EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Adaptive Spatio-Temporal 3D Gaussian Splatting for Scenes with Oscillatory Motion

Petros Tzathas, Jeffrey Hu, Andréas Meuleman, Guillaume Cordonnier, and George Drettakis

Inria, Université Côte d'Azur, France

Ours | SCGS [HSY*23] | Ours | 4D-GS [WYF*24]

Figure 1: *Our method introduces a new explicit adaptive spatio-temporal representation and densification for 3D Gaussian splatting to capture dynamic scenes with incoherent motion, such as vegetation moving in the wind. We achieve a significant improvement in rendering speed and higher quality compared to state-of-the-art methods, both on small-scale scenes (left) and large outdoor forest scenarios (right).*

Abstract

Our goal is to reconstruct scenes with stochastic, incoherent motion such as leaves moving in the wind, that can be particularly challenging because of small objects with similar appearance that move independently. Previous dynamic 3D Gaussian Splatting solutions either represent motion implicitly with neural networks achieving good quality but lower framerate, or explicitly with a function, often with higher training times and lower quality. To overcome these limitations, we propose an explicit method that introduces adaptive space-time densification and smoother optimization. We introduce a new densification approach based on error moments that are used to guide primitive splitting, and we adaptively refine the number of keyframes used based on the variance of error. We observe that dynamic reconstruction from monocular video is hard for standard optimization pipelines. To counter this, we introduce a weighted Adam approach that improves results based on primitive visibility. Finally, to handle the hard case of independent motion of similar-looking objects, we introduce an image-driven as-rigid-as-possible regularization. Our method has higher quality than previous explicit solutions, and has significantly higher framerate for rendering.

1. Introduction

3D Gaussian Splatting (3DGS) [KKLD23] allows fast reconstruction and high-quality rendering of captured scenes. Reconstructing *dynamic* scenes with moving objects using 3DGS is much harder than the original use-case of *static* scenes. For the case of such scenes captured with multiple cameras, many successful solutions exist (e.g., [LKLR24, YYPZ24, YGZ*24, DWD*24]). Reconstructing scenes with motion from monocular video however is a fundamentally ill-posed problem. Broadly speaking, previous methods represent motion either implicitly using a neural net-

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

work (e.g., [WYF*24, YGZ*24, SHK*24]), achieving good quality but lower framerate, or explicitly with a function (e.g., a polynomial) to represent motion of Gaussian primitives, sometimes resulting in higher training times and often having lower quality [YYPZ24, LWH*24, PBB*25]. We propose an explicit method that – thanks to spatio-temporal refinement and carefully designed optimization – provides up to 2.5x times faster rendering speeds than previous methods, and has better visual quality for the AmbientGaussians dataset [SHK*24] we use for evaluation. Our solution also trains faster compared to previous methods that are mainly explicit.

We target scenes with stochastic, incoherent motion such as leaves moving in the wind (Fig.1). Rendering captured scenes with such effects adds a significant level of realism and "liveliness" to radiance fields. However, such scenes can be particularly challenging due to the presence of small objects with similar appearance that move independently.

Neural networks (NNs) used by implicit methods to represent motion [LKL*25, WYF*24, YGZ*24, SHK*24] achieve satisfactory quality and optimize fast, but are limited in the framerate at rendering time because of the NN evaluation cost, and have difficulty for larger scenes. Previous explicit methods are slow to optimize since the optimization landscape is not as smooth as for NNs, and often require tracks [LWH*24, PBB*25, WYG*25] as input to initialize the motion estimate. Obtaining such tracks for our target scenes is often very hard, because of the small, same-colored moving objects such as leaves. Another difficulty of explicit methods is that in addition to *spatial* densification, the time dimension needs to be represented at a resolution that fits motion in the scene. Such *temporal densification* can be critical for effective optimization.

To overcome these limitations, we introduce an *explicit* method based on per-primitive splines that introduces adaptive space-time densification and smoother optimization. Standard gradient-based densification [KKLD23] is hard to adapt to our space-time case. Instead, we densify based on error. We introduce a new formulation that estimates the first and second moments of error for each Gaussian, and use them to guide adaptive splitting of Gaussian primitives. We introduce a method to adaptively add (densify) keyframes based on variance of error over time, assigning finer temporal granularity where needed. Dynamic reconstruction from monocular video poses significant challenges for the standard optimization pipeline: due to changes in time, many observations are invalid at each iteration. To counter this, we introduce an improved weighted Adam approach for optimization that handles this issue based on visibility of each primitive. Finally, to handle the hard case of independent motion of similar-looking objects, we introduce an image-driven as-rigid-as-possible regularization.

In summary, we introduce three main contributions:

- A new error moment formulation for 3D Gaussian Splatting, that combines 2D metrics to 3D, thus exploiting rich pixel-level error information.
- A fully adaptive space-time densification method, based on error moments.
- An improved optimization method, well adapted to the hard case of monocular videos with independent stochastic motion of objects that have similar appearance – such as leaves in the wind.

We evaluate our method on the previously published AmbientGaussians dataset [SHK*24], demonstrating that we achieve a state-of-the-art visual quality and much faster rendering speed.

2. Previous Work

Novel view synthesis aims to generate images of a scene from previously unseen viewpoints, enabling free-viewpoint navigation. In dynamic scenes, this task is a broad and actively researched area. Given the diversity of approaches, we focus on radiance fields and methods that leverage 3D Gaussian Splatting. We first briefly discuss static implicit and explicit representations. We then highlight how these representations have been extended to dynamic view synthesis, considering both the multi-view setting and the more challenging monocular scenario.

Static view synthesis. Implicit volumetric neural representations (NERF) [BMV*22, BMV*23, LHY*24] are easy to optimize and can produce high-quality results. However, their reliance on ray marching makes both training and rendering computationally expensive. Their implicit nature also makes them difficult to edit, requiring complex manipulation schemes [YSL*22, JKK*23, PPGT*23]. In contrast, primitive-based representations such as 3D Gaussian Splatting [KKLD23] are efficiently rasterized on the GPU, which enables real-time rendering and simplifies scene editing [CCZ*24, XZQ*23]. However, these methods require more complex optimization strategies, specifically in terms of particle initialization and progressive densification. Further work explored improved densification schemes [BPK24, KRS*24, MGK*24, DDL*25a, ZHL*24, FCC*24, DDL*25b] or replaced densification with a better initialization [LHND25, MSL*25]. As observed previously [BPK24, ZHL*24] the original densification strategy fails to adequately cover some regions, particularly in outdoor scenes with high frequency content like vegetation, which we target. Alternative densification strategies usually rely on optimization to move new primitives where needed, rather than placing them carefully, thus delaying convergence, especially for dynamic scenes. Inspired by [BPK24] that uses reconstruction error to trigger densification, we improve the positioning of newly added Gaussians.

