Monocular Dynamic Gaussian Splatting is Fast and Brittle but Smooth Motion Helps

Yiqing Liang[1], **Mikhail Okunev**[1], **Mikaela Angelina Uy**[2], **Runfeng Li**[1], **Leonidas Guibas**[2], **James Tompkin**[1], **Adam W Harley**[2]

[1]Brown University [2]Stanford University

{yiqing_liang, mikhail_okunev, runfeng_li, james_tompkin}@brown.edu, mikacuy@stanford.edu, {guibas, aharley}@cs.stanford.edu

Abstract

Gaussian splatting methods are emerging as a popular approach for converting multi-view image data into scene representations capable of view synthesis. In particular, there is great interest in enabling view synthesis for *dynamic* scenes using only *monocular* input data—an ill-posed and challenging problem. The fast pace of work in this area has yielded multiple simultaneous claims of which methods work best, which cannot all be true. In this work we organize, benchmark, and analyze many Gaussian-splatting-based methods, providing apples-to-apples comparisons which prior work has lacked. We define the conceptual differences between the methods and quantify how these details impact performance. Empirically we find that the fast rendering speed of Gaussian-based methods appears to come at the cost of brittleness in the optimizability. The rank-ordering of methods is well-defined on synthetic data, but the complexity of real-world data currently overwhelms differences in the methods. Overall, we lay groundwork for making clearer progress in this lively domain.

1 Introduction

Every visual computing researcher and their dog seems to have published a paper on 3D Gaussian Splatting in the past year, especially for the problem of *dynamic scene* view synthesis. One problem setting with monocular cameras rather than multi-view cameras is highly ill-posed for dynamic scenes; last year's paper DyCheck [13] proposed metrics to better decide which data were truly monocular. Yet, in the face of an ill-posed problem, many papers still claim superior performance compared to their peers even with only minor methodological differences between them. This seems implausible. Further, the lack of a shared evaluation benchmark and inconsistent dataset splits make fair comparison between methods difficult if not impossible. We note that different papers even use different dataset splits to highlight only the successes of their works on particular sequences.

Our work provides a more reliable snapshot of progress on the problem of dynamic view synthesis for monocular sequences with Gaussian splatting approaches. We demonstrate that the performance of these GS-based methods heavily depends on the sequence distribution, with significantly worse results than previous work on certain datasets. To show this on sequences with reliably controlled camera and object motion, we also quantify performance across methods using a small instructive synthetic dataset. Further, we observe that many GS-based methods share almost-identical designs, with the exception of small changes in motion modeling. We consolidate these methods to fairly compare them. This reveals key factors that affect their performance, including the locality of the motion models, the propensity of some methods to overfit, and their brittleness in optimization.

Public release. Along with code and the instructive synthetic dataset, we also created segmentation masks and improved camera poses for existing datasets. These will all be released.

Preprint. Under review.

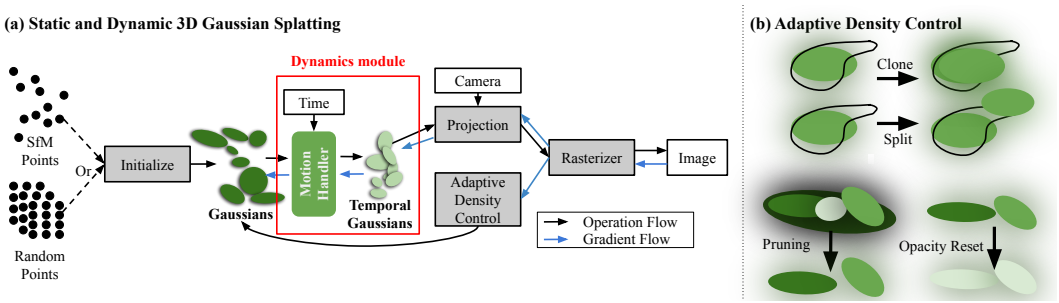

Figure 1: **Gaussian splatting with motion.** a) Overview of 3D Gaussian Splatting. Adding motion to static 3DGS typically modifies the pipeline only by adding a dynamics module, as marked by the red box. b) Adaptive densification via cloning, splitting, and pruning Gaussians to reflect scene detail. This brittle process often causes difficulties during optimization for monocular dynamic sequences.

2 Background

In this section we establish the specific background relevant to our experiments. We additionally present a more comprehensive discussion of related work in the supplementary.

2.1 3D Gaussian Splatting (3DGS)

Gaussian splatting [20] represents a scene as a set of 3D anisotropic Gaussians allowing for high-quality and fast view synthesis given only input images for a static 3D scene. Concretely, the scene is represented as a collection of 3D Gaussians with underlying attributes, where each Gaussian $G_i = (x_i, \Sigma_i, \alpha_i, c_i)$ is parameterized with mean x_i, covariance matrix Σ_i, opacity α_i, and view-dependent RGB color c_i parameterized by 2nd-order spherical harmonic coefficients. The covariance matrix Σ_i is factorized into a rotation matrix R_i derived from a unit quaternion $q_i \in \mathbb{R}^4$, and scaling vector $s_i \in \mathbb{R}^3_+$ to guarantee it to be positive semi-definite, giving $\Sigma_i = R_i \text{diag}(s_i)\text{diag}(s_i)^T R_i^T$.

These parameters are optimized directly given a set of images with known camera intrinsics K and extrinsics V via gradient descent using differentiable rasterization. To form an image, the color of a pixel (u, v) can be obtained by alpha-blending the contributions of the visible Gaussians $\{G_j\}_{j=1}^N$ sorted by depth along the camera ray for pixel (u, v):

$$C(u,v) = \sum_{j \in N} c_j \alpha_j p_j(u,v) \prod_{k=1}^{j-1} (1 - \alpha_j p_j(u,v)) \,, \tag{1}$$

where $p_j(u, v)$ is the contribution of Gaussian G_j to pixel (u, v), which is the probability density of the j-th Gaussian at pixel (u, v). To obtain $p_j(u, v)$ given camera intrinsic K and extrinsic V, the 3D Gaussian $G_j = \mathcal{N}(x_j, \Sigma_j)$ is approximated by a 2D Gaussian via linearization of the perspective transformation [59]. This gives the 2D mean $x'_j = KVx_j \in \mathbb{R}^2$ and covariance $\Sigma'_j = JV\Sigma_j V^T J^T$, where J is the Jacobian of the perspective projection.

To balance rendering quality and computational efficiency, 3DGS uses adaptive densification via periodically performing cloning, splitting, and pruning of the Gaussians (Figure 1). The optimization requires the static scene to be observed from multiple viewpoints such that 3D Gaussians can converge to the underlying 3D scene geometry even under the ambiguity from 2D projection. This makes it extremely challenging for both the monocular and dynamic setting. Further difficulty is introduced in non-Lambertian scenes where view-dependent scene elements such as mirrors or glossy objects as Gaussians can be mistakenly placed to hallucinate reflections.

2.2 Gaussian Splatting for Dynamic View Synthesis

Extending 3DGS methods to dynamic scenes from monocular input requires a prime decision in how to represent time t (or spacetime). For instance, what kind of model is used to describe Gaussian motion, whether that model applies to Gaussians individually or collectively, whether motion is offset from a single timestep or from a canonical space representing all time, or even whether to use a motion model at all ('3D+motion') or simply to represent a 4D space. Methods must also decide which Gaussian parameters to change over time, e.g., position, rotation, typically via offsets $\delta G_{i,t}$. The choice of motion design can significantly impact the expressiveness, efficiency, and robustness of the overall dynamic 3DGS system.

Table 1: **Overview of existing dynamic GS-based methods.** We organize and summarize the type, motion model and list of monocular datasets tested on by the different methods.

Method	Ref.	Type	Motion model	Monocular; tested datasets
Dynamic 3D Gaussians	[34]	Iterative	-	✗ -
3DGStream	[44]	Iterative	-	✗ -
RTGS	[55]	4D	-	✓ dnerf
Rotor-Based 4DGS	[8]	4D	-	✓ dnerf
SpaceTimeGaussians	[26]	3D+motion	Poly+RBF	✗ -
GauMesh	[51]	3D+motion	HexPlane	✗ -
GauFRe	[29]	3D+motion	MLP	✓ dnerf, hypernerf, nerfds
Deformable-GS	[54]	3D+motion	MLP	✓ dnerf, hypernerf, nerfds
4DGS	[50]	3D+motion	HexPlane	✓ dnerf, hypernerf
EffGS	[19]	3D+motion	Fourier+Poly	✓ dnerf, hypernerf
DynMF	[23]	3D+motion	Fourier+MLP	✓ dnerf, hypernerf
Gaussian-Flow	[30]	3D+motion	Fourier+Poly	✓ dnerf, hypernerf
GaGS	[33]	3D+motion	Voxel+MLP	✓ dnerf, hypernerf
E-D3DGS	[1]	3D+motion	MLP	✓ hypernerf

Motion reference frame. Iterative approaches assume that Gaussian motion over time can be optimized from some reference timestep (typically $t = 0$) in which Gaussians are already well-placed. Liuten et al. first showed multi-view dynamic 3DGS [34] by updating Gaussian parameters one frame at a time. We might define a discrete function f to query at each time t to obtain Gaussian offsets:

$$f(i, 0) = G_i \ , \qquad f(i, t) = f[i, t-1] + \delta G_{i,t} \tag{2}$$

This approach relies upon the reference frame being well reconstructed, such as from multi-view input, otherwise errors can be propagated across time (Fig. 2). Instead, canonical methods ([54, 50, 23, 29, 1, 33]) assume 3D Gaussians are offset from an embedding that represents *all* of time. This helps for monocular cameras where multi-view constraints must be formed over time. Effectively, all monocular methods use some form of canonicalization.

Figure 2: Iterative method on monocular input.

Motion complexity. Motion itself may be represented with varying complexity, defined by a function of time t. For instance, we could define Gaussian motion to be linear over time, which would only be able to describe basic motions. A piecewise-linear model could explain many motions—e.g., Luiten et al.'s Dynamic 3D Gaussian work is effectively piecewise linear [34] as Gaussian positions vary over integer timesteps. But, for monocular input, in practice authors must try to find low-dimensional 'sweet spot' functions that can constrain or smooth the ill-posed motion optimization.

Curve methods use a polynomial basis of order L to define a f over time that determines Gaussian offsets [30]. f could also use a Fourier basis [19] or a Gaussian Radial Basis [26]. Most of these typically use low order, where $L = 2$ or even $L = 1$. A Gaussian motion function could also be defined by a multi-layer perceptron (MLP); for instance, a ReLU MLP would be a piecewise-linear model with changes at arbitrary points in continuous time rather than at discrete integer times.

