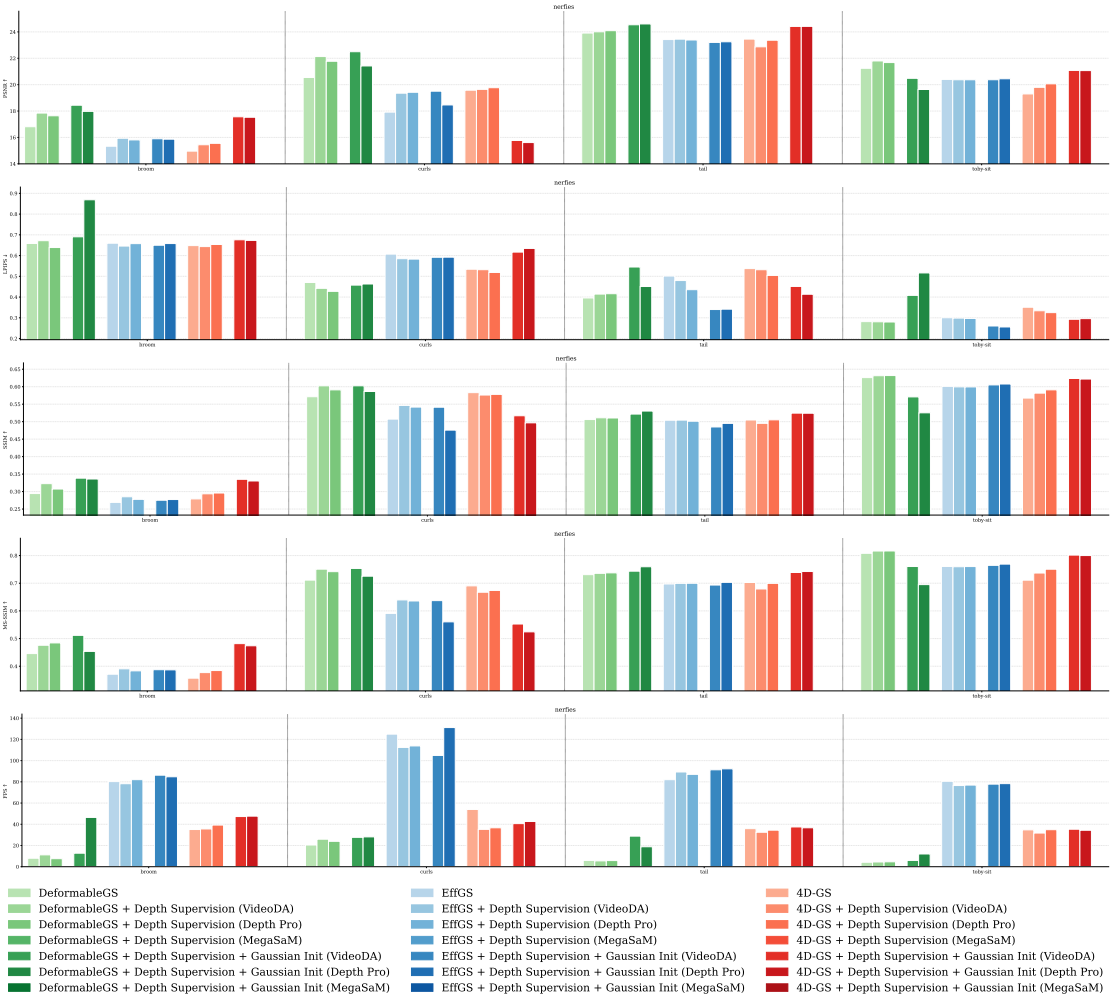


Figure C.5: ***Reconstruction Metrics.*** *Per scene quantitative results on the Nerfies dataset evaluated on the foreground. The results demonstrate that the broom foreground especially benefits from the depth-based Gaussian initialization, with 4D-GS showing an approximate PSNR gain of 2 dB.*