Dynamic representations. The key difference with static scenes is that dynamic representations [LSS*19, LSS*21, AHR*23] have to account for the temporal evolution of the Gaussians to enable the reconstruction and rendering of dynamic scenes from novel viewpoints in both space and time. Many of these approaches feature an explicit static scene representation combined with an implicit model for motion to ensure local consistency and stability during optimization [LKL*25, WYF*24, YGZ*24, SHK*24, YXL*25, CCK*25]. These methods benefit from the local smoothness of the implicit deformation field, which naturally regularizes the spatial variations of the scene deformation. This regularization yields robust methods, but sacrifices speed and editability. There also exist methods based on explicit temporal representations. Examples include 4D primitives [YYPZ24, DWD*24, WYX*25, XXY*24, LCLX24] that have limited temporal support, allowing them to fade in and out over time. While achieving good visual quality, these 4D primitives do not directly track motion, preventing direct motion editing and simulation. They are also redundant, as

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

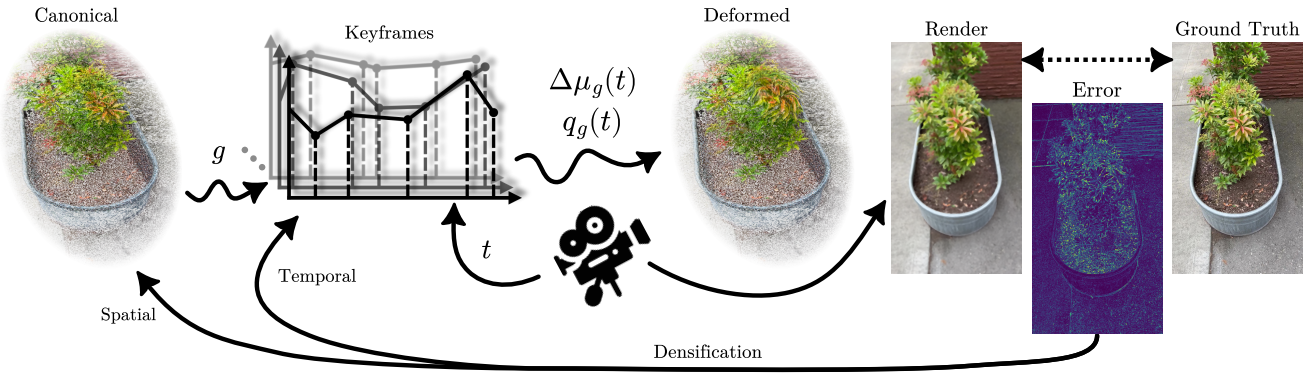

Figure 2: *We augment the 3D Gaussian Splatting representation with per-primitive splines that represent motion. Each Gaussian g has an associated keyframe list which gets queried with time t to give a displacement $\Delta\mu_g(t)$ and rotation $q_g(t)$ (Section 4). We use the pixel error of the renderings to guide the spatio-temporal densification which adapts both Gaussians and keyframes (Sections 4.1 and 4.2).*

Gaussians must be duplicated when the motion is complex, making these representations memory inefficient. Other approaches model the motion of Gaussians [SHU*24,LWH*24,LWJ*24,PBB*25] using dense translation sequences, motion scaffold graphs, or splines. Most methods that explicitly model motion use a constant number of parameters regardless of the trajectory, which is inefficient for parts of the scene with simple motion and insufficiently expressive for others. While SplineGS [PBB*25] adapts the number of parameters via a pruning strategy, it requires initializing all Gaussians with the maximum number of parameters and cannot increase the number of parameters during optimization if needed. Similar to [LWJ*24, PBB*25], we use per-primitive splines to represent time-varying translation and rotation. However, instead of cubic Hermite splines we opt for simpler linear ones for the translation which, combined with our space-time densification strategy, allows for efficient adaptation of the number of keyframes per primitive.

Dynamic view synthesis from monocular inputs. Reconstructing dynamic scenes from a single image per observation in time is significantly more challenging due to the inherent camera-scene motion ambiguity: it is unclear whether an observed displacement is caused by scene motion or camera motion. Among the dynamic novel-view synthesis methods, a subset (e.g., [TTG*21, GSKH21, LGM*23, LWC*23, SHK*24, KVN24, YGZ*24, LWH*24, SHU*24, PBB*25]) have demonstrated the ability to handle this setting. These methods either rely on implicit deformation fields that provide local smoothness at the cost of rendering performance or require priors such as motion masks, depth, and/or dense optical flow. Motion masks are particularly challenging to obtain automatically for general videos. For example, most explicit methods for monocular videos [SHU*24,PBB*25] require the user to manually click on dynamic elements for segmentation, which is impractical for scenes with a large number of distinct moving objects, such as leaves.

In this work, we enable the reconstruction of dynamic scenes from monocular video using an explicit representation that maintains memory efficiency through adaptive spatio-temporal densification, while allowing for fast optimization, and fast rendering. We achieve this through our efficient modified Adam optimization and minimal regularization with our image-based ARAP self-supervision, which does not require user interaction or prior-based supervision.

3. 3DGS Background

Given a set $\mathcal{I}$ of calibrated images of a scene, 3D Gaussian Splatting reconstructs a 3D scene as an ensemble of 3D Gaussian primitives, which have the following parameters: center μ, covariance Σ, opacity value o, and per-color-channel spherical harmonics coefficients. In the following, we will use indices I, p, g on quantities that vary over images/views, pixels, and/or Gaussians, respectively.

3.1. Image formation model

Given a calibrated camera, the first step of the image formation is to transform each 3D Gaussian to viewspace and sort the primitives with respect to the z-depth of their centers. Then, the Gaussians are mapped to screen space by perspectively projecting their center and using the affine approximation of the perspective projection for their covariances Σ_g [KKLD23]. Specifically, if W_I is the camera's rotation matrix, $J_{g,I}$ is the Jacobian of the perspective projection at the Gaussian's center:

$$\Sigma'_{g,I} = A_{g,I}\Sigma_g A_{g,I}^T \tag{1}$$

$$A_{g,I} = J_{g,I}W_I \tag{2}$$

where $A_{g,I}$ is the combined effect of the aforementioned transformations/matrices. The color of a Gaussian for a particular view I is computed by evaluating the spherical harmonics for the direction of the ray passing through the center of that Gaussian. A pixel's color c_p is then the alpha-blended Gaussians' colors:

$$c_p = \sum_g c_{g,I} a_{g,p} T_{g,p} \tag{3}$$

$$T_{g,p} = \prod_{g' <_I g} (1 - a_{g',p}) \tag{4}$$

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

where $a_{g,p}$ is the Gaussian's opacity multiplied by the 2D falloff described by Σ'_g, and $g' <_I g$ denotes that the product is taken over the Gaussians g' that are before the Gaussian g in their ordering for view I.