Motion complexity is important to reconstruction quality and computational speed. It also affects the success of optimization as it determines motion smoothness. For instance, MLP-based method are robust function fitters with self-regularization properties, but come with a higher computational burden. In our survey, we found that function complexity like the polynomial order was not always reported, making it difficult to consider these trade-offs.

Motion locality. As 3D Gaussian splatting is primitive based, many dynamic approaches define motion models also as a per-Gaussian property. This extends the parameter set of each Gaussian by the parameters of the motion model, which can be costly in terms of memory, and ignores the spatial implications of motion: that nearby primitives may move together.

Field-based approaches [52] attempt to encapsulate this idea by using a function to estimate the entire motion field. Gaussians then query this field by their position and time or by an embedding vector.

Fields can be advantageous in that the number of parameters in the motion model is independent of the number of Gaussians, and that the field can enforce smoothness between Gaussians directly by the complexity of the motion function.

One popular approach is to use field embeddings to represent both locality in space and motion over time [29, 54, 50, 33]. For instance, a continuous deformation field encoded by an MLP f_θ might take as input each Gaussian G_i's embedding z_i and t to produce an offset:

$$f_\theta(z_i, t) = \delta G_{i,t} \ , \qquad G_{i,t} = G_i + \delta G_{i,t} \tag{3}$$

As they represent volumes of continuous space, field-based approaches can exploit ideas from volume-based NeRF literature, such as voxel [33] and HexPlane [50] structures. These discrete data structures can also help to aggregate information and accelerate indexing.

Finally, intermediate approaches have begun to aggregate motion model parameters over local Gaussian neighborhoods [24]. In general, there are many schemes to model both motion complexity and locality, but it helps to consider how both space and time are represented and smoothed.

Higher dimensionality instead of motion. Another alternative instead of modelling motion together with 3D Gaussians is to define Gaussians directly in 4D space [55, 8]: mean $x_i \in \mathbb{R}^4$ and covariance matrix $\Sigma_i \in \mathbb{R}^{4\times4}$. This representation couples the space $(\mathbb{R}^3)$ and time $(\mathbb{R})$ dimensions. To optimize this representation through rendering, the 4D Gaussians are first conditioned on or projected to a given time t to obtain 3D Gaussians; then, these can be rendered and compared to the 2D images at the given timeframes. 4D methods are flexible and can describe complex dynamic scenes. However, as there are many possible projections of plausible motions within the 4D space, these models may be difficult to fit correctly with the few constraints provided by monocular input. Moreover, the representation also does not have inherent smoothness or locality constraints.

3 Evaluation Setup

3.1 Datasets

Video sequence variations can affect performance significantly within and across datasets. Scenes with small scene motion only or large camera motion relative to object motion are easier than sequences with large scene motion and small camera motion, and scenes with highly-textured objects are easier to reconstruct but require more Gaussians. To gain as much variation as possible, we collect all common datasets used across current papers. Our supplemental material provides a fuller description and examples of each, and we will briefly explain key differences here.

D-NeRF ([39]) shows synthetic objects captured by 360-orbit inward-facing cameras against a white background (8 scenes). Nerfies ([37]) (4 scenes) and HyperNeRF ([38]) (17 scenes) data contain general real-world scenes of kitchen table top actions, human faces, and outdoor animals. NeRF-DS ([53]) contains many reflective surfaces in motion, such as silver jugs or glazed ceramic plates held by human hands in indoor tabletop scenes (7 scenes). The iPhone dataset from DyCheck ([13]) (14 scenes) has large dynamic objects often undergoing large motions, with relatively small camera motions. In total, these comprise **50** scenes.

Instructive synthetic dataset. We also create a simple synthetic dataset as an instructive aid on which scene parameters cause methods to struggle. These scenes contain a textured cube, textured background wall, and a moving camera. Each scene is 60 frames. In *SlidingCube*, the cube accelerates from stationary along a straight line to cover different distances $D \in \{0, 5, 10\}$. The camera moves around the cube in an arc with a total baseline $B \in \{1, 3, 5, 10, 20\}$. The camera rotates to track the cube at all times. In *RotatingCube*, the cube additionally rotates along its vertical axis by π radians.

3.2 Metrics

For view synthesis, we use image quality metrics of peak signal-to-noise ratio (PSNR), Structural Similarity Index (SSIM) [48] and its variant Multi-Scale SSIM (MS-SSIM) [49], and learned perceptual metric LPIPS [58] using the AlexNet-trained features. We also report training and rendering time in seconds. Existing research typically reports a metric as a mean average over sequences with one optimization attempt; we also report the standard deviation of three independent optimizations. To fairly compare train time and rendering speed, we use Nvidia GeForce RTX 3090 cards.

Static vs. dynamic. Static scene areas imaged by a moving camera may have enough constraints for reconstruction, and if these areas are large then they will dominate the metric. So, to demonstrate reconstruction quality on dynamic regions only, we compute per-frame binary dynamic masks using SAM-Track [5] and report masked metrics mPSNR, mSSIM, mMS-SSIM and mLPIPS.

3.3 Methods

We compare all works thus far, e.g., in Table 1. that span different methods, apply to monocular input, and had official code (as of June 2024, time of paper writing).

1. Local low-order poly: **EffGS** [19] uses a 2-order fourier basis and a 1-order polynomial basis per Gaussian to model motion.
2. Local low-order RBF: **SpaceTimeGaussians** [26] or **STG** uses a radial basis function, 3-order and 1-order polynomial basis per Gaussian to model motion. This method uses a small MLP to decode color; we remove this for fairer comparison and label it STG-decoder.
3. Field MLP: **DeformableGS** [54] uses an MLP to estimate Gaussian offsets from a deformation field; the MLP is shared among all Gaussians.
4. Field HexPlane: **4DGS** [50] discretizes and factorizes the field with a HexPlane [3]; this is shared among all Gaussians.
5. 4D Gaussian: **RTGS** [55]; this has no explicit motion model.

To minimize differences in implementation except for the motion model, we integrate these algorithms into a single codebase. For hyperparameters, we follow the original works closely, e.g., we use separate hyperparameters for synthetic and real-world scenes (see supplemental material).

Baselines: We also evaluate **3DGS** without any motion, as adding a motion model might make results worse. We also include **TiNeuVox** [10] as a comparison point for what a voxel feature grid method can accomplish on dynamic scenes, even if it cannot be rendered quickly.

4 Results

We present our main findings and include full details in the supplemental material. To help frame our findings, we note that our implementations approximately match the results of the original works on their reported datasets. We also note that most dynamic Gaussian methods do not report performance across all datasets, and no dynamic Gaussian method has yet reported performance metrics on the iPhone dataset, so the comparisons made possible by this full-slate evaluation are of special interest.

4.1 Finding 1: Gaussian Methods Struggle In Comparison to a Hybrid Neural Field

We begin our analysis by evaluating all methods on all datasets. We present a summary of the findings, averaged across 5 datasets, in Table 2. The full results are available in supplementary.

Table 2: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across all five datasets.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	**24.54**	<u>0.706</u>	**0.804**	0.349	0.29	**2664.89**
3DGS	19.48	0.651	0.688	0.358	**243.47**	<u>2964.81</u>
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	<u>181.70</u>	5359.56
DeformableGS	<u>24.07</u>	0.694	0.755	<u>0.283</u>	20.20	6227.43
4DGS	23.55	**0.708**	<u>0.765</u>	**0.277**	62.99	8628.89
RTGS	21.61	0.663	0.720	0.350	143.37	7352.52

From this summary, we can make the broad and sobering observation that TiNeuVox, a non-Gaussian method, regularly outperforms all Gaussian methods in image quality. TiNeuVox also trains quickly and converges reliably, as compared to the Gaussian methods. The main drawback of TiNeuVox is simply its rendering time, operating at 0.3 FPS compared to 20–200 FPS for the Gaussian methods. The speed difference is due to the rasterization in 3DGS versus volume rendering in NeRF.

4.2 Finding 2: Low-dimensional motion representations help

Comparing motion-based and 4D representations in Table 2, it appears that local, low-dimensional representations of motion perform better than less-constrained systems.

Motion locality helps quality. Comparing Gaussian methods on the LPIPS metric, we can observe that the field-based methods (DeformableGS & 4DGS) perform better than the methods which attach motions to individual Gaussians (EffGS & STGs) in rendering quality.

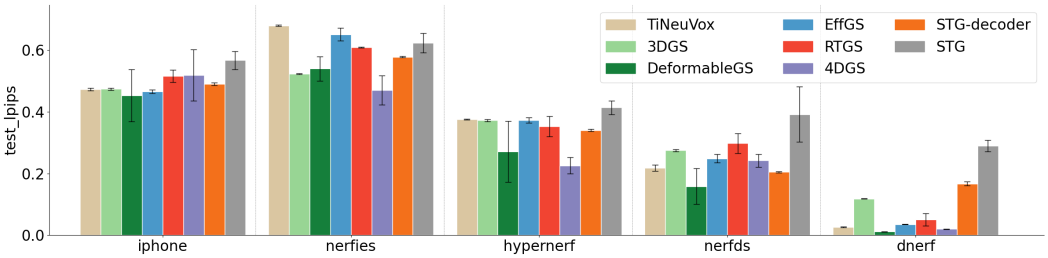

Figure 4: **Per-dataset Quantitative Results.** Figure shows the test set LPIPS along with error bars for all methods on each of the different datasets. Note that lower is better. We see that the datasets have different winning methods.

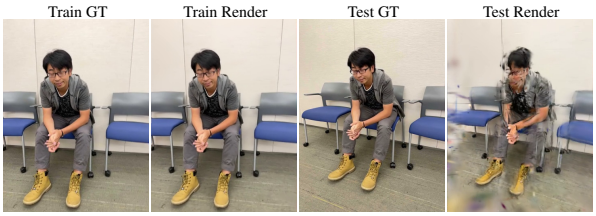

Figure 3: RTGS overfits to training views

Motion representation complexity hurts efficiency. Comparing methods on training time and rendering speed, we find that basis-based methods (EffGS & STGs) are faster to train and faster to render than MLP-based methods (DeformableGS & 4DGS).

Going to 4D makes things worse. The 4D Gaussian approach, which is the most expressive representation of spacetime, performs worst in both quality and efficiency. This is consistent with our expectations established in Section 2.1. We qualitatively demonstrate this effect in Figure 3.