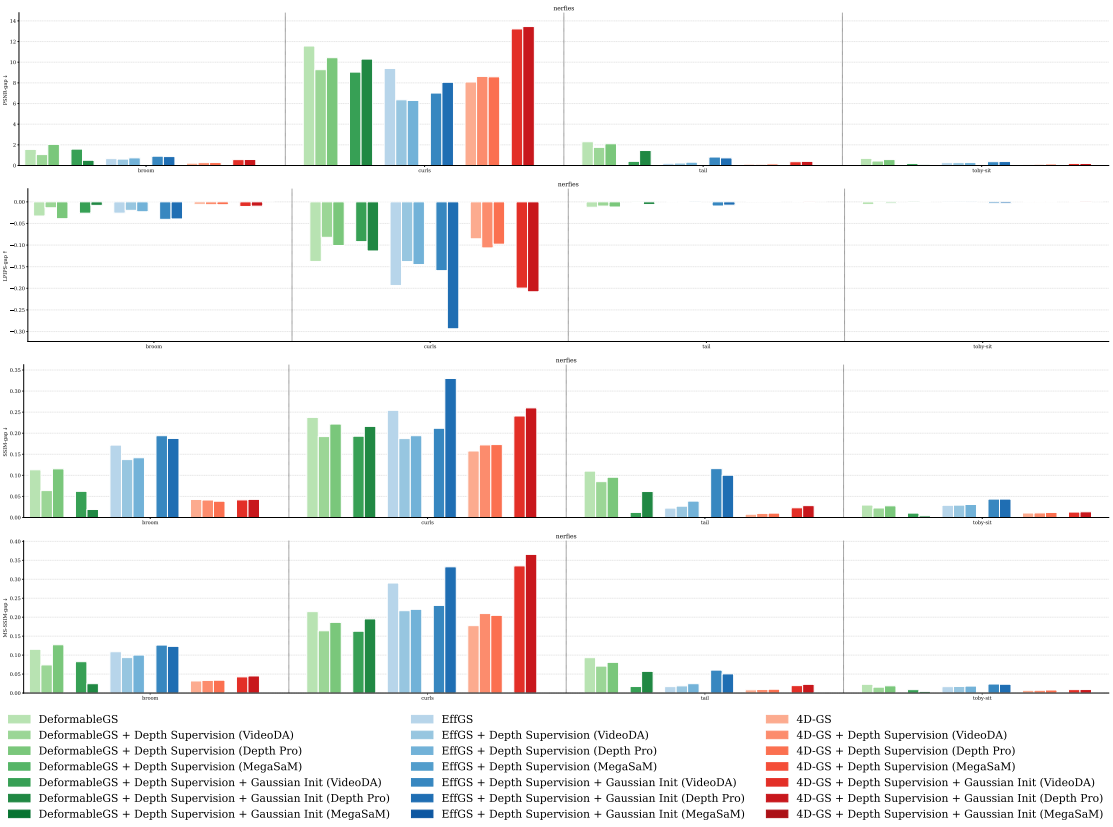


Figure C.6: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the Nerfies dataset evaluated on the foreground. Remarkably, reconstructions of the broom scene using EffGS benefit from depth-based Gaussian initialization, as the SSIM gap decreases from* 0.1 *at baseline to only* 0.2*.*