3.2. Densification

The representation is optimized using stochastic gradient descent, using a loss combining L_1 and SSIM, computed on one image per step.

$$\mathcal{L}_{\text{photo}}(I, I_{gt}) = (1-\lambda_{\text{ssim}})L_1(I, I_{gt}) + \lambda_{\text{ssim}}\,(1-\text{SSIM}(I, I_{gt})) \tag{5}$$

The original method does not rely on a fixed number of primitives, but instead adds and removes points every few iterations. Gaussians are removed simply by culling primitives with opacity below a threshold. New Gaussians are added by *densifying* existing ones either by cloning or splitting. Densification is decided based on the magnitude of the gradient of position with respect to its projected center. If it exceeds a threshold, the Gaussian is densified. The choice between cloning and splitting is based on its size. Specifically:

- Small Gaussians are **cloned**: A new identical Gaussian is added.
- Large Gaussians are **split**: The original is removed and 2 new ones are added in its place. Their positions are sampled from a Normal distribution with mean and covariance given by the original. Their covariances are a scaled-down version of the original's.

We next present our improved space-time densification scheme based on an analysis of the reconstruction error, both for a densification criterion (Section 4.1) and the placement of new primitives (Section 4.2). Finally, we will show how we adapt the Gaussian optimization framework to the dynamic setting (Section 4.3).

4. Method

Our input is a monocular video. We assume that we have a sequence $\mathcal{I}$ of N calibrated images, each image I with its unique timestamp $t_I \in [0,1]$. Calibration is typically performed using Structure-from-Motion (SfM) [SF16]. Our scenes contain spatially variable non-rigid motion (e.g, foliage oscillating in the wind); our goal is to reconstruct this motion.

To represent motion, at each time t we displace primitives from the configuration given by their centers μ and covariances Σ. For this reason, we refer to this configuration as the *canonical* space. The canonical space does not necessarily correspond to some particular input frame, and it is learned during training.

Storing the displacement of each Gaussian at each frame would be prohibitively expensive in memory. Instead, we represent the displacements over time with piecewise splines. We augment each Gaussian primitive with a sequence of n_g *keyframes* t_g^i, $0 \le i < n_g$, serving as the nodes of the spline. The frames of the input sequence are uniform in time, and the keyframes correspond to some of these frames. For each keyframe of a Gaussian, we store one translation $\Delta\mu_g^i$, and one rotation represented by a quaternion q_g^i.

Given a time t, the translation $\Delta\mu_g(t)$ of the Gaussian is given by linear interpolation. More formally:

$$\Delta\mu_g(t) = \begin{cases} \Delta\mu_g^0 & ,t < t_g^0 \\ \text{lerp}\left(\Delta\mu_g^i, \Delta\mu_g^{i+1}, \frac{t-t_g^i}{t_g^{i+1}-t_g^i}\right) & ,\exists i : t_g^i \le t < t_g^{i+1} \\ \Delta\mu_g^{n_g} & ,t > t_g^{n_g} \end{cases} \tag{6}$$

The same applies to the quaternions, but using spherical linear interpolation instead.

The position and covariance of a Gaussian at time t are then:

$$\mu_g(t) = \mu_g + \Delta\mu_g(t) \tag{7}$$

$$\Sigma_g(t) = R_g(t)\Sigma_g R_g^T(t) \tag{8}$$

where $R_g(t)$ is the rotation matrix corresponding to the quaternion $q_g(t)$. Eq. 2 is adapted as follows:

$$\Sigma'_{g,I} = A_{g,I}\Sigma_g A_{g,I}^T \tag{9}$$

$$A_{g,I} = J_{g,I}W_I R_g(t_I) \tag{10}$$

Note that we use gradient descent to optimize the canonical positions and quaternions, and the keyframed translation and rotations. However, we do not optimize the keyframe times t_g^i, but instead analyze the motion to perform adaptive temporal densification (Section 4.2.2).

4.1. Error Moments for Spatio-temporal Densification

Gaussians are initialized from the SfM points with 1 keyframe each, $t_g^0 = 0$. We will thus be both adding more Gaussians, as in the static case, but also adding more keyframes to each Gaussian. We refer to these processes as spatial and temporal densification respectively.

Adapting gradient-based densification strategies to this spatio-temporal context is not obvious. Instead, we introduce a splitting strategy that is guided by the error incurred by the Gaussian representation of the scene, and in particular *error moments*. Based on these error moments, our approach will place new Gaussians where needed, as opposed to randomly positioning them as in previous solutions. We draw inspiration from VET [FRF*23], where the authors propose using a tomography model (NeAT) [RWL*22] to reconstruct a 3D error distribution based on the loss images, in order to place new points where they are needed. Employing this neural model directly in the context of Gaussian splatting would greatly increase training time.

To avoid this overhead, we exploit the explicit nature of the scene representation and reason about the 3D error using per-Gaussian quantities. We start by computing the moments of the error in each 2D input view, since these describe the shape of the error distribution. The error incurred by each pixel provides fine-grain information to each Gaussian primitive. We then need to handle our multi-view context, where the 2D error from several input views contributes to the overall error of a given Gaussian. As we shall see below, error moments will guide the *placement* of new Gaussians during splitting.

Similarly to previous work [RWL*22, BPK24, LWJ*24], for a

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

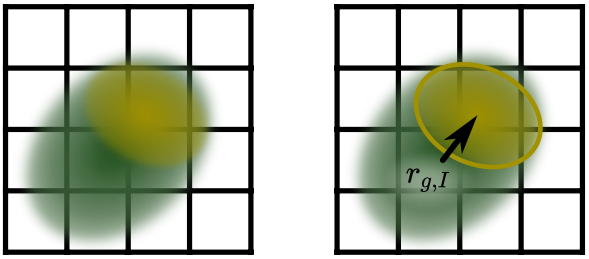

Figure 3: *The green region illustrates a Gaussian projected on a specific view and the orange region is the distribution of the error. The first moment $\boldsymbol{r}_{g,I}$ is a vector indicating the mean position of the error, and the second moment $\boldsymbol{S}_{g,I}$ is visualized as a yellow ellipse describing the shape of the distribution around it.*

pixel p, we distribute its error e_p over the Gaussians by multiplying with the weight used for the color of the Gaussian. However, instead of the total photometric loss, we use only the L_1 loss as the error function, since it is more detailed and consistent across views:

$$w_{g,p} = a_{g,p} T_{g,p} \tag{11}$$

$$e_{g,p} = e_p w_{g,p} \tag{12}$$

The error of a Gaussian for a particular image I is then:

$$w_{g,I} = \sum_{p \in I} w_{g,p} \tag{13}$$

$$e_{g,I} = \sum_{p \in I} e_{g,p} \tag{14}$$

This can be seen as the 0-order moment of the image error for the Gaussian. We combine these Gaussian view errors over the set of relevant input views $\mathcal{I}$ by weighting with $w_{g,I}$ to get the final Gaussian error:

$$e_g = \frac{\sum_{I \in \mathcal{I}} e_{g,I} w_{g,I}}{\sum_{I \in \mathcal{I}} w_{g,I}} \tag{15}$$

In [BPK24], the authors take the maximum over the views instead. We prefer these weighted averages because they incorporate occlusion information through the transmittance $T_{g,p}$, while being less sensitive to outlier view errors and therefore avoiding redundant densification.