4.3 Finding 3: Dataset Variations Overwhelm Gaussian Method Variations

Inspecting the results of Gaussian methods *across* datasets, we find, contrary to the claims in the individual works, that a rank-ordering of the methods is unclear. We plot the per-dataset performance metrics in Figure **??**, and note that the datasets have different winning methods. We observe that 4DGS and DeformableGS perform relatively well, but their performance across datasets varies greatly.

4.4 Finding 4: Adaptive Density Control Causes Headaches

As discussed in Section 2.1, a key detail of Gaussian Splatting methods is adaptive density control: Gaussians are added and removed during optimization, according to carefully tuned heuristics, allowing the complexity of the representation to adapt to the complexity of the data. The total number of Gaussians after optimization varies greatly across scenes (Figure 5). While this feature adds expressivity to the overall representation, we find that it causes noticeable brittleness, such that (1) the efficiency (in both optimization and rendering) can shift undesirably across scenes, (2) the risk of overfitting increases, and (3) optimization occasionally fails completely. These factors may explain why some prior works tune the number of Gaussians and the adaptive density settings on a per-scene basis [20].

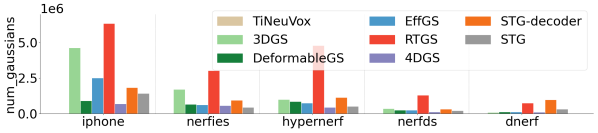

Figure 5: **# Gaussians** after optimization for each dynamic Gaussian method as a result of the adaptive densification.

Training and rendering efficiency varies greatly across scenes. The optimization time for Gaussian Splatting methods varies greatly across scenes, with certain scenes consistently optimizing quickly and others optimizing slowly, across all methods. Further, the methods take significantly more time to reconstruct real-world datasets (HyperNeRF, iPhone) compared to the synthetic datasets. This factor is partially explained by the **frequency content** of the data (Figure 6). Scenes with a predominance of higher frequencies take longer to optimize. For this analysis, we use a Fast Fourier Transform (FFT) to transform each frame to the frequency domain, and aggregate the magnitude spectrum from all images to calculate the mean frequency.

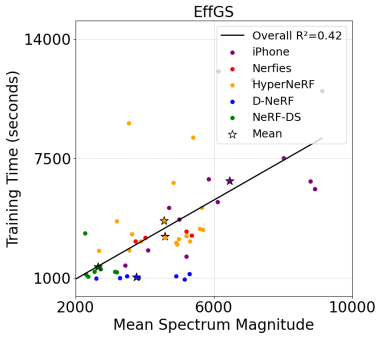

Figure 6: **Convergence-frequency**

Density adaptation makes overfitting worse. In Figure 7, we show the train-test performance gap for all Dynamic Gaussian methods, along with 3DGS and TiNeuVox. For this evaluation, we

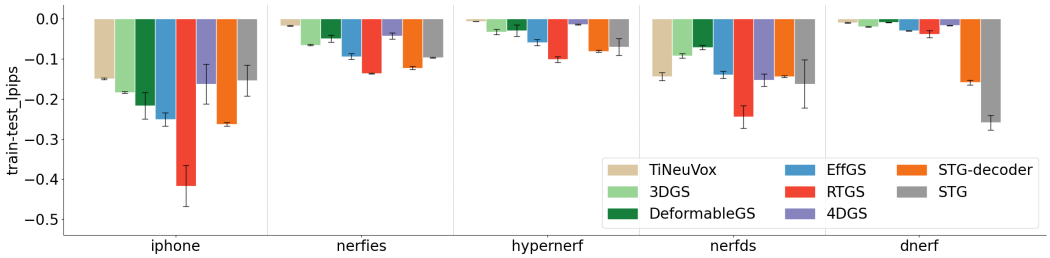

Figure 7: **Train-Test Performance Gaps.** We show the difference between the average LPIPS on the train and test set, where a larger gap indicates more overfitting to the training sequence. Note that here larger negative values indicate more severe overfitting.

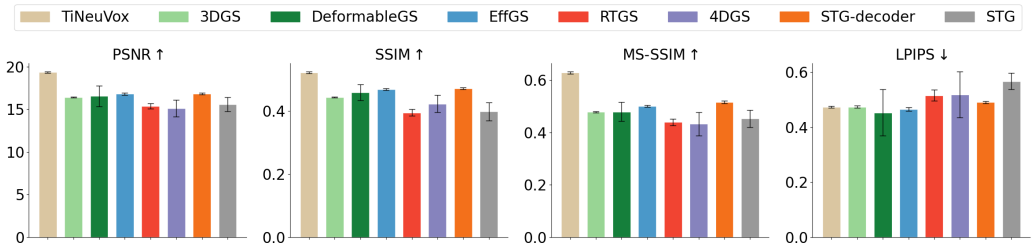

Figure 9: **Strictly-Monocular Dataset.** Test metrics of the different methods on the strictly-monocular **iPhone** dataset.

calculate LPIPs metrics for train and test set of the same sequence, and then substract test set metric results from train set metrics. Larger quantities in this evaluation indicate wider train-test gaps, and serve as an indicator for overfitting. The evaluation reveals that the methods with adaptive density (i.e., all methods except TiNeuVox) have consistently wider train-test gaps. The only exception is in the NeRF-DS dataset, where we hypothesize that the reflective objects in that data cause issues for all methods, flattening the performance disparities between them.

Density adaptation risks total failure. Density changes can occasionally result in catastrophic failures with empty scene reconstructions (Figure 8). Gaussians may be pruned due to their opacities being under a hand-chosen threshold, and once a substantial number are deleted then it is difficult to recover the scene structure by subsequent cloning and splitting. We find that this happens unpredictably; we exclude these runs from our evaluation statistics.

Figure 8: **Instability.** Training frame, and renderings from 3 runs of DeformableGS [54].

4.5 Finding 5: Lack of Multi-View Cues Hurts Dynamic Gaussians More

Most datasets used for monocular dynamic view synthesis consist of data where there is a slow-moving scene captured by a rapidly-moving camera. This circumstance allows methods to leverage multi-view cues for optimization [13]. In contrast, the **iPhone** dataset aims to evaluate methods on "strictly-monocular" scenarios, meaning that the camera is moving more naturally. Figure 9 quantifies all methods' performance on this dataset. Excepting for the perceptual metric **LPIPS**, all Dynamic Gaussians perform worse than **TiNeuVox** by a large margin. This is consistent with our finding on overfitting: since the Gaussian methods are more susceptible to overfitting, they are also more likely to do poorly with data lacking Multi-View Clues.

4.6 Finding 6: Narrow Baselines and Fast Objects Cause Error

Camera and object motion can both influence reconstruction performance considerably [13]. This is related to the amount of multi-view information available to constrain optimization, but can be studied in detail with the help of our instructive synthetic dataset. Figure 11-left shows a high-level overview of the performance of all methods, across different camera baselines and different motion

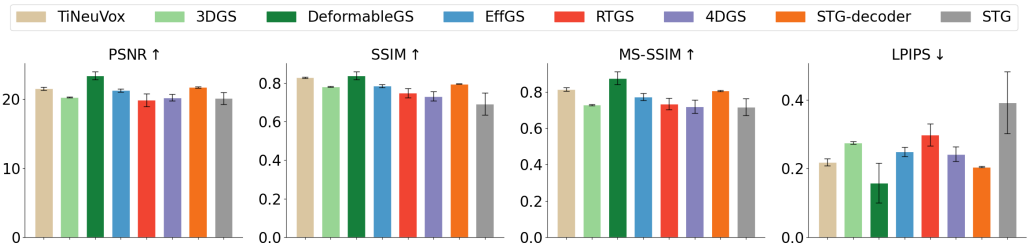

Figure 10: **Specular Dataset.** Test results on **NeRF-DS**, which includes reflective objects.

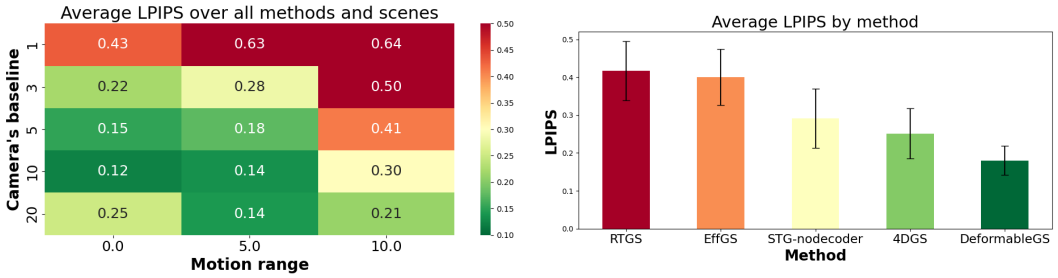

Figure 11: **Results on our instructive synthetic dataset.** *Left*: LPIPS heatmap (smaller is better) shows how baseline scale and object motion range affect performance across all methods on average. Decreasing object motion range (right to left) affects reconstruction performance positively; decreasing camera baseline (bottom to top) has the same effect. Values over 0.3 usually represent a failure to reconstruct the dynamic object. *Right*: Ranking of methods by average LPIPS over all scenes.

ranges. As the camera's baseline decreases, and/or as the object motion increases, reconstructions degrade. This result also holds for each method **individually** (see supplemental).

Figure 11-right summarizes LPIPs across methods and scenes in our instructive synthetic dataset. Here, unlike in the complex real-world datasets (Table 2), we see a clear ranking emerge between methods. We find that DeformableGS is the most reliable here, and RTGS the least reliable. We also note that RTGS occasionally crashes during optimization, due to an out-of-memory error triggered as a side-effect of the small-baseline setting. In sum, we find that all methods are sensitive to camera baseline and the magnitude of the underlying motion, although the levels of robustness vary.

4.7 Finding 7: Specular Objects are a Challenge for All Methods

Yang et al. [54] highlighted that their **DeformableGS** method outpeforms previous state-of-art methods on **NeRF-DS** dataset, including **TiNeuVox**. As NeRF-DS focuses on specular objects, this might imply that other Gaussian methods are able to handle specular objects well. However, we found that this is not the case (Figure 10). Generally, it is difficult for methods to distinguish between specular effects and small object motions especially under small baselines.

4.8 Finding 8: Foreground/Background Separation Clarifies Static/Dynamic Results

Inspecting the 3DGS performance in table 2, we note that it is performing surprisingly well, considering that it is a static scene representation. This is partly because the moving foreground constitutes a small fraction of the pixels in the given dataset sequences. We also find that 3DGS is able to smuggle a pseudo-dynamic scene into its representation, by inserting Gaussians which only render from certain viewpoints (and therefore only certain timesteps), and thereby reduce loss on dynamic parts of the training images. Please see the supplementary material for visualizations of this effect.