C.4.2 iPhone

Full-Image Evaluation

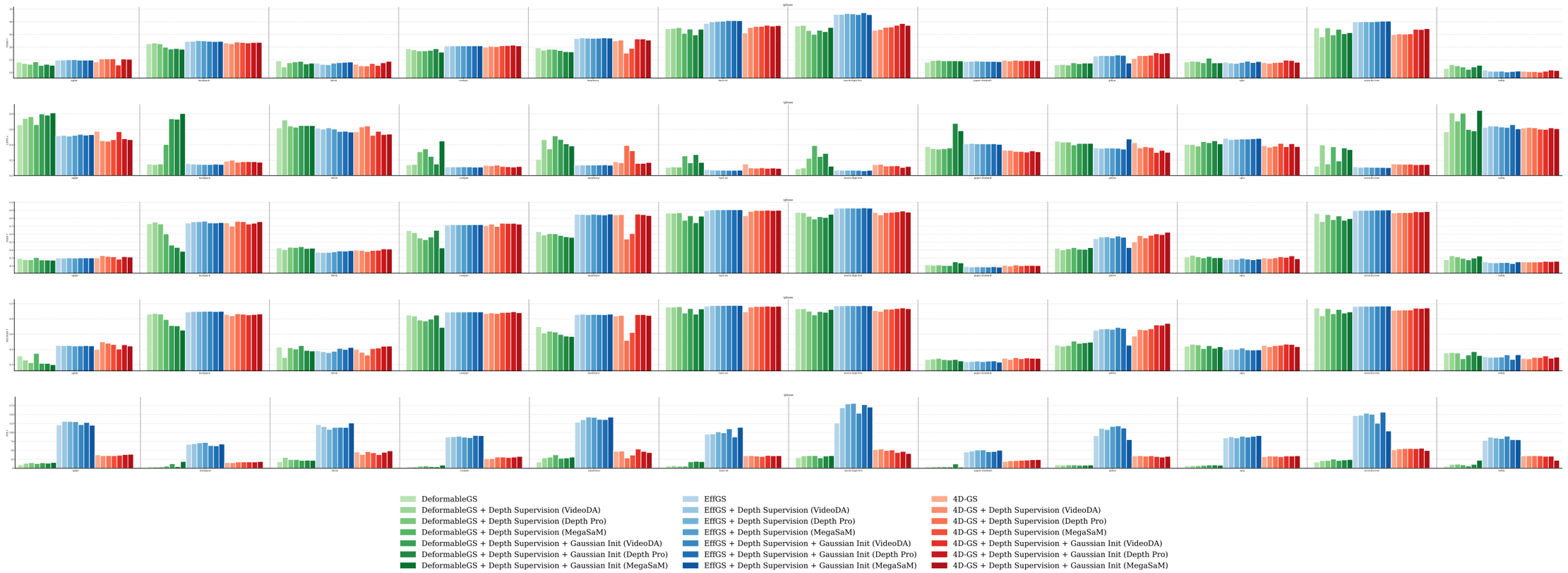


Figure C.7: ***Reconstruction Metrics.*** *Per scene quantitative results on the iPhone dataset evaluated on the full image. The results reveal that 4D-GS leverages the depth-based Gaussian initialization, particularly on the sriracha-tree scene, achieving a PSNR increase of approximately* 2.5 *dB.*

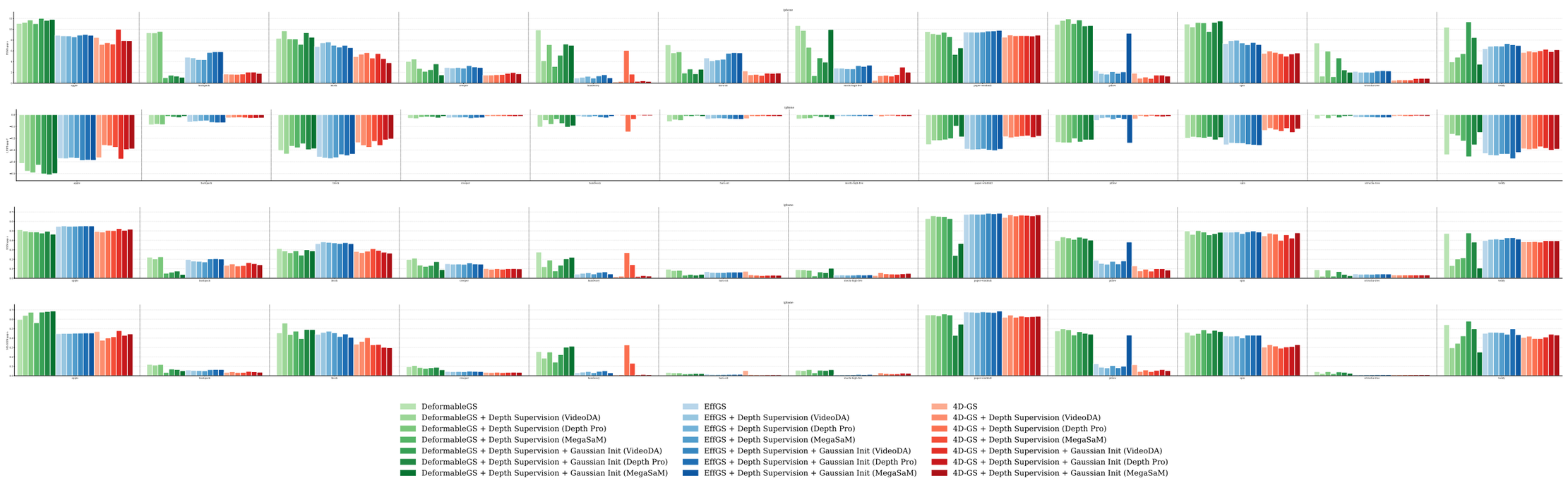


Figure C.8: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the iPhone dataset evaluated on the full image. It is notable that DeformableGS can reduce its overfitting on the paper-windmill scene with the depth-based initialization, as the gaps of all metrics decrease significantly.*