For each view we also compute the (2D) first and second moments of the projected error distribution, which codify its center and spread. Note that the first moment $\boldsymbol{r}$ is a vector, while the second moment $\boldsymbol{S}$ is a symmetric matrix.

$$\boldsymbol{r}_{g,I} = \frac{\sum_{p \in I} \boldsymbol{r}_{g,p} e_{g,p}}{e_{g,I}} \tag{16}$$

$$\boldsymbol{S}_{g,I} = \frac{\sum_{p \in I} \boldsymbol{r}_{g,p} \boldsymbol{r}_{g,p}^T e_{g,p}}{e_{g,I}} \tag{17}$$

where $\boldsymbol{r}_{g,p}$ is the vector from the projected center of the Gaussian g to the center of pixel p. These quantities are illustrated in Fig. 3.

Combining the per-view quantities in this case is more complex than the case of just the error (Eq. 15), because we need to "unproject" the moments – no longer scalars – from their respective 2D space to the common 3D one. We do this by minimizing the distance between the view moments and the projections of the corresponding 3D moments. Since these projections are enacted by the matrices $A_{g,I}$, this is accomplished by solving the following least squares problems:

$$\min_{\boldsymbol{r}_g} \sum_{I \in \mathcal{I}} w_{g,I} \left\| \boldsymbol{r}_{g,I} - A_{g,I} \boldsymbol{r}_g \right\|^2 \tag{18}$$

$$\min_{\boldsymbol{S}_g} \sum_{I \in \mathcal{I}} w_{g,I} \left\| \boldsymbol{S}_{g,I} - A_{g,I} \boldsymbol{S}_g A_{g,I}^T \right\|_F^2 \tag{19}$$

where $\| \cdot \|_F$ is the Frobenius norm, and $A_{g,I}$ are the same matrices approximating perspective projection as used in Eq. 1.

The solution of Eq. 18 will provide, for each Gaussian, a vector $\boldsymbol{r}_g$ originating from its center and which, when projected, lies close to the centers of the view error distributions, therefore acting as the first moment of a 3D distribution of error. We illustrate this in Fig. 4, where we show how the projections of $\boldsymbol{r_g}$ matches the view first moments. We use $\boldsymbol{r_g}$ for splitting.

The minimizer of 18 is:

$$\boldsymbol{r}_g = \left(\sum_{I \in \mathcal{I}} w_{g,I} A_{g,I}^T A_{g,I} \right)^{-1} \sum_{I \in \mathcal{I}} w_{g,I} A_{g,I}^T \boldsymbol{r}_{g,I} \tag{20}$$

The form of the equation corresponds to a weighted average and so suggests that the view moments are un-projected from 2D to 3D and combine to give r_g.

The per-view second moments can also be combined, providing a 3D ellipsoid that describes the spread of the error distribution. We can arrive at a similar expression for the second moment after some algebraic manipulation.

Specifically, we utilize properties of the vec operation [Min00], which transforms a matrix to a vector by concatenating its rows. The detailed derivation is provided in Appendix A. Similarly to the first moments the solution is:

$$\text{vech}(\boldsymbol{S}_g) = \left(\sum_{I \in \mathcal{I}} w_{g,I} D_3^T B_{g,I}^T B_{g,I} D_3 \right)^{-1} \sum_{I \in \mathcal{I}} w_{g,I} D_3^T B_{g,I}^T \text{vec}(\boldsymbol{S}_{g,I}) \tag{21}$$

$$\boldsymbol{S}_g^c = \boldsymbol{S}_g - \boldsymbol{r}_g \boldsymbol{r}_g^T \tag{22}$$

The forms of Eq. 20 and 21 suggest that they can be computed efficiently in an online manner, since we can compute the sums involved while iterating over the views and invert only at the end.

In order to make the process more robust, after computing the moments we "clamp" them based on the shape of the Gaussian. We detail how this is done in Appendix B.

4.2. Adaptive Spatio-Temporal Splitting

We next show how to use the error moments calculated above for our adaptive spatio-temporal splitting. We start with the spatial splitting case, then describe adaptive temporal splitting.

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

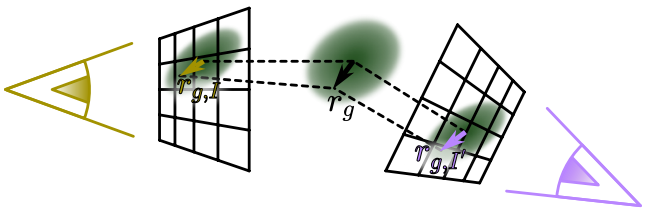

Figure 4: *We combine the 2D first moments from the yellow and purple views by un-projecting them into world space, using Eq. 20. The combined black 3D vector is used for splitting.*

4.2.1. Spatial Splitting

We choose Gaussians to be densified by thresholding their error. For each Gaussian with error above τ_{space} we use a plane to separate it in two. We rely on the algorithm proposed by [FCC*24] that, given a plane, tries to place Gaussians on either side of it such that the difference with the original Gaussian is minimal. In order to avoid Gaussians getting too small too quickly, we choose a plane passing through the center of the Gaussian, but unlike previous methods [FCC*24, DDL*25b], that use the plane perpendicular to the Gaussian's longest axis, we use the error moments to choose the plane's orientation.

Because the second moment is a symmetric positive definite matrix it can be visualized as an ellipse centered around the point given by the first moment. We choose the plane to be parallel to the tangent of this ellipse in the direction of the first moment as shown in Fig. 5. The normal of this plane is given as follows:

$$\frac{\left(\boldsymbol{S}_g^c\right)^{-1}\boldsymbol{r}_g}{\left\|\left(\boldsymbol{S}_g^c\right)^{-1}\boldsymbol{r}_g\right\|} \tag{23}$$

4.2.2. Temporal densification

For each Gaussian g and each time t there exists a keyframe t_g^i such that $t_g^i \leq t < t_g^{i+1}$. If the Gaussian only has 1 keyframe, then $t_g^0 \leq t$. A Gaussian's keyframes thus divide the frame sequence into *segments*.

If a Gaussian follows the motion of the area it represents correctly, then its error should be consistent across images. On the contrary, if the Gaussian doesn't not follow the motion correctly we expect the error to show variation across frames.

A natural way to estimate this variation is to compute the ratio of the standard deviation of the Gaussian's image error compared to its mean over time. However, here we divide the image error with the number of pixels $N_{g,I}^p$ the Gaussian is projected onto, thus ignoring the variation of the error due to different projected areas. We compute this ratio ρ_g for each Gaussian and each segment between adjacent keyframes:

$$\bar{e}_{g,I} = \frac{e_{g,I}}{N_{g,I}^p} \tag{24}$$

$$\bar{e}_g = \frac{\sum_{I\in\mathcal{I}} \bar{e}_{g,I} w_{g,I}}{\sum_{I\in\mathcal{I}} w_{g,I}} \tag{25}$$

$$\sigma_{\bar{e}_g} = \sqrt{\frac{\sum_{I\in\mathcal{I}} \bar{e}_{g,I}^2 w_{g,I}}{\sum_{I\in\mathcal{I}} w_{g,I}} - \bar{e}_g^2} \tag{26}$$

$$\rho_g = \frac{\sigma_{\bar{e}_g}}{\bar{e}_g} \tag{27}$$

where the $w_{g,I}$ is the weight defined in Eq. 13. Note that the equations above should also be indexed by the segment, but we omit this for clarity of notation.