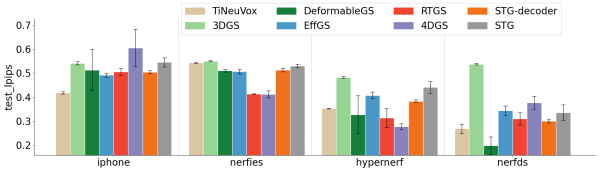

Figure 12: **Foreground-Only LPIPs Evaluation (↓).**

To better evaluate Dynamic Gaussians' advantage over 3DGS, we evaluate mLPIPS, mPSNR, mSSIM and mMS-SSIM (fig. 12). Comparing Figure 4 and Figure 12, we see that Dynamic Gaussians' performance does not change much after masking, but 3DGS's performance worsens dramatically, suggesting that the dynamic scene representations are indeed better at capturing the moving part of the scene.

5 Conclusion

With the emerging popularity of Gaussian splatting, multiple works were recently introduced simultaneously that tackle the challenging setting of dynamic scene view synthesis using only monocular input, each claiming superior performance even with only minor methodological differences and inconsistent evaluation settings. In this work, we organize, benchmark and analyze many of these Gaussian splatting-based methods and provide a shared, consistent, apples-to-apples comparison between them. We also define conceptual differences between these methods and analyze their impact on a variety of performance metrics. We consolidate these methods and benchmark their performance on instructive synthetic data and complex real-world data. This work may lead to broader societal impact in areas such as video editing, visual art, and 3D scene analysis for industrial applications.

Limitations. Evaluations such as ours would benefit from being able to describe each sequence's complexity of reconstruction independently of any method's individual performance upon that sequence. With this, it would be simpler to find insights by correlating methods with scene difficulty. We do not do this; past works have proposed some approaches (e.g., DyCheck's Ω and ω [13]), but this task is a chicken-and-egg problem: it invariably requires estimating scene geometry and motion, which is our initial problem. Similarly, ground-truth evaluation of reconstructions such as for depth or motion would also aid in this task; capturing ground truth is difficult for real world dynamic scenes.

6 Additional Results

Please download the webpage and check index.html for qualitative results on both existing datasets and our instructive dataset.

6.1 Quantitative Results Per Dataset

Table 3: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across D-NeRF dataset.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	33.03	0.97	0.99	0.03	0.42	2582.54
3DGS	20.64	0.92	0.88	0.12	**340.85**	**928.83**
EffGS	30.52	0.96	0.98	0.04	289.47	1042.04
STG-decoder	25.89	0.91	0.90	0.17	160.32	3462.29
STG	17.09	0.88	0.66	0.29	208.98	4889.50
DeformableGS	**37.14**	**0.99**	**0.99**	**0.01**	50.78	2048.38
4DGS	33.27	0.98	0.99	0.02	134.13	1781.46
RTGS	28.78	0.96	0.96	0.05	192.37	1519.60

Table 4: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across HyperNeRF dataset.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	**26.45**	0.74	0.88	0.38	0.12	2604.45
3DGS	20.98	0.69	0.76	0.37	**244.87**	**2587.02**
EffGS	22.24	0.70	0.79	0.37	138.79	4119.66
STG-decoder	23.92	0.73	0.83	0.34	66.72	7423.41
STG	22.92	0.70	0.78	0.41	183.21	5729.80
DeformableGS	24.58	0.74	0.83	0.27	10.91	8855.46
4DGS	25.70	**0.79**	**0.89**	**0.23**	37.46	10170.86
RTGS	22.99	0.71	0.79	0.35	104.64	11507.80

Table 5: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across NeRF-DS dataset.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	21.54	0.83	0.81	0.22	0.49	2696.48
3DGS	20.29	0.78	0.73	0.28	**353.99**	**1066.48**
EffGS	21.28	0.78	0.77	0.25	307.70	1597.90
STG-decoder	21.73	0.80	0.81	0.20	212.24	2131.48
STG	20.13	0.69	0.72	0.39	302.89	2214.62
DeformableGS	**23.42**	**0.84**	**0.88**	**0.16**	30.27	2885.07
4DGS	20.25	0.73	0.72	0.24	100.22	4075.43
RTGS	19.88	0.75	0.73	0.30	259.83	3116.21

Table 6: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across Nerfies dataset.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	**22.89**	0.45	<u>0.71</u>	0.68	0.11	<u>3145.67</u>
3DGS	20.14	0.46	0.66	<u>0.52</u>	**209.80**	**2823.33**
EffGS	20.13	0.43	0.61	0.65	<u>159.17</u>	3249.33
STG-decoder	20.93	<u>0.47</u>	0.67	0.58	90.46	6050.42
STG	20.55	0.46	0.66	0.62	142.30	5264.58
DeformableGS	21.26	0.43	0.66	0.54	14.99	6950.94
4DGS	<u>21.84</u>	**0.50**	**0.72**	**0.47**	35.70	9797.50
RTGS	20.06	0.42	0.62	0.61	153.38	9059.54

Table 7: **Summary of Quantitative Results.** Table shows a summarized quantitative evaluation of all methods averaged across iPhone dataset.

Method\Metric	PSNR↑	SSIM↑	MS-SSIM↑	LPIPS↓	FPS↑	TrainTime (s)↓
TiNeuVox	**19.35**	**0.52**	**0.63**	0.47	0.36	**2632.17**
3DGS	16.41	0.44	0.48	0.47	**140.50**	<u>5777.45</u>
EffGS	16.82	0.47	0.50	<u>0.47</u>	99.64	6275.33
STG-decoder	<u>16.85</u>	<u>0.47</u>	<u>0.52</u>	0.49	86.18	7694.83
STG	15.60	0.40	0.45	0.57	<u>114.95</u>	6629.64
DeformableGS	16.56	0.46	0.48	**0.45**	10.47	7278.28
4DGS	15.13	0.42	0.43	0.52	42.53	14205.48
RTGS	15.36	0.39	0.44	0.52	101.33	8484.05

6.2 Quantitative Result Plots Per Metric

6.2.1 PSNR

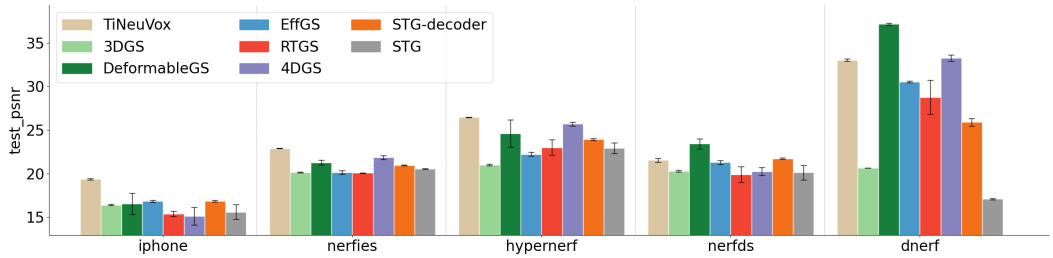

Figure 13: **Per-dataset Quantitative Results.** Test set PSNR along with error bars for all methods on each of the datasets. (↑). We see that the datasets have different winning methods.

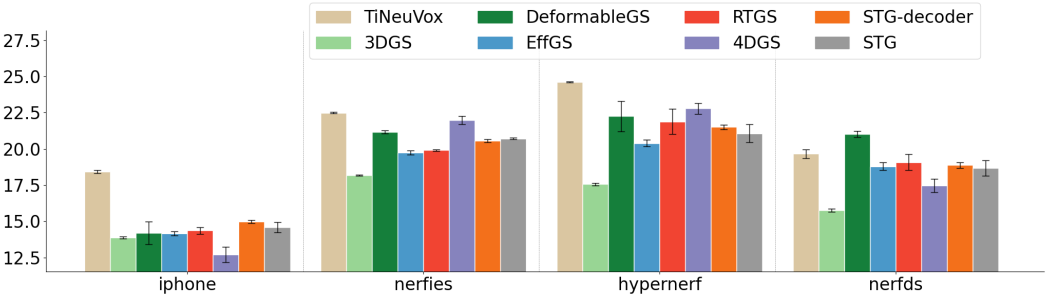

Figure 14: **Foreground-Only PSNRs Evaluation (↑).**

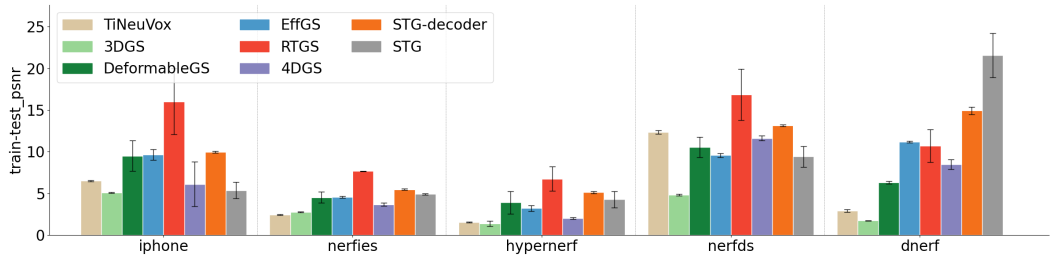

Figure 15: **Train-Test Performance Gaps.** We show the difference between the average PSNR on the train and test set, where a larger gap indicates more overfitting to the training sequence.

6.2.2 SSIM

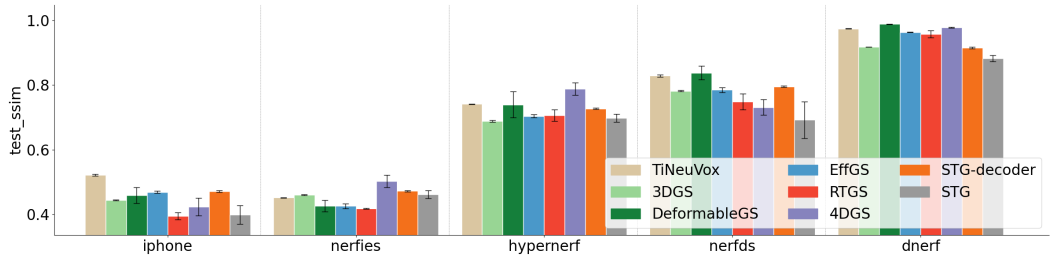

Figure 16: **Per-dataset Quantitative Results.** Test set SSIM along with error bars for all methods on each of the datasets. (↑). We see that the datasets have different winning methods.