Foreground Evaluation

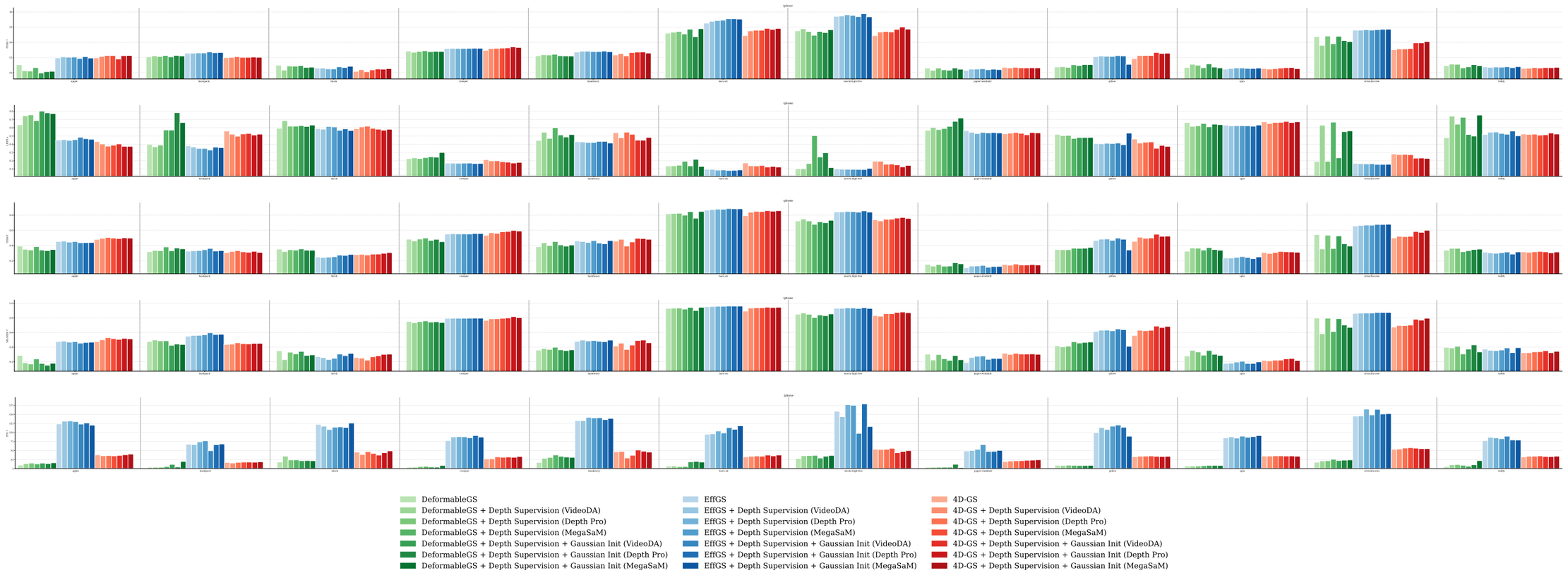


Figure C.9: ***Reconstruction Metrics.*** *Per scene quantitative results on the iPhone dataset evaluated on the foreground. The results reveal that EffGS leverages the integration of depth-based Gaussian Initialization, especially on the haru-sit scene.*

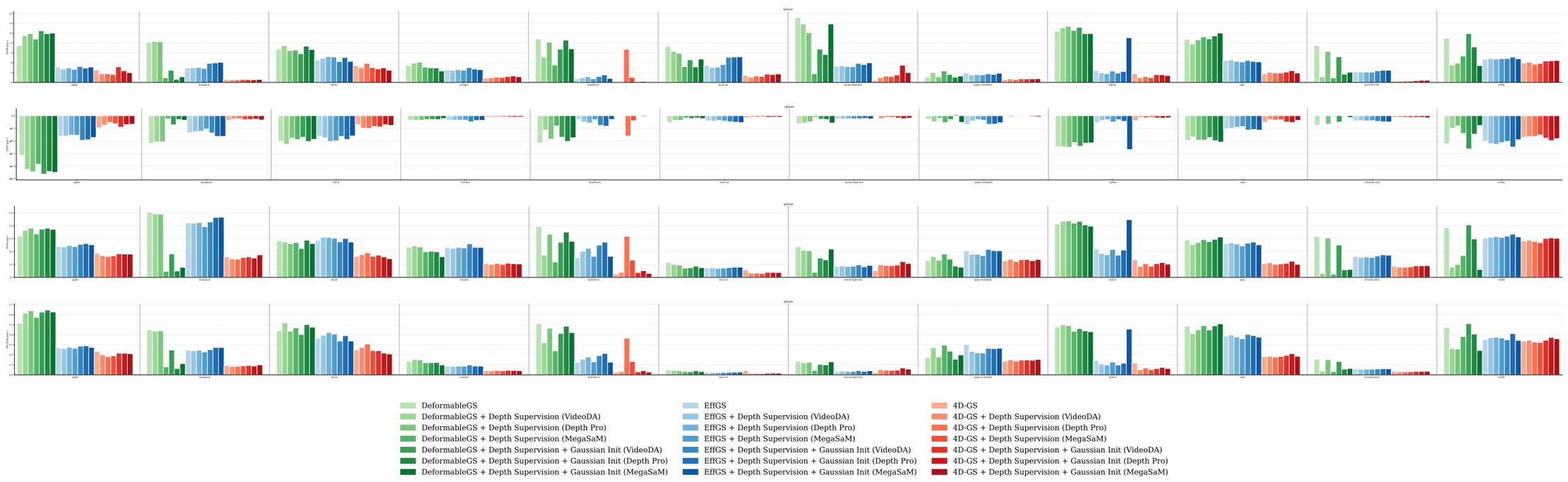


Figure C.10: ***Train-Test Gaps of Reconstruction Metrics.*** *Per scene quantitative train-test gaps on the iPhone dataset evaluated on the foreground. Remarkably, when using MegaSaM for the depth-based initialization, EffGS shows increased overfitting in the foreground of the pillow scene.*

D

Erklärung Abschlussarbeit

Erklärung

Laut Beschlüssen der Prüfungsausschüsse Bioinformatik, Informatik, Informatik Lehramt, Kognitionswissenschaft, Machine Learning, Medieninformatik und Medizininformatik der Universität Tübingen vom 05.02.2025. Gültig für Abschlussarbeiten (B.Sc./M.Sc./B.Ed./M.Ed.) in den zugehörigen Fächern. Bei Studienarbeiten und Hausarbeiten bitte nach Maßgabe des/der jeweiligen Prüfers/Prüferin.