To capture error due to motion of Gaussians that are close to moving edges, we split all the segments for a Gaussian if the ratio of at least one of its segments exceeds threshold τ_{time}. To account for Gaussians in uniformly colored areas — where variations of error are smaller — we also split Gaussians that have a KNN neighbor that satisfies the criterion above, even if the Gaussian itself does not; we use $K = 10$.

A new keyframe is added at the argmax of $\bar{e}_{g,I}$, unless this argmax coincides with the left keyframe, in which case the new keyframe is the midpoint of the segment. We do not add new keyframes when the resulting segments would be smaller than length l_{seg}.

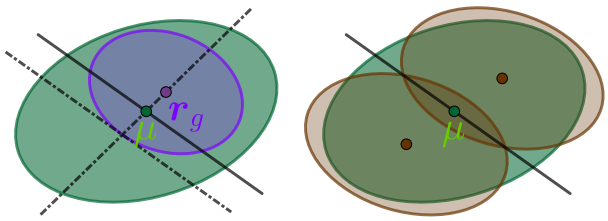

Figure 5: *2D illustration of the use of moments for spatial splitting. On the left, the green and purple ellipses represent the Gaussian and the error moments respectively. The splitting plane (line in 2D) is parallel to the tangent at the purple ellipse in the direction of the first moment. On the right, the brown ellipses represent the Gaussians resulting from splitting.*

4.2.3. Spatial vs Temporal Densification

It is hard to decide whether a Gaussian needs to be densified spatially or temporally at every iteration: doing so may lead to Gaussians failing to reconstruct either motion or appearance in detail. To avoid this issue we interleave spatial and temporal densification, alternating between them every 2 epochs.

4.3. Improved Adaptive Spatio-Temporal Optimization

The error moments and the adaptive spatio-temporal splitting provide a new basis for dynamic 3DGS with monocular video input. However, the optimization landscape differs in several ways from previous solutions, requiring improvements to the standard optimization process. In particular we propose a weighted Adam approach with an adaptive learning rate, and a geometric regularization to preserve the shapes of moving objects.

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

4.3.1. Weighted Adam

Most Gaussian Splatting methods use an unmodified version of Adam [KB14]. TamingGS [MGK*24] uses masked updates but their reasoning is efficiency, and they do not adjust the learning rate. This implies that all the parameters that are being optimized and their moments used by Adam are being updated at every training iteration, regardless of how "meaningfully" they contributed to the loss. For example, the gradients of the attributes of a Gaussian that lies outside the frustum of the camera that is used at a particular training iteration will trivially be zero, however this information is not useful for optimization. When Adam is given a zero gradient it cannot determine that this is because the parameter has had no effect in the image formation and instead treats it as if the parameter is at a critical point of the loss function. This argument also holds for non-zero gradients that similarly do not carry meaningful information, for example, the position of a Gaussian that is in the frustum but is occluded almost completely by other Gaussians.

In many cases, this situation can indirectly be resolved by the adaptive nature of Adam, as long as the parameters that are being optimized affect a considerable portion of the training samples. This is true for the original algorithm, and representations of motion which rely on neural components or per Gaussian coefficients for basis functions of non-local support (like polynomials or sinusoids). In our case, and in particular when the segments of a Gaussian become short, the involved keyframed attributes do not make any contribution for the majority of the training sequence. Standard Adam will, however, continually update these parameters and their momenta with information from previous, but not currently relevant, samples leading to divergence, as seen visually in Fig. 8 and reflected in metric in Table 3.

To overcome this problem, we propose an Adam variant which uses weighted updates to the parameters, using a weight based on visibility of Gaussian primitives. In particular, we will use the average transmittance for the Gaussian over the image used for the training iteration as a weight:

$$\eta = \overline{T}_{g,I} = \frac{\sum_{p \in I} T_{g,p}}{N^p_{g,I}} \tag{28}$$

For keyframed attributes we only update them if the were used in the iteration. The exact Adam updates are given in Appendix C.

In addition, because our splitting strategy makes Gaussians smaller more quickly than the original densification, instead of using a scheduled learning rate for the centers and the translations of the Gaussians as in the original 3DGS, we use the largest scale of the Gaussian (largest singular value of covariance), to modulate the learning rate:

$$\gamma_g = \min\left\{1, \alpha \max_i \{s_{g,i}\}\right\} \gamma \tag{29}$$

where α is a multiplier we set to 0.01.

Nevertheless, the sparser but more consistent updates of our weighted Adam allow us to increase the learning rates for the parameters. We report these learning rates together with other parameter values in Appendix D.

4.3.2. Geometric regularization

Due to the sparseness of observations, monocular video comes with inherent ambiguities, which are accentuated for scenes with foliage, since all leaves have the same color. Importantly, the motion of a point along the ray direction is not observable. Most commonly this difficulty is overcome by the use of some sort of regularization that enforces the deformation of the undergone by the points varies coherently over space. The most common one is some version of As-Rigid-As-Possible (ARAP) loss [IMH05, SA07], which favors deformations that are locally rigid. If we have a collection of points with different positions and rotations at two times t, t' then the ARAP loss is:

$$\mathcal{L}_{\text{arap}} = \sum_i \sum_{j \in \mathcal{N}(i)} \left\| \left(p_j^t - p_i^t\right) - R_i^t \left(R_i^{t'}\right)^T \left(p_j^{t'} - p_i^{t'}\right) \right\|^2 \tag{30}$$

where p_i^t and R_i^t are the position and rotation, respectively, of point i at time t, and the inner sum is taken over a neighborhood $\mathcal{N}$ of i. For meshes, the neighborhood is readily given by the edges. For methods using Gaussians, KNN is usually used to define it [LKLR24, LWH*24].

However, defining the neighborhood with KNN can lead to grouping Gaussians that belong to parts of the scene that even though are close spatially, move independently from each other (e.g., two leaves of a tree). This can lead to a dampening of the motion. Instead, we group implicitly in image space, inspired by the bilateral filter design [TM98], which weights the influence across pixels by using their color difference along with the usual distance on the image plane.

Given a training image I with time t_I, we sample another time t', not necessarily corresponding to an input frame. For each Gaussian we compute the 6D screw-axis representation of the transformation between times t_I and t' [PSB*21], we then blend these vectors to form an image the same way we blend color [Ale02]. For each pixel p we compute a weighted sum of the squared distances of the splatted screw-axis representations between p and neighboring pixels p':

$$\mathcal{L}_{\text{geo}} = \sum_{p' \in \mathcal{N}(p)} k_{c,c'} k_{d,d'} k_{p,p'} \left\| s_p - s_{p'} \right\|^2 \tag{31}$$

where like for the bilateral filter $k_{c,c'}$ is a similarity weight between the colors c, c' of pixels p, p', $k_{d,d'}$ is a similar weight for the rendered depth, and $k_{p,p'}$ is a weight decreasing with distance on the image plane. The final loss is the average of the per pixel losses. Note that we do not divide by the sum of the weights because in contrast to a normal bilateral filter we do not care about conserving "energy", we merely want to compare pixels.