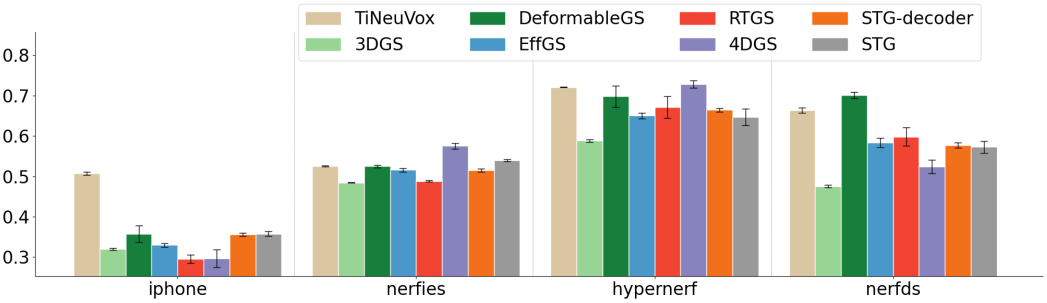

Figure 17: **Foreground-Only SSIMs Evaluation (↑).**

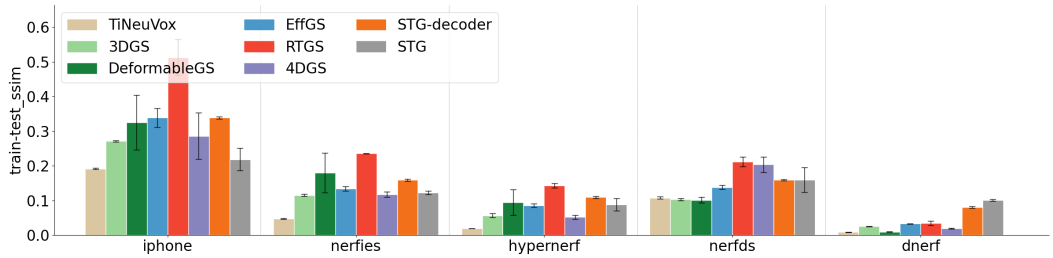

Figure 18: **Train-Test Performance Gaps.** We show the difference between the average SSIM on the train and test set, where a larger gap indicates more overfitting to the training sequence.

6.2.3 MS-SSIM

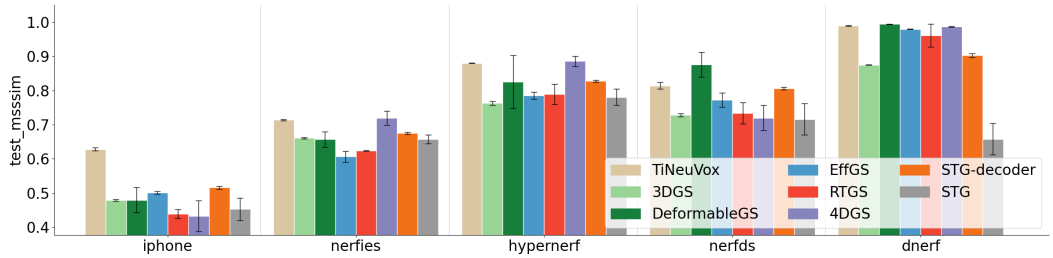

Figure 19: **Per-dataset Quantitative Results.** Figure shows the test set MS-SSIM along with error bars for all methods on each of the different datasets. Note that higher is better. We see that the datasets have different winning methods.

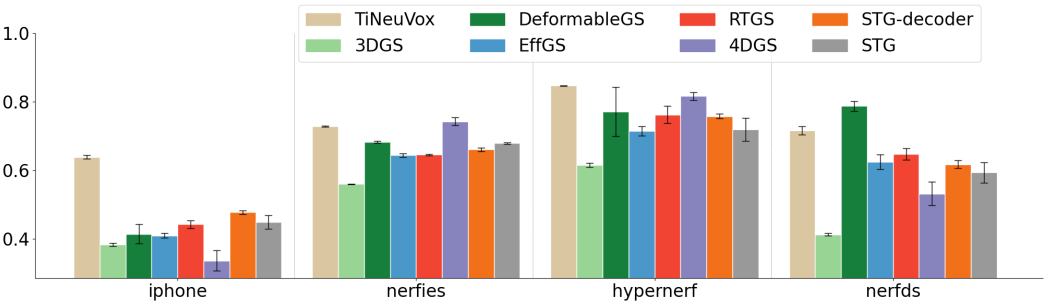

Figure 20: **Foreground-Only MS-SSIMs Evaluation (↑).**

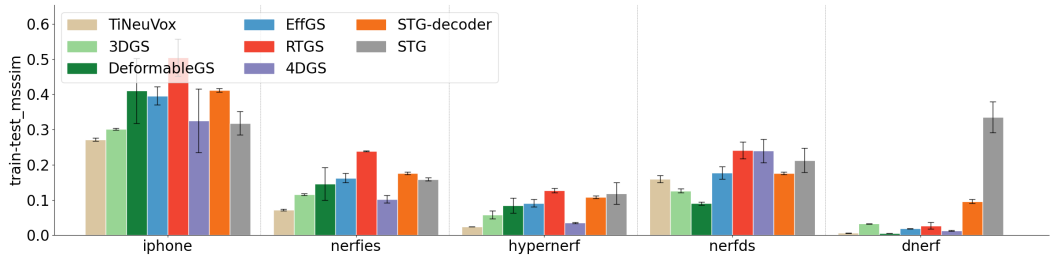

Figure 21: **Train-Test Performance Gaps.** We show the difference between the average MS-SSIM on the train and test set, where a larger gap indicates more overfitting to the training sequence.

6.2.4 Train Time and FPS

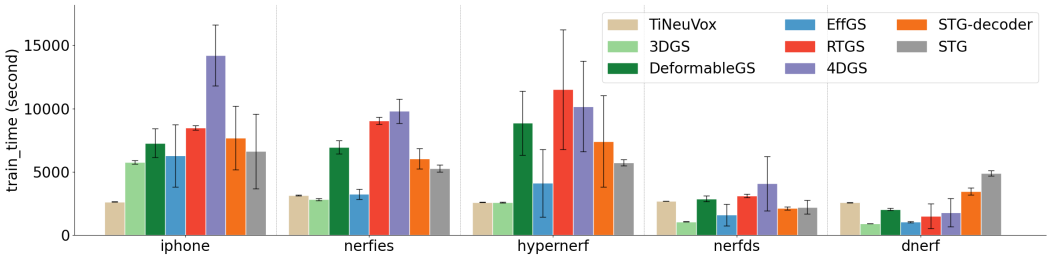

Figure 22: **Per-dataset Quantitative Results.** Figure shows the test set Train Time along with error bars for all methods on each of the different datasets. Note that lower is better. We see that the datasets have different winning methods.

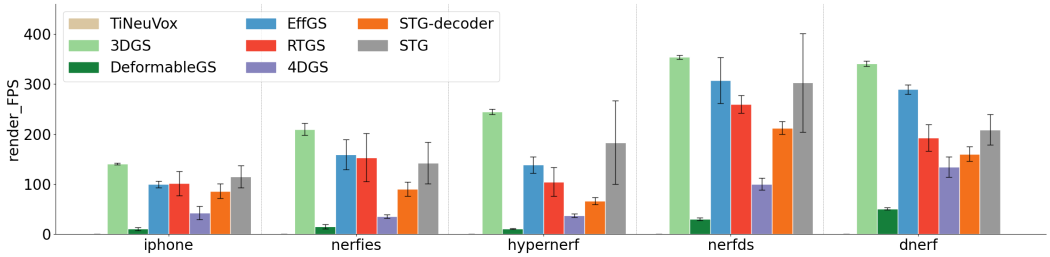

Figure 23: **Per-dataset Quantitative Results.** Figure shows the test set render FPS along with error bars for all methods on each of the different datasets. Note that higher is better. We see that the datasets have different winning methods.

6.3 Convergence-Frequency Relationships Per Method

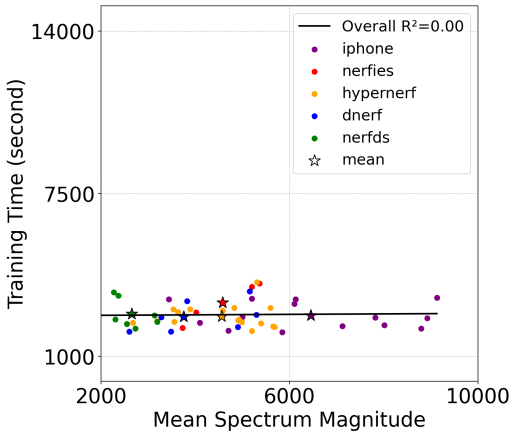

Figure 24: **Convergence-frequency** relationship for TiNeuVox.

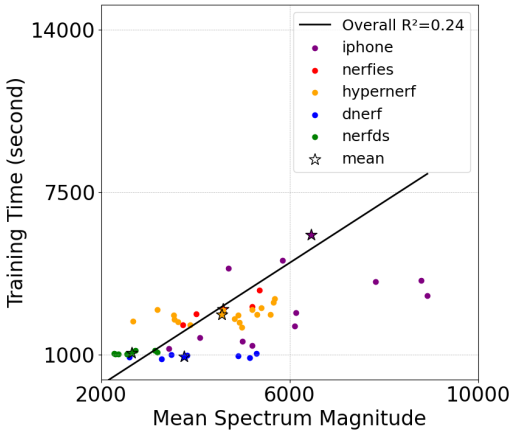

Figure 25: **Convergence-frequency** relationship for 3DGS.

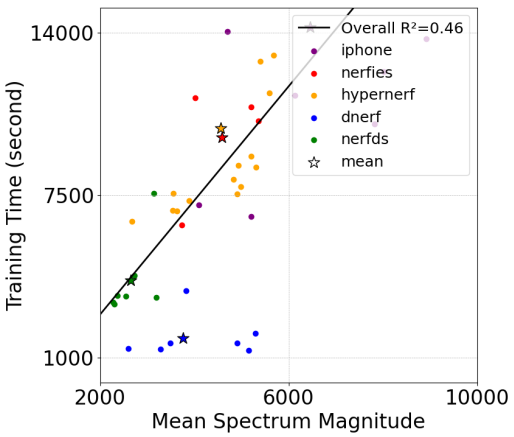

Figure 26: **Convergence-frequency** relationship for 4DGS.