1. Allgemeine Erklärungen

Hiermit erkläre ich:

- Ich habe die vorgelegte Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
- Ich habe alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet.
- Die Arbeit war weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens.
- Falls ich ein elektronisches Exemplar und eines oder mehrere gedruckte und gebundene Exemplare eingereicht habe (z.B., weil der/die Prüfer/in(nen) dies wünschen): Das elektronisch eingereichte Exemplar stimmt exakt mit dem bzw. den von mir eingereichten gedruckten und gebundenen Exemplar(en) überein.

2. Erklärung bezüglich Veröffentlichungen

Eine Veröffentlichung ist häufig ein Qualitätsmerkmal (z.B. bei Veröffentlichung in Fachzeitschrift, Konferenz, Preprint, etc.). Sie muss aber korrekt angegeben werden. Bitte kreuzen Sie die für Ihre Arbeit zutreffende Variante an:

☒ Die Arbeit wurde bisher weder vollständig noch in Teilen veröffentlicht.

☐ Die Arbeit wurde in Teilen oder vollständig schon veröffentlicht. Hierfür findet sich im Anhang eine vollständige Tabelle mit bibliographischen Angaben.

3. Nutzung von Methoden der künstlichen Intelligenz (KI, z.B. chatGPT, DeepL, etc.)

Die Nutzung von KI kann sinnvoll sein. Sie muss aber korrekt angegeben werden und kann die Schwerpunkte bei der Bewertung der Arbeit beeinflussen. Bitte kreuzen Sie alle für Ihre Arbeit zutreffenden Varianten an und beachten Sie, dass die Varianten 3.4 - 3.6 eine vorherige Absprache mit dem/der Betreuer/in voraussetzen:

☐ 3.1. Keine Nutzung: Ich habe zur Erstellung meiner Arbeit keine KI benutzt.

☒ 3.2. Korrektur Rechtschreibung & Grammatik: Ich habe KI für Korrekturen der Rechtschreibung und Grammatik genutzt, ohne dass es dabei zu inhaltlich relevanter Textgeneration oder Übersetzungen kam. Das heißt, ich habe von mir verfasste Texte in derselben Sprache korrigieren lassen. Es handelt sich um rein sprachliche Korrekturen, sodass die von mir ursprünglich intendierte Bedeutung nicht wesentlich verändert oder erweitert wurde. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☐ 3.3. Unterstützung bei der Softwareentwicklung: Ich habe KI als Unterstützung beim Schreiben von Code in der Softwareentwicklung genutzt. Es handelt sich hierbei lediglich um Unterstützung und nicht um die automatische Generierung von größeren Programm-Teilen. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☐ 3.4. Übersetzung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Übersetzung von mir in einer anderen Sprache geschriebenen Texte genutzt. Jede derartige Übersetzung ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller übersetzten Textstellen und der verwendeten Programme mit Versionsnummer.

☒ 3.5. Code-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Code in der Softwareentwicklung genutzt. Der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

☐ 3.6. Text-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Text in meiner Arbeit genutzt. Jede derartige Verwendung von KI ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

Falls ich in irgendeiner Form KI genutzt haben (siehe oben), dann erkläre ich:

Mir ist bewusst, dass ich die Verantwortung trage, falls es durch die Verwendung von KI zu fehlerhaften Inhalten, zu Verstößen gegen das Datenschutzrecht, Urheberrecht oder zu wissenschaftlichem Fehlverhalten (z.B. Plagiaten) kommt.

4. Abschluss und Unterschrift(en)

Mir ist bekannt, dass ein Verstoß gegen diese Erklärung prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass die Prüfungsleistung mit „nicht ausreichend" bzw. die Studienleistung mit „nicht bestanden" bewertet wird und bei mehrfachem oder schwerwiegendem Täuschungsversuch eine Exmatrikulation erfolgen bzw. ein Verfahren zur Entziehung eines eventuell verliehenen akademischen Titels eingeleitet werden kann.

Jonas Klötzl	Obersheim, 17.10.2025	Jonas Klötzl
Vorname, Nachname Student/in	Ort, Datum	Unterschrift

Die Punkte 3.4 - 3.6 erfordern eine Zustimmung des/r Betreuer/in. Sollten Sie einen dieser Punkte angekreuzt haben, dann sollte der/die Betreuer/in bitte hier unterschreiben:

Ich habe der oben genannten Nutzung von KI zur Erstellung der Arbeit zugestimmt.