4.4. Loss

Besides Eq. 5 and the geometric loss described above, we include a velocity distortion loss inspired by the depth distortion loss used in other methods [BMV*22, HYC*24]. This loss aims at making Gaussians that are splatted to the same pixels have consistent velocities. The expression for a single pixel p is:

$$\sum_g \sum_{g'} w_{g,p} w_{g',p} \| v_g - v_{g'} \|^2 \tag{32}$$

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

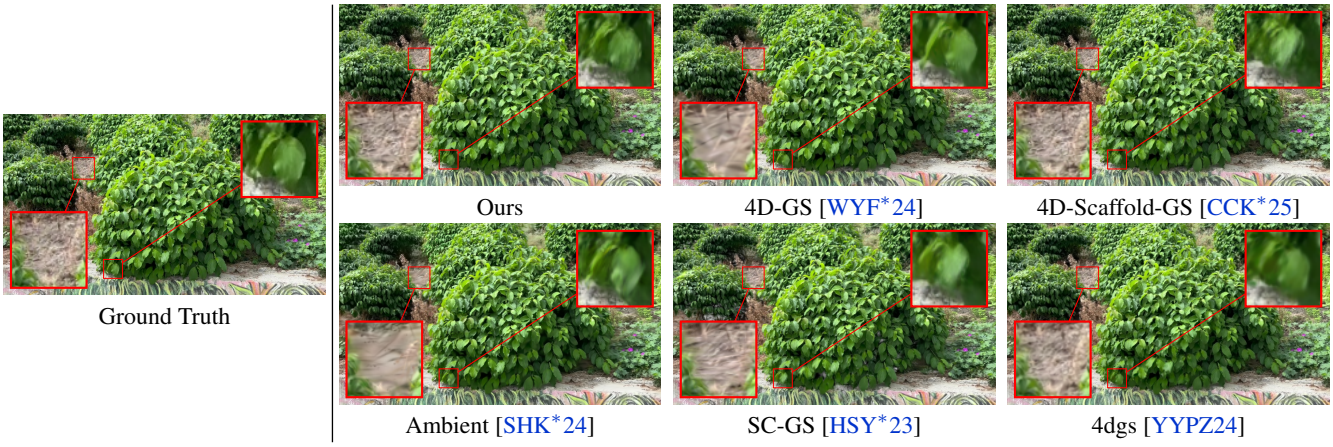

Figure 6: *We provide renderings for one of the urban scenes from every method, trained with default settings. Our method is able to reconstruct high frequency details on images, compared to the other methods. Please also see the supplemental video.*

Scene	Ours	4D-GS [WYF*24]	4D-Scaffold-GS [CCK*25]	AmbientGaussians [SHK*24]	SC-GS [HSY*23]	4dgs [YYPZ24]
	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑
Urban Scene #1	28.82 / 0.92 / 0.08 / 86.07	28.98 / 0.91 / 0.11 / 88.52	28.73 / 0.92 / 0.07 / 83.15	26.54 / 0.85 / 0.15 / 79.29	29.81 / 0.94 / 0.06 / 90.13	26.76 / 0.90 / 0.11 / 69.21
Urban Scene #2	27.47 / 0.89 / 0.10 / 74.05	28.49 / 0.90 / 0.12 / 78.77	26.84 / 0.89 / 0.10 / 68.81	25.38 / 0.83 / 0.16 / 62.53	24.34 / 0.81 / 0.18 / 54.52	26.31 / 0.87 / 0.12 / 62.53
Urban Scene #3	27.77 / 0.89 / 0.10 / 77.63	28.56 / 0.88 / 0.14 / 81.06	27.74 / 0.90 / 0.08 / 70.36	26.93 / 0.82 / 0.20 / 72.11	23.68 / 0.72 / 0.24 / 50.86	25.98 / 0.85 / 0.13 / 60.45
Urban Scene #4	26.33 / 0.86 / 0.13 / 61.98	26.48 / 0.85 / 0.17 / 59.90	27.52 / 0.88 / 0.10 / 67.29	24.39 / 0.78 / 0.23 / 44.73	26.72 / 0.88 / 0.12 / 62.24	25.68 / 0.84 / 0.17 / 47.24
Average (small scenes)	27.60 / 0.89 / 0.10 / 74.93	**28.12** / 0.88 / 0.13 / **77.06**	27.71 / **0.90** / **0.09** / 72.40	25.81 / 0.82 / 0.19 / 64.67	26.14 / 0.84 / 0.15 / 64.44	26.18 / 0.87 / 0.13 / 59.86
Forest Scene #1	25.85 / 0.85 / 0.18 / 76.85	24.32 / 0.74 / 0.28 / 65.65	24.73 / 0.83 / 0.18 / 67.96	– / – / – / –	– / – / – / –	– / – / – / –
Forest Scene #2	25.01 / 0.81 / 0.18 / 64.15	25.49 / 0.80 / 0.21 / 68.49	25.67 / 0.84 / 0.15 / 69.22	– / – / – / –	– / – / – / –	– / – / – / –
Forest Scene #3	23.65 / 0.78 / 0.20 / 67.18	23.03 / 0.69 / 0.30 / 62.04	22.92 / 0.77 / 0.18 / 61.33	– / – / – / –	– / – / – / –	– / – / – / –
Forest Scene #4	25.01 / 0.83 / 0.17 / 70.39	24.68 / 0.78 / 0.23 / 68.03	26.08 / 0.86 / 0.12 / 77.03	– / – / – / –	– / – / – / –	– / – / – / –
Forest Scene #5	24.24 / 0.82 / 0.19 / 70.36	22.99 / 0.71 / 0.30 / 60.03	24.65 / 0.84 / 0.16 / 71.34	– / – / – / –	– / – / – / –	– / – / – / –
Forest Scene #6	25.25 / 0.83 / 0.18 / 70.97	23.72 / 0.73 / 0.28 / 60.22	24.29 / 0.80 / 0.20 / 63.61	– / – / – / –	– / – / – / –	– / – / – / –
Average (large scenes)	**24.84** / 0.82 / 0.18 / **69.98**	24.04 / 0.74 / 0.27 / 64.08	24.72 / **0.83** / **0.17** / 68.42	– / – / – / –	– / – / – / –	– / – / – / –
Average (all)	**25.94** / **0.85** / 0.15 / **71.96**	25.67 / 0.80 / 0.21 / 69.27	25.92 / **0.85** / **0.13** / 70.01	– / – / – / –	– / – / – / –	– / – / – / –

Table 1: *Quantitative metrics for our method compared to 4D-GS [WYF*24], 4D-Scaffold-GS [CCK*25], AmbientGaussians [SHK*24], SC-GS [HSY*23] and 4dgs [YYPZ24].*

The total velocity distortion $\mathcal{L}_{\text{vel}}$ is the average over all pixels of an image.