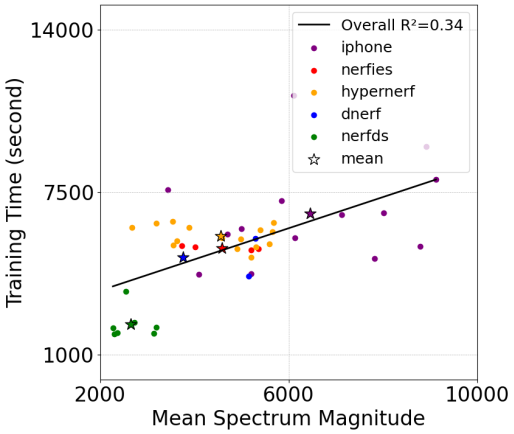

Figure 27: **Convergence-frequency** relationship for STG.

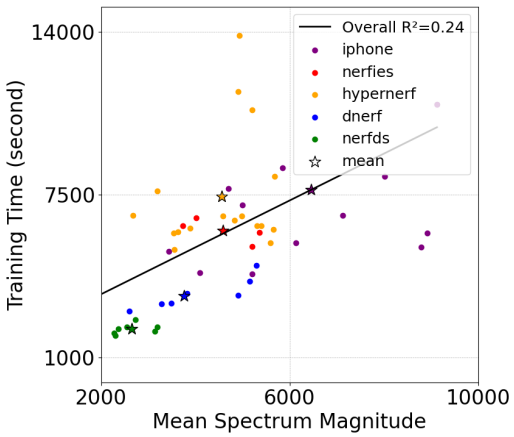

Figure 28: **Convergence-frequency** relationship for STG-decoder.

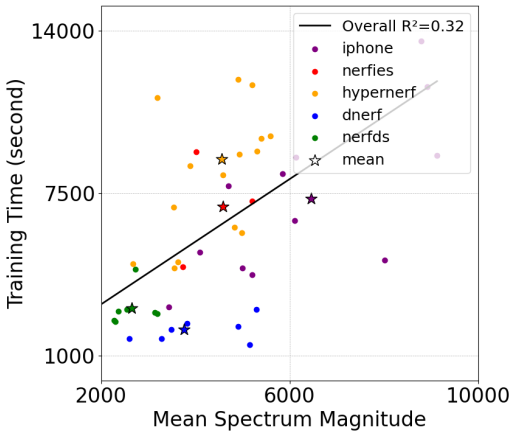

Figure 29: **Convergence-frequency** relationship for DeformableGS.

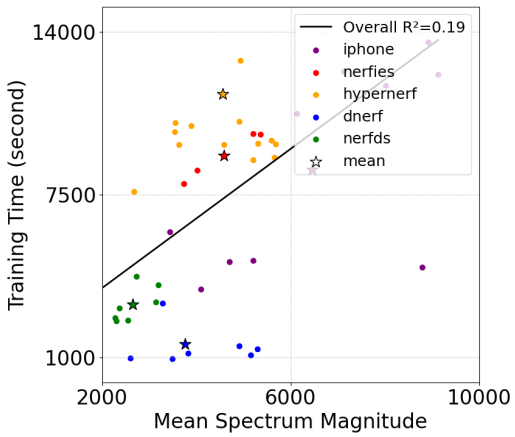

Figure 30: **Convergence-frequency** relationship for RTGS.

6.4 Ablations for Instructive Dataset

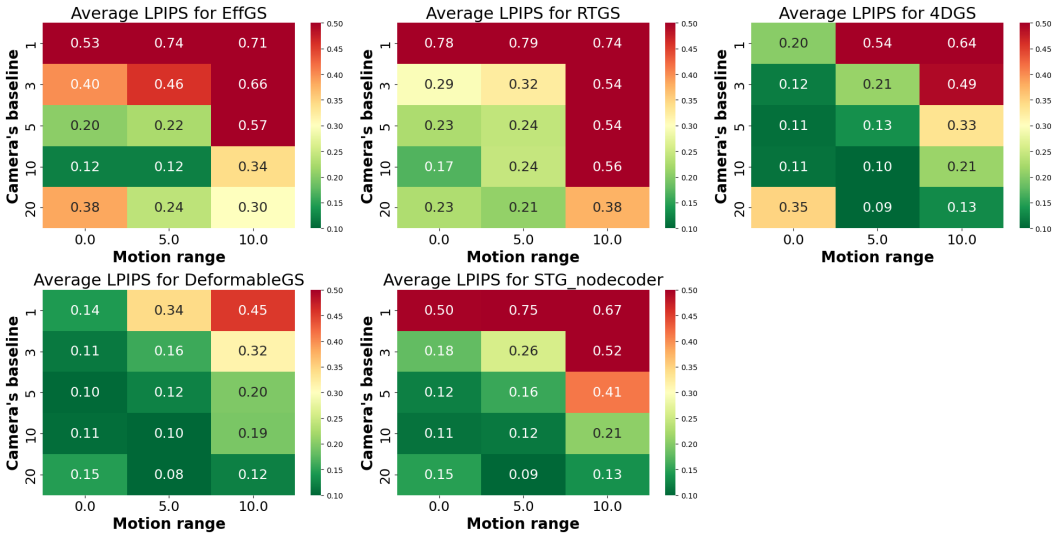

Figure 31: **Individual ablations for the instructive dataset** The figure shows camera baseline and motion range ablations for each method separately using LPIPS↓ metric. On average the reconstruction becomes harder with increase of the motion range (left to right) and with decrease of the camera's baseline (bottom to top).

7 Additional Data-related Details

Our work shows a greater set of comparisons than existing works as they typically only report on a subset of the data by selection (Table 8).

Table 8: Previous works typically only report on a subset of data.

Method		D-NeRF	Nerfies	HyperNeRF	NeRF-DS	iPhone
RTGS	[55]	✓	✗	✗	✗	✗
	Ours	✓	✓	✓	✓	✓
DeformableGS	[54]	✓	✗	✗	✓	✗
	Ours	✓	✓	✓	✓	✓
4DGS	[50]	✓	✗	4 of 13	✗	✗
	Ours	✓	✓	✓	✓	✓
EffGS	[19]	✓	✗	4 of 13	✗	✗
	Ours	✓	✓	✓	✓	✓
STG	[26]	✗	✗	✗	✗	✗
	Ours	✓	✓	✓	✓	✓

7.1 Dataset Descriptions

D-NeRF ([39]) is a synthetic dataset containing eight single-object scenes captured by 360-orbit inward-facing cameras. Test views are rendered at unseen angles. The test set is misaligned in the *lego* sequence, and so we use *lego*'s validation set for testing instead following [54].

Nerfies ([37]) and HyperNeRF ([38]) contain general real-world scenes of kitchen table top actions, human faces, and outdoor animals. Thus far, most methods [50, 10] report results on four sequences from the 'vrig' split of HyperNeRF dataset that uses a second test camera rigidly-attached to the first, rather than the 'interp' split to render held-out frames from a single camera (an easier task). Some works report their own *sequence* splits from HyperNeRF [23]. We include all 17 HyperNeRF sequences and 4 Nerfies sequences unless otherwise noted.

The HyperNeRF data has known bad camera poses. To quantify the camera pose error's effect on reconstruction quality, we improve the camera poses in HyperNeRF by segmenting out moving objects and highly specular regions and rerunning COLMAP; see Section 7.2.

NeRF-DS ([53]) contains many reflective surfaces in motion, such as silver jugs or glazed ceramic plates held by human hands in indoor tabletop scenes. This contains seven sequences. Test sequences are again generated by a second rigidly-attached camera. Lastly, the iPhone ([13]) dataset's 14 scenes include large dynamic objects often undergoing large motions, with relatively small camera motions. Test views are from two static witness cameras with a large baseline difference.

We do not include the NVIDIA Dynamic Scene dataset [56] because they sub-sample fake monocular camera sequences from a 12-camera wide baseline setup. This creates an unrealistically large degree of scene motion between frames.

7.2 Erroneous Camera Pose

Camera pose is required as input for most dynamic Gaussians methods for monocular videos. However, it's difficult to measure precise poses with real-world dynamic videos, leading to pose errors. Erroneous camera poses can be detected by simply applying static Gaussian Splatting [20] to dynamic frames. If we observe blurry static background renderings as output, then this indicates erroneous camera poses.

HyperNeRF [38] is a dataset that suffers badly from camera pose inaccuracy, as dynamic regions were not masked for COLMAP [41] during their camera pose estimation. To address this problem, we use SAM-Track [5] plus human labor to mask dynamic, reflective and textureless regions, and

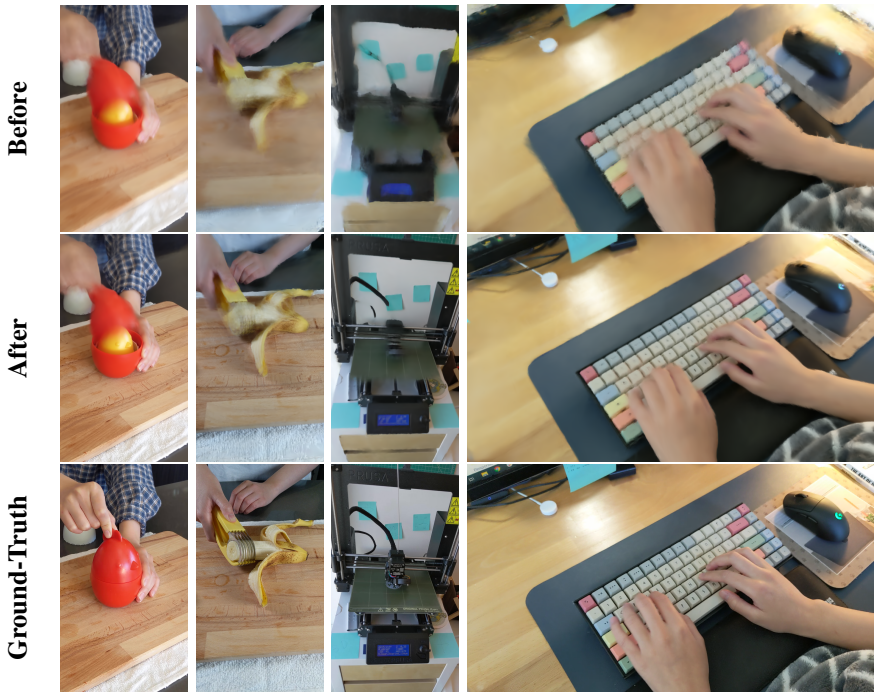

Figure 32: Comparison of ground-truth and static Gaussian Splatting [20] results on before and after correcting the camera poses for HyperNeRF [38] scenes (vrig-chicken, slice-banana, vrig-3dprinter, and keyboard). link

then rerun COLMAP with these masks and customized arguments for each scene. Figure 32 shows improved static Gaussian rendering in background areas. Out of HyperNeRF's 17 scenes, we improve the camera poses in 7 scenes and slightly improve them in 3 scenes. We evaluate the pose quality by reporting mPSNR of static 3DGS on static regions, using the inverse of our motion masks, as summarized in Table 9.