Andreas Geiger	Tü, 30.11.2025	
Vorname, Nachname Betreuer/in	Ort, Datum	Unterschrift

E

Usage of AI

This appendix provides full documentation of AI usage in this thesis, in accordance with the "Erklärung Abschlussarbeit" Appendix D. Section Section E.1 lists all the programs used and their version numbers as demanded by variant 3.2. Section E.2 fulfills the documentation prescribed by variant 3.5 by additionally indicating the corresponding locations in the code, and the prompts used.

E.1 Correction of Spelling & Grammar

- Grammarly (Safari extension): 9.94.0
- ChatGPT: GPT5

E.2 Code Generation

E.2.1 Own viewmatrix transform

Location `MonoPriorsDyGauBench_code/src/models/GS3d.py/forward()`

Software and Version Number ChatGPT, GPT4.5

Prompt

```
transform this cuda code exactly to python:
__forceinline__ __device__ float3 transformPoint4x3(const float& p, const float* matrix)
{
        float3 transformed = {
                matrix[0] * p.x + matrix[4] * p.y + matrix[8] * p.z + matrix[12],
                matrix[1] * p.x + matrix[5] * p.y + matrix[9] * p.z + matrix[13],
                matrix[2] * p.x + matrix[6] * p.y + matrix[10] * p.z + matrix[14],
        };
        return transformed;
}}
called like: p_view = transformPoint4x3(p_orig, viewmatrix)
using those python variables:
points3D = result["means3D"].detach()  # (N, 3)
```

```
        # Transform points to camera/view space
        viewmatrix = raster_settings.viewmatrix
```

E.2.2 Rectification of Out-of-Memory Error through Checkpointing

Location `MonoPriorsDyGauBench_code/src/models/GS3d.py/forward()`

Software and Version Number ChatGPT, GPT-5

Prompt

```
although emptying the caches i get an oom error here always when calling the second rasterizer:
[PASTED TWO RASTERIZER CALLS OF MonoPriorsDyGauBench_code/src/models/GS3d.py/forward() FUNCTION]
---
but how can i release this rasterizer gpu memory before the second rasterizer call ?
---
again i need gradients for both rasterizers !!!
```

E.2.3 Proposal of using Absolute Value of the Scale for Alignment in Loss Function

Location `MonoPriorsDyGauBench_code/src/utils/depth_utils.py/normalized_depth_scale_and_shift()`

Software and Version Number ChatGPT, GPT-5

Prompt

```
I have following depth loss for my thesis:
def get_depth_loss(pred, gt, log_output=False, mask=None):
    pred = get_scaled_shifted_depth(pred, gt, mask)
    if mask is None:
        mask = torch.ones_like(pred)
    if log_output:
        loss = torch.abs(pred - gt) * mask
    else:
        loss = torch.sum(torch.abs(pred - gt) * mask) / torch.sum(mask)
    return loss

with
[PASTED WHOLE MonoPriorsDyGauBench_code/src/utils/depth_utils.py FILE]
which i use with inverted depth. Problem now is that i get negative scale values because the
↪  initialization is very sparse and consequentely I have many areas with "infinity" distance as
↪  the max distance in inv depth is defined with 0.0. What should i do ?
```

E.2.4 Proposition of Memory Cleaning Function for Out-of-Memory Error

Location `MonoPriorsDyGauBench_code/src/models/GS3d.py/memory_cleanup()`

Software and Version Number Claude, Sonnet 4.5

Prompt

```
---
[PASTED WHOLE MonoPriorsDyGauBench_code/src/models/GS3d.py FILE]
this is the code of my dynamic gaussian splatting comparing paper where I use the methods mlp,
↪  curve and hexplane. However, the map method gets out of memory issues very fast. So what do
↪  you think could cause the issue:
---
How can I track the sizes of tensors like this ?
---
is there a way of finding out the size of things the garbage collector deletes ?
---
```

E.2.5 Initialization of Gaussians via Depth Map

Location `MonoPriorsDyGauBench_code/src/data/init_gaussians_with_depth.py/init_gaussians_with_depth()`