The full loss is:

$$\mathcal{L} = \mathcal{L}_{\text{photo}} + \lambda_{\text{geo}}\mathcal{L}_{\text{geo}} + \lambda_{\text{vel}}\mathcal{L}_{\text{vel}} \tag{33}$$

where we use $\lambda_{\text{geo}} = \lambda_{\text{vel}} = 15$.

5. Results and Evaluation

In order to evaluate our method we use the AmbientGaussians dataset [SHK*24], which features long monocular videos of dynamic outdoor scenes with large amounts of motion of leaves and vegetation. Additional results are provided in Appendix E. We show comparisons to methods that can run these sequences without modification. These are methods that do not require user interaction or point tracking for initialization. User interaction is impractical for segmenting leaves and point trackers run out of memory at these sequence lengths and resolutions. We run all experiments on a machine with an NVIDIA A100 GPU with 40GB of VRAM. We include a supplemental video with selected and annotated sequences and comparisons, and an additional video with the full unedited set of sequences.

5.1. Evaluation Protocol

The typical evaluation protocol involves holding out every 8th image for the test set [KKLD23]. However, for captures from monocular video, we find the test views are too similar to the training views, and thus do not provide adequate evaluation of the reconstruction of motion. We perform our quantitative evaluation by holding out every 8th segment, where each segment consists of four consecutive frames. We report reconstruction accuracy on these held-out frames using PSNR, SSIM, LPIPS, and VMAF [LLLK13], which partially accounts for cross frame consistency. Additionally, we present qualitative results under two standard settings: moving the camera at fixed time to evaluate spatial consistency, and evolving time from a fixed viewpoint to assess motion modeling: please see supplemental video and Fig. 6 and 7.

5.2. Quantitative Results and Efficiency

We report results of quantitative evaluation of our method. The AmbientGaussians dataset contains some scenes that are more "*urban*", typically with a small number of bushes in an urban environment, and a set of "*forest*" scenes which are set in outdoor forests. Some of the latter have larger spatial extent, making them more challeng-

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

Scene	Ours FPS↑ / Gauss.↓ / Time↓	4D-GS [WYF*24] FPS↑ / Gauss.↓ / Time↓	4D-Scaffold-GS [CCK*25] FPS↑ / Gauss.↓ / Time↓	AmbientGaussians [SHK*24] FPS↑ / Gauss.↓ / Time↓	SC-GS [HSY*23] FPS↑ / Gauss.↓ / Time↓	4dgs [YYPZ24] FPS↑ / Gauss.↓ / Time↓
Urban Scene #1	314.54 / 0.28M / 0h 37m	69.51 / 0.36M / 0h 35m	50.47 / 0.45M / 0h 21m	19.35 / 0.66M / 7h 14m	33.84 / 1.37M / 2h 12m	22.26 / 3.79M / 5h 10m
Urban Scene #2	304.58 / 0.24M / 0h 39m	65.53 / 0.23M / 0h 26m	29.66 / 1.12M / 0h 36h	34.66 / 0.37M / 4h 45m	39.56 / 1.02M / 3h 27m	25.61 / 4.81M / 6h 59m
Urban Scene #3	222.06 / 0.39M / 0h 50m	56.10 / 0.36M / 0h 36m	76.41 / 0.55M / 0h 20h	18.03 / 0.75M / 7h 52m	23.79 / 1.76M / 3h 17m	24.79 / 5.47M / 6h 41m
Urban Scene #4	208.00 / 0.29M / 0h 49m	84.66 / 0.23M / 0h 26m	17.92 / 2.29M / 1h 01m	25.47 / 0.49M / 8h 14m	44.82 / 0.97M / 1h 58m	23.21 / 7.55M / 1h 36m
Average (small scenes)	**262.30** / 0.30M / 0h 44m	43.62 / **0.30M** / **0h 31m**	57.41 / 1.10M / 0h 35m	24.38 / 0.57M / 7h 1m	35.50 / 1.28M / 2h 43m	23.97 / 5.40M / 5h 6m
Forest Scene #1	103.54 / 1.30M / 2h 10m	67.65 / 0.37M / 0h 56m	62.62 / 1.17M / 0h 22m	– / – / –	– / – / –	– / – / –
Forest Scene #2	94.98 / 0.57M / 1h 01m	75.33 / 0.19M / 0h 24m	14.91 / 5.49M / 1h 44m	– / – / –	– / – / –	– / – / –
Forest Scene #3	108.46 / 1.88M / 3h 13m	68.16 / 0.36M / 0h 36m	54.17 / 2.02M / 0h 29m	– / – / –	– / – / –	– / – / –
Forest Scene #4	152.96 / 0.59M / 0h 58m	68.72 / 0.36M / 0h 35m	20.58 / 3.58M / 1h 10m	– / – / –	– / – / –	– / – / –
Forest Scene #5	93.41 / 1.04M / 1h 56m	55.47 / 0.37M / 0h 54m	20.74 / 3.28M / 1h 02m	– / – / –	– / – / –	– / – / –
Forest Scene #6	91.87 / 1.32M / 2h 39m	53.28 / 0.37M / 1h 14m	54.93 / 1.39M / 0h 30m	– / – / –	– / – / –	– / – / –
Average (large scenes)	**107.54** / 1.12M / 2h 0m	37.99 / **0.34M** / **0h 47m**	41.45 / 2.82M / 0h 53m	– / – / –	– / – / –	– / – / –
Average (all)	**169.44** / 0.79M / 1h 29m	66.44 / **0.32M** / **0h 40m**	40.24 / 2.13M / 0h 46m	– / – / –	– / – / –	– / – / –

Table 2: *Efficiency metrics: runtime (FPS), number of Gaussians, and training time for our method in small scenes compared to 4D-GS [WYF*24], 4D-Scaffold-GS [CCK*25], AmbientGaussians [SHK*24], SC-GS [HSY*23] and 4dgs [YYPZ24].*

ing, especially for methods with an implicit, grid-based representation, that implies limitations in resolution for parts of these scenes that have fine detail.

In Table 1, we compare the quantitative image quality metrics for our method to 4D-GS [WYF*24], 4D-Scaffold-GS [CCK*25], AmbientGaussians [SHK*24], SC-GS [HSY*23], and 4dgs [YYPZ24] that is the state-of-the-art explicit method on the urban scenes. For the forest scenes only 4D-GS managed to run; the other methods fail due to memory requirements. We thus report the average for the two subsets (urban, forest) in addition to the overall average. In Table 2, we further compare the runtime, number of Gaussians, and training time.

These comparisons show that on average over all scenes our method has better image quality compared to all previous methods. The quality improvement is significant for all methods, i.e., over the (mostly) explicit methods SC-GS and 4dgs, but also compared to AmbientGaussians. Rendering speed of our method is on average 2.5 faster compared to the next best method (4D-GS).