We use **No Diff/Worse** group's 7 sequences from fixed and corresponding 7 sequences from original HyperNeRF dataset to show the effect of camera pose inaccuracy on reconstruction (Figures 33, 34, 35, 36). Performance is improved after pose fixes across for **EffGS**, **STG**s and **RTGS**. But, even though static regions are now deblurred in the standard 3DGS rendering showing improved poses, surprisingly not all method improve: **TiNeuVox**, **DeformableGS**, and **4DGS** instead often become worse. We are yet to explain this phenomenon.

Result	Ratio Used	Scene	Original Poses mPSNR	Corrected Poses mPSNR
Better	2x	chickchicken	22.290	23.503
	2x	cut-lemon1	24.053	24.809
	1x	keyboard	22.952	24.831
	2x	slice-banana	22.794	24.690
	4x	tamping	20.817	21.671
	1x	vrig-chicken	23.334	26.075
	1x	vrig-3dprinter	17.685	22.118
Slightly Better	2x	aleks-teapot	22.343	22.543
	1x	broom2	21.987	22.810
	2x	cross-hands1	19.552	19.987
No Diff/Worse	1x	americano	24.348	24.418
	1x	espresso	22.707	22.291
	2x	hand1-dense-v2	25.547	25.214
	1x	oven-mitts	25.565	19.891
	1x	split-cookie	24.389	24.277
	2x	torchocolate	23.345	18.356
	2x	vrig-peel-banana	23.427	22.251

Table 9: Static region masked PSNR results for HyperNeRF scenes before and after correcting the camera poses. link

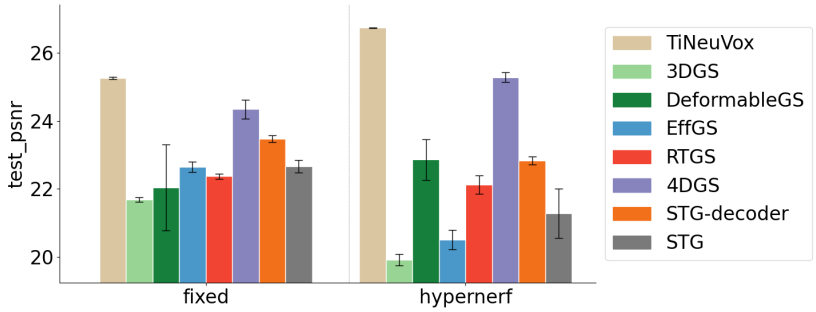

Figure 33: PSNR Change on the same set of HyperNeRF sequences to show Camera inaccuracy's effect on reconstruction. Higher is better.

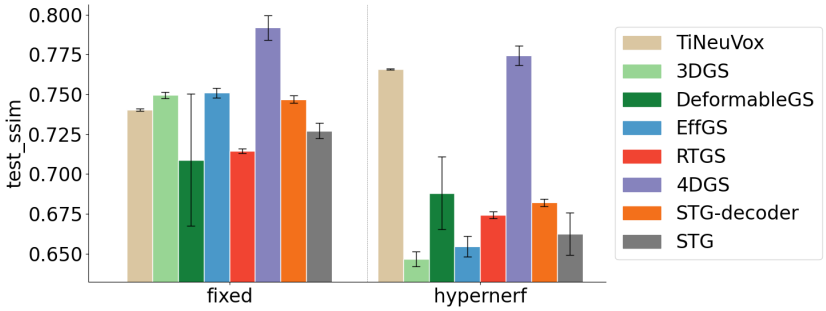

Figure 34: SSIM Change on the same set of HyperNeRF sequences to show Camera inaccuracy's effect on reconstruction. Higher is better.

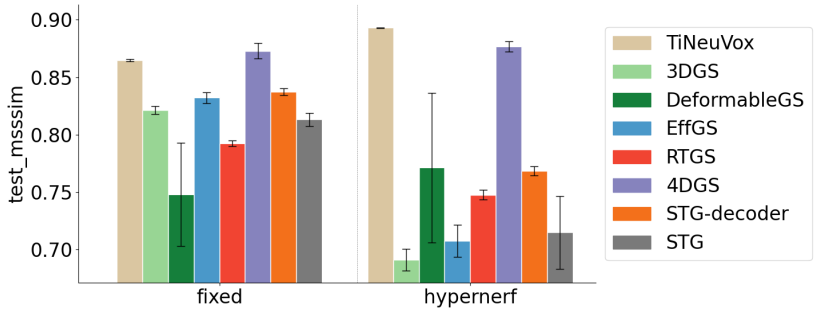

Figure 35: MS-SSIM Change on the same set of HyperNeRF sequences to show Camera inaccuracy's effect on reconstruction. Higher is better.

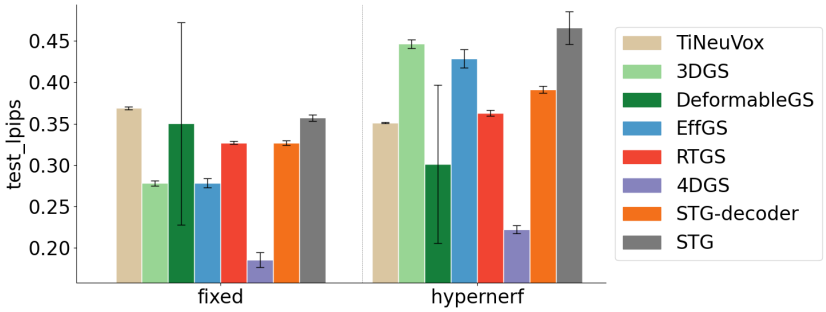

Figure 36: LPIPS Change on the same set of HyperNeRF sequences to show Camera inaccuracy's effect on reconstruction. Lower is better.

8 Experimental Implementation

8.1 Shared Hyperparameters

Most of GS-related hyperparameters are inherited from 3DGS paper, and identical among algorithms with objective function (Equation (4)):

$$\mathcal{L} = \|I_{\text{pred}} - I_{\text{gt}}\| + \lambda_{\text{SSIM}} \cdot (1 - \text{SSIM}(I_{\text{pred}}, I_{\text{gt}})), \lambda_{\text{SSIM}} = 0.2 \tag{4}$$

For learning rates, people set spherical harmonics feature learning rate as 0.0025, opacity learning rate as 0.05, scaling learning rate as 0.005, rotation learning rate as 0.001, position learning rate as 0.00016 that exponentially decrease to 0.0000016 with learning rate delay 0.01, after 30000 iterations.

Gaussian densification starts from 500th training iteration, and ends until 15000th training iteration. Densification gradient threshold is set to 0.0002. The spherical harmonics (SH) degree is set to be 3 in 3DGS, percent dense is set to be 0.01.

To align with dynamic Gaussian works' implementation, a few hyperparameter might be different from 3DGS default setting:

Batch size 3DGS's default setting is to use batch size 1 during training. **RTGS** uses 4 for real-world scenes, **4DGS** uses 2 for real-world scenes, and **SpaceTimeGaussians** use 2 for all scenes.

Warm up stage **DeformableGS**, **4DGS** and **EffGS** first fix the motion component, and train the static part for the first 3000 steps.

Opacity reset During training, 3DGS set G_i's opacity σ_i to 0 periodically every 3000 iterations during densification phase. **4DGS** disable this opacity reset action when dealing with real-world sequences.

Initialization Original 3DGS suggests initialize G_is with $100,000$ random points uniformly sampled in space for synthetic scenes, and with structure-from-motion (SfM) [41] point cloud for real-world scenes. **RTGS** suggests appending extra uniformly-sampled random point cloud to SfM point cloud for a multi-view reconstruction dataset [25]. **EffGS** uses random point cloud in the place of SfM point cloud for the same dataset. We follow 3DGS's suggestion.

Apart from shared hyperparameters, each motion model also introduces extra hyperparameters.

8.2 Extra Hyperparameters: Deformation

Deformation Learning Rate starts from 0.00016, and exponentially decays to 0.0000016 with delay multiplier 0.01 after certain steps. Deformation happens on position x_i, scale s_i and rotation q_i, with σ_i, c_i unchanged across time.

DeformableGS [54] Following original paper, we set network depth as 8; network width as 256; time embedding dim as 6 for synthetic, 10 for real-world; position embedding dim as 10; time is additionally processed with an embedding MLP with 3 layers, 256 width before feed into deformation network for synthetic scene; exponential decay max step as 40000.

4DGS [50] Following original paper, we set network depth as 0 for synthetic, 1 for real-world; width as 64 for synthetic, 128 for real-world; plane resolution as $[1, 2]$ for synthetic, $[1, 2, 4]$ for real-world; exponential decay max step as 30000; plane TV loss weight as 0.0001 for synthetic, 0.0002 for real-world; time smoothness loss weight as 0.01 for synthetic, 0.0001 for real-world; L1 time planes loss weight as 0.0001; Grid Learning Rate similarly exponentially decay like Deformation Learning Rate, except for going from 0.0016 to 0.000016; grid dimension as 2, input coordinate dimension as 4, output coordinate dimension as 16 for synthetic, 32 for real, grid resolution as $[64, 64, 64, 100]$ for synthetic, $[64, 64, 64, 150]$ for real.

To stabilize training of deformation networks, we perform gradient norm clipping by 5.

8.3 Extra Hyperparameters: Curve

EffGS [19] uses Fourier Curve for x_i motion, and Polynomial Curve for q_i motion. Curve order is set to 2 for x_i, 1 for q_i. An optical-flow-based loss is additionally introduced to augment eq. (4) only for multi-view dataset, and here we do not include the flow loss as default.

STG [26] Original code is only supporting multi-view dataset as Gaussians are initialized by per-frame dense point cloud. To extend to monocular case, we copy point cloud for 10 times along time axis uniformly. During rendering, one option is to directly rasterize SH appearance as in 3DGS, but another option is to rasterize feature instead, which would be later fed into a decoder network D to generate color image. D's trained alongside with learning rate 0.0001.