Software and Version Number ChatGPT, GPT-5

Prompt

```
ok so my current model takes as input the images and a sfm pointcloud to roughly initialize the
↪  scene. But I now want to use a metric depth map from video depth anything, unproject it, and
↪  align it to the pointcloud to have a better initialization. Right now i have this:
        ply_path = os.path.join(datadir, "points.npy")
        xyz = np.load(ply_path, allow_pickle=True)
        print("Reading in Points from the provided pointcloud.")
        # if xyz's  shape[0] is greater than 100000, then subsample evenly 100000 points
        if len(xyz) > 100000:
            gap = len(xyz) // 100000
            xyz = xyz[::gap]
            print("Limiting Points read in from the provided pointcloud")
        ######################################################################
        # First try of loading the depth map as pointcloud initialization
        ######################################################################
        depth = np.load("/home/geiger/gwb215/MonoPriorsDyGauBench_code/data....npy")
        print("reading depth init", depth.shape)
        # Load camera
        import json
        with open("/home/geiger/gwb215/MonoPriorsDyGauBench_code/data/....json", "r") as f:
            cam = json.load(f)
        print("reading cam init")
---
the camera.json looks like this:
[PASTED WHOLE MonoPriorsDyGauBench_code/data/nerfies/broom/camera.json FILE]
---
ok i guess the problem is a bit that my depth maps have many points in the foreground where the
↪  moving object is. But there find_correspondences will not find any because the sfm are very
↪  sparse for the static background.
---
or maybe it is easier to align the points in 2d and then unproject them ?
---
because the problem with the 3d alignment at the moment is that the depth map pointcloud is
↪  squished and the sfm points have scale of factor 10x
---
I'm thinking of something like this:
def load_camera_json(json_path):
```

```
[GENERATED FUNCTION FROM PREVIOUS PROMPT]
def rasterize_sfm_points(xyz_sfm, intrinsics, img_size, max_depth=5.0):
[GENERATED FUNCTION FROM PREVIOUS PROMPT]
def unproject_depth_to_camera_points(depth, intrinsics, mask_valid=None):
[GENERATED FUNCTION FROM PREVIOUS PROMPT]
def transform_points(points, pose4):
[GENERATED FUNCTION FROM PREVIOUS PROMPT]
# -------------------------
# High-level pipeline function that ties everything together
# -------------------------
def align_depth_to_sfm(xyz_sfm, depth, cam_json_path, max_depth_points=100000):
    intrinsics, pose_cam_to_world = load_camera_json(cam_json_path)

    # rasterize sfm points into image plane to get depth map

    # align depth map to sfm depth map via ransac maybe

    # unproject depth map to get depth points in camera space

    # transform depth points to world space
---
but do the world to camera translation exactly like that:
[PASTED TRANSFORM CODE BETWEEN TWO RASTERIZER CALLS OF
↪  MonoPriorsDyGauBench_code/src/MODELS/GS3d.py/forward() FUNCTION]
---
Maybe i should use these functions from the utils.py file:
[PASTED WHOLE MonoPriorsDyGauBench_code/src/data/utils.py FILE]
---
Now that we have this, we need to align the depth map to it and unproject it again:
[PASTED MY CURRENT init_gaussians_with_depth FUNCTION]
---
now i just need to add this after calling the function and i have a pointcloud with sfm And depth
↪  map points:
all_points = np.stack((pts_world, xyz), axis=1)
---
but now I am saving all hundreds of thousand of points ?
---
so this should add all sfm points + depth map points until the total limit is reached right ?
        # Subsample, e.g., max 100k points
        max_points = 100_000
        max_depth_map_points = max_points - xyz.shape[0]
        if pts_world.shape[0] > max_depth_map_points:
            idx = np.random.choice(pts_world.shape[0], max_depth_map_points, replace=False)
            pts_world = pts_world[idx]

        all_points = np.concatenate((pts_world, xyz), axis=0)  # shape [N_depth + N_sfm, 3]

        # optional saving of the created pointcloud

        ↪  np.save("/home/geiger/gwb215/MonoPriorsDyGauBench_code/data/nerfies/toby-sit/my_points.npy",
        ↪  all_points)
---
ok good, but did you think about the fact that the depth map resolution is smaller when
↪  unprojecting ?
---
so this is correct ?
[PASTED MY CURRENT init_gaussians_with_depth FUNCTION]
---
because actually the depth map is a bit slanted, so the "wall" of the depth map does not match the
↪  "wall" of the sfm points in angle, if you know what i mean ?
```

```
---
ok, true. But one problem is that the sfm and depth map are flipped 180°, so they are not matching
---
but the depth map projection is correct, it is only after the unprojection that it is not correct.
---
please load in the .png mask from the mask_path
---
why is this valid mask where i integrated now the newly loaded one is not only true false but has
↪  other values between 0 and 1 in it ?
    valid_mask = np.isfinite(sfm_depth_map) & (sfm_depth_map > 0) & (depth > 0) & (~mask)
---
but i want to only keep the background as only here the sfm points have meaningful values.
---
having this, i want to also output the corresponding rgb values in this
---
why not do it here ?
    # fill depth map (rasterization)
    for px, d in zip(sfm_pixels_rescaled.astype(int), depths):
        x, y = px
        if 0 <= x < W and 0 <= y < H:
            sfm_depth_map[y, x] = min(sfm_depth_map[y, x], d)
---
and how can I save them now in the same order as the gaussians ?
---
this is not good, because we do not have real data for the sfm data:
            sfm_color_map[y, x] = img[y, x]
I did that, but i don't know how i match the 2d rgb values now to the individual 3d points:
    valid_pixels_colors = np.zeros((valid_pixels.shape[0], 3), dtype=np.float32)
    for i in range(valid_pixels.shape[0]):
        valid_pixels_colors[i] = img[int(valid_pixels[i,1]), int(valid_pixels[i,0])] / 255.0  #
        ↪  normalize to [0,1]
---
ok can you create a 3d version of this with ICP (iterative closest point) ?
[PASTED MY CURRENT init_gaussians_with_depth FUNCTION]
---
or is there any library which ships an icp algorithm ?
---
ok good then rewrite this with trimesh
---
so maybe unproject a secondary depth map from 2d where only points with sfm correspondencies are
↪  used for alignment
---
i was wrong, we need this, because we need to apply the translation to all world points, is this
↪  then correct ?
    # Run ICP
    T_icp, pts_aligned = run_icp_trimesh(pts_world_to_align, sfm_pts_to_align)

    print(valid_pixels.shape, valid_pixels)

    # Apply transformation to full resolution source points
    src_h = np.hstack([pts_world, np.ones((pts_world.shape[0], 1))])
    pts_world = (T_icp @ src_h.T).T[:, :3]
---
is this the correct way to assign the colors only to sfm points in front of the depth map points?
[PASTED MY CURRENT init_gaussians_with_depth FUNCTION]
```