8.4 Details of Dynamic 3D Gaussian Implementation

We update the codebase from Luiten et al. [34] to support our evaluation across datasets. We use the HyperNeRF scenes with our improved cameras poses to minimize the reconstruction error of static areas.

9 Related Work

Dynamic scene reconstruction has been a longstanding problem in computer vision [36, 42, 28, 34]. Earlier works utilize accurate depth cameras [36, 42] and reconstruct a dynamic scene using a monocular RGBD video. On the other hand, Joo et. al. [18] propose to use MAP visibility estimation with a large number of cameras and correspondences from optical flow to reconstruct a dynamic scene. In contrast, the focus of our study is on using a single-view RGB video to reconstruct dynamic scenes without taking depth or correspondences as input.

Recently, neural radiance fields [35] (NeRFs) revolutionized 3D reconstruction allowing for the reconstruction of a static 3D scene only given 2D input images. This success led to various extensions onto the dynamic setting, i.e. the 4D domain [40, 46, 43, 27, 12, 13, 3, 11, 2, 16] by treating time as an additional dimension in the neural field. Other relevant approaches include combining a 3D NeRF representation with a learned time-dependent deformation field [32, 10, 37, 38, 17, 22, 47, 45] and learning a set of motion basis [28] to optimize a dynamic scene.

The more recent explosion of works representing scenes with Gaussians [20, 21, 4] such as 3D Gaussian Splatting (3DGS) [20] has led to a number of promising extensions to its scene representation into the 4D domain which is the main focus of study in our work. As discussed in the earlier sections, one option to extend static 3DGS is to model motion as a 3D trajectory through time, learning motion by iteratively optimizing for per-Gaussian offsets into the next frame [34, 44, 9]. Methods also represent these 3D trajectories differently such as using explicit motion basis [6, 19, 30, 26, 57] or sparse control points [15]. Another line of works [50, 54, 29, 14, 33, 31, 7] learn a 3D Gaussians that live in canonical space and optimize a time-conditioned deformation network to warp Gaussians from canonical space to each timeframe. Lastly, a few works [8, 55] directly model Gaussians in 4D, i.e. extending across space and time.

10 Mathematical Motion Model Definitions

10.1 Polynomial

Curve methods ([19, 30, 26]) may use a polynomial basis of order L to define a f over time that determines Gaussian offsets, with coefficients $a_{i,j}$, and time offset t_i: Here, we denote g_i of G_i as the subset of parameters to change:

$$
\begin{aligned}
f(i,t) &= \sum_{j=0}^{L} a_{i,j}(t-t_i)^j \\
G_{i,t} &= \sum_{j=0}^{L} a_{i,j}(t-t_i)^j + G_i
\end{aligned}
\tag{5}
$$

10.2 Fourier

f could also use a Fourier basis with coefficients $a_{i,j}, b_{i,j} j = 0^L$, and time offset t_i:

$$G_{i,t} = \sum_{j=0}^{L} (a_{i,j} sin(j(t - t_i)) + b_{i,j} cos(j(t - t_i))) + G_i \tag{6}$$

10.3 RBF

f could be a Gaussian Radial Basis function with scaling factor c_i and time offset t_i

$$G_{i,t} = G_i \exp(-c_i(t - t_i)^2) \tag{7}$$

10.4 RTGS's 3D Rendering

RTGS [55] induces 3D world from 4D representation by obtaining the distribution of a 3D position conditioned on a specific time t:

$$\begin{aligned} x_{i|t} &= x_i[:3] + \Sigma_i[:3,3]\Sigma_i[3,3]^{-1}(t - x_i[3]), \\ \Sigma_{i|t} &= \Sigma_i[:3,:3] - \Sigma_i[:3,3]\Sigma_i[3,3]^{-1}\Sigma_i[3,:3] \end{aligned} \tag{8}$$

References

[1] Jeongmin Bae, Seoha Kim, Youngsik Yun, Hahyun Lee, Gun Bang, and Youngjung Uh. Per-gaussian embedding-based deformation for deformable 3d gaussian splatting, 2024.

[2] Minh-Quan Viet Bui, Jongmin Park, Jihyong Oh, and Munchurl Kim. Dyblurf: Dynamic deblurring neural radiance fields for blurry monocular video. *arXiv preprint arXiv:2312.13528*, 2023.

[3] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. *CVPR*, 2023.

[4] Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting, 2024.

[5] Yangming Cheng, Liulei Li, Yuanyou Xu, Xiaodi Li, Zongxin Yang, Wenguan Wang, and Yi Yang. Segment and track anything. *arXiv preprint arXiv:2305.06558*, 2023.

[6] Devikalyan Das, Christopher Wewer, Raza Yunus, Eddy Ilg, and Jan Eric Lenssen. Neural parametric gaussians for monocular non-rigid object reconstruction. *arXiv preprint arXiv:2312.01196*, 2023.

[7] Gang Zeng Diwen Wan, Ruijie Lu. Superpoint gaussian splatting for real-time high-fidelity monocular dynamic scene reconstruction. In *Forty-first International Conference on Machine Learning*, 2024.

[8] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4d gaussian splatting: Towards efficient novel view synthesis for dynamic scenes, 2024.

[9] Bardienus Pieter Duisterhof, Zhao Mandi, Yunchao Yao, Jia-Wei Liu, Mike Zheng Shou, Shuran Song, and Jeffrey Ichnowski. Md-splatting: Learning metric deformation from 4d gaussians in highly deformable scenes. *ArXiv*, abs/2312.00583, 2023.

[10] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.

[11] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023.

[12] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021.

[13] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Dynamic novel-view synthesis: A reality check. In *NeurIPS*, 2022.

[14] Zhiyang Guo, Wengang Zhou, Li Li, Min Wang, and Houqiang Li. Motion-aware 3d gaussian splatting for efficient dynamic scene reconstruction, 2024.

[15] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. *arXiv preprint arXiv:2312.14937*, 2023.

[16] Hankyu Jang and Daeyoung Kim. D-tensorf: Tensorial radiance fields for dynamic scenes. *ArXiv*, abs/2212.02375, 2022.

[17] Erik C.M. Johnson, Marc Habermann, Soshi Shimada, Vladislav Golyanik, and Christian Theobalt. Unbiased 4d: Monocular 4d reconstruction with a neural deformation model. *CVPR Workshop*, 2023.

[18] Hanbyul Joo, Hyun Soo Park, and Yaser Sheikh. Map visibility estimation for large-scale dynamic 3d reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[19] Kai Katsumata, Duc Minh Vo, and Hideki Nakayama. An efficient 3d gaussian representation for monocular/multi-view dynamic scenes, 2023.

[20] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[21] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *European Conference on Computer Vision (ECCV)*, 2022.

[22] Tobias Kirschstein, Shenhan Qian, Simon Giebenhain, Tim Walter, and Matthias Nießner. Nersemble: Multi-view radiance field reconstruction of human heads. *ACM Trans. Graph.*, 42(4), jul 2023.

[23] Agelos Kratimenos, Jiahui Lei, and Kostas Daniilidis. Dynmf: Neural motion factorization for real-time dynamic view synthesis with 3d gaussian splatting. *arXiV*, 2023.

[24] Jiahui Lei, Yijia Weng, Adam Harley, Leonidas Guibas, and Kostas Daniilidis. Mosca: Dynamic gaussian fusion from casual videos via 4d motion scaffolds. *arXiv preprint arXiv:2405.17421*, 2024.

[25] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition, pages 5521–5531, 2022.
[26] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. *arXiv preprint arXiv:2312.16812*, 2023.
[27] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
[28] Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. Dynibar: Neural dynamic image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
[29] Yiqing Liang, Numair Khan, Zhengqin Li, Thu Nguyen-Phuoc, Douglas Lanman, James Tompkin, and Lei Xiao. Gaufre: Gaussian deformation fields for real-time dynamic novel view synthesis, 2023.
[30] Youtian Lin, Zuozhuo Dai, Siyu Zhu, and Yao Yao. Gaussian-flow: 4d reconstruction with dynamic 3d gaussian particle. *arXiv:2312.03431*, 2023.
[31] Qingming Liu, Yuan Liu, Jiepeng Wang, Xianqiang Lv, Peng Wang, Wenping Wang, and Junhui Hou. Modgs: Dynamic gaussian splatting from causually-captured monocular videos, 2024.
[32] Yu-Lun Liu, Chen Gao, Andreas Meuleman, Hung-Yu Tseng, Ayush Saraf, Changil Kim, Yung-Yu Chuang, Johannes Kopf, and Jia-Bin Huang. Robust dynamic radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.
[33] Zhicheng Lu, Xiang Guo, Le Hui, Tianrui Chen, Ming Yang, Xiao Tang, Feng Zhu, and Yuchao Dai. 3d geometry-aware deformable gaussian splatting for dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.
[34] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis, 2023.
[35] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.
[36] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015.
[37] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.
[38] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.
[39] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.
[40] Sameera Ramasinghe, Violetta Shevchenko, Gil Avraham, and Anton van den Hengel. Blirf: Band limited radiance fields for dynamic scene modeling. In *AAAI 2024*, 2024.
[41] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
[42] Miroslava Slavcheva, Maximilian Baust, Daniel Cremers, and Slobodan Ilic. Killingfusion: Non-rigid 3d reconstruction without correspondences. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5474–5483, 2017.
[43] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. Nerfplayer: A streamable dynamic scene representation with decomposed neural radiance fields. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2732–2742, 2023.
[44] Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 3dgstream: On-the-fly training of 3d gaussians for efficient streaming of photo-realistic free-viewpoint videos. 2024.
[45] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video, 2020.
[46] Chaoyang Wang, Ben Eckart, Simon Lucey, and Orazio Gallo. Neural trajectory fields for dynamic novel view synthesis. March 2021.
[47] Chaoyang Wang, Lachlan Ewen MacDonald, László A. Jeni, and Simon Lucey. Flow supervision for deformable nerf. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21128–21137, June 2023.
[48] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[49] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[50] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Wang Xinggang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023.

[51] Yuting Xiao, Xuan Wang, Jiafei Li, Hongrui Cai, Yanbo Fan, Nan Xue, Minghui Yang, Yujun Shen, and Shenghua Gao. Bridging 3d gaussian and mesh for freeview video rendering, 2024.

[52] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022.

[53] Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8285–8295, 2023.

[54] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023.

[55] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024.

[56] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5335–5344, 2020.

[57] Heng Yu, Joel Julin, Zoltan A Milacski, Koichiro Niinuma, and Laszlo A Jeni. Cogs: Controllable gaussian splatting. *arXiv*, 2023.

[58] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[59] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001.