E.2.6 Integration of Custom Sampler for Training Image Scheduling

Location `MonoPriorsDyGauBench_code/src/data/Nerfies.py/ProgressiveSampler Class`

Software and Version Number ChatGPT, GPT-5

Prompt

```
will my custom sampler work ?
[PASTED WHOLE MonoPriorsDyGauBench_code/src/data/Nerfies.py FILE]
---
this seems to work. but can you give me some code to print some info about it
---
ok this is interesting the printing says step 39 at global_step 1386
---
```

E.2.7 Creating Initial Script for Comparison of Depth Methods

Location `MonoPriorsDyGauBench_code/depth_prediction_models_evaluation/new_mse_over_iphone.py`

Software and Version Number ChatGPT, GPT-4.5

Prompt

```
Alright I aligned my image depth predictions with two different methods, so I want to check how
↪  much the metric depth is deviating between the two methods using mean square error. data is
↪  stored in .npy files. Can you give me a python method which does that ? So as Input two metric
↪  depth predictions with one .npy file for each frame and as an output an per image mean square
↪  error over all pixels and a plot which gives the mse of each image during the video.
---
make such that it can be called from terminal
---
perfect, now allow two more inputs and also plot them
---
But each input folder of .npy file should be plotted as an individual line.
---
No one line for method 1 with mse over all frames, one line for method 2 with mse over all frames,
↪  one line for method 3 ...
---
Ok having this, can you add a separate plot which is a bar chart. make groups for each metric
↪  (mse, absrel, delta_1) and plot the median errors of the corresponding depth prediction
↪  method.
[PASTED CURRENT own_scripts/new_mse_over_iphone.py FILE]
---
perfect, ok problem is that for mse and absrel lower is better and for delta 1 higher is better
↪  can you apply a y axis scaling similar to a sigmoid function ?
---
Is there no function which plots values from 0 to 1 to 0 to 1 but which focuses on the areas of 0
↪  and 1
---
nononono, I want this but three different plots side by side:
---
no, normal yscale just three plots side by side
---
```

```
ok now now change here:
1. tilt the method names more such that they don't get shifted
2. add the absolute values inside or on top of each bar
3. add the range in the delta 1 to 0.8 to 1
---
can I do some tabulators such that all median values are equally placed:
plt.plot(x, smooth, color=color, label=f'{pretty_methods[method]} Median MSE:
↪  {mse_medians[method]:.5f}', linewidth=0.5)
---
Looking at this script how can I change it such that I can easily add another depth prediction
↪  method:
```

E.2.8 Creating Function for Alignment via RANSAC

Location `MonoPriorsDyGauBench_code/depth_prediction_models_evaluation/align_metric_with_ransac_lidar.py`

Software and Version Number ChatGPT, GPT-4.5

Prompt

```
explain to me the ransac algorithm
---
ok good, then try to implement it into this alignment procedure:
[PASTED WHOLE own_scripts/align_metric_with_lidar.py FILE]
---
aah ok, I see I think you misunderstood, I want to align all images of a scene at once via ransac.
---
perfect, could you now give the code snippet which prints out the percentage or amount of outlier
↪  pixels not used for ransac alignment
---
ok, then maybe also output a line with the percentage of invalid lidar pixels
---
should I make the max_trials dependent on the number of images ?
---
ok change this code such that it appends or creates to a txt file the method, scene and ransac
↪  outlier percentage with absolute numbers:
[PASTED CURRENT own_scripts/align_metric_with_ransac_lidar.py FILE]
---
Save it as a table in a markdown instead of a text file
```