For the larger forest scenes (1, 3, 4 and 5), our methods results in significantly higher quality reconstruction, while for smaller scenes (urban, and forest 2), it is on-par or worse that 4D-GS. In all cases however, we have significantly faster rendering speed. As we can see from Table 2, our method adds more Gaussian primitives in suitable positions, improving quality, while 4D-GS on the other hand is unable to densify appropriately in these scenes. We experimented the densification threshold in 4D-GS using several values to try and increase the number of Gaussians created by 4D-GS, but were unable to get a significantly higher number of primitives nor better quality. A possible explantion for this is the original densification algorithm's inability to adapt to sparse gradients and its suboptimal placement of new Gaussians, especially in large scenes where complex visibility makes thresholding hard to tune.

5.3. Qualitative Analysis

Fig. 6 and 7 as well as the supplemental video highlight the visual improvements of our method compared to 4D-GS [WYF*24], 4D-Scaffold-GS [CCK*25], AmbientGaussians [SHK*24], SC-GS [HSY*23] and 4dgs [YYPZ24]. We observe in particular that our method is able to address static and dynamic regions equally well, resulting in significantly improved background reconstruction in the presence of motion, and better overall visual quality. The supplemental video shows several examples of how our method improves quality in forest and urban scenes.

5.4. Ablations

We next evaluate the effect of each of our individual contributions to the overall result of our method, summarized in Table 3.

Simple split is an alternative splitting strategy where we use our error approach, but simply split the Gaussians with a plane perpendicular to the largest axis at the midpoint during densification similar to [FCC*24, DDL*25b]; note that this baseline uses the error as well.

The original densification from the 3DGS paper [KKLD23] results in a significant drop in quality when compared to our error-informed method.

Our image ARAP loss slightly reduces the quality metrics, which is a cost to pay for improved temporal consistency (which is not reflected in the metrics). This can be seen in Fig. 9, where structure is visibly preserved. Unfortunately existing metric, including VMAF are unable to quantify this effect.

Velocity regularization serves the same purpose as the ARAP loss and results in a small improvement in quality.

Method	PSNR↑	SSIM↑	LPIPS↓	VMAF↑
Ours	25.95	0.848	0.101	71.96
Ours w/ simple split	25.86	0.846	0.099	71.55
Ours w/ original densification	22.56	0.683	0.289	47.90
Ours w/o image ARAP	26.15	0.852	0.099	73.18
Ours w/o velocity regularization	25.92	0.847	0.101	71.77
Ours w/o Weighted Adam	20.42	0.551	0.258	28.34

Table 3: *Ablations of the different components of our algorithm.*

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

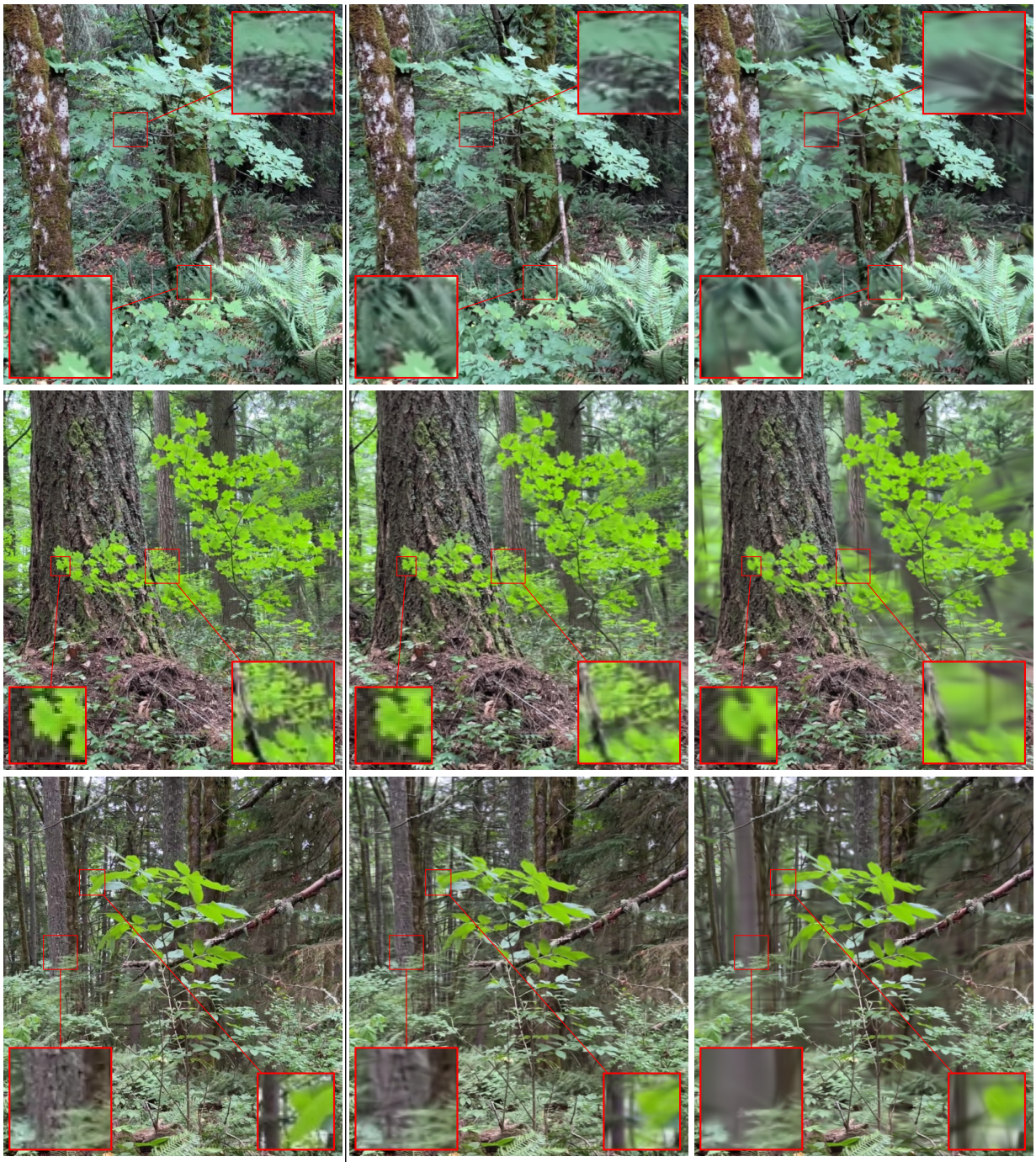

Figure 7: *Qualitative comparison of the performance of our method for the large forest scenes. We see clear improvement in quality, in part thanks to our error guided densification.*

Weighted Adam brings a significant improvement to the quality of our results. This result confirms our intuition that the optimization dynamics are very different compared to the standard 3DGS; our adaptive spatio-temporal subdivision makes this more pronounced. Our weighted Adam addresses this issue.

5.5. Keyframe distribution and memory overhead

In order to demonstrate the ability of our temporal densification to focus on the moving parts of a scene we examine how the number of keyframes varies over the Gaussians. In Table 4 we show

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

Ours Standard Adam Ours Standard Adam

Figure 8: *Standard Adam updates all parameters on every iteration irrespectively of whether their contribution to the rendered image, leading to divergence, particularly in cases when the primitives are obscured, or when they have many keyframes.*

Ours No ARAP Ours No ARAP

Figure 9: *We show renderings from the perspective of an input camera, with the configuration of the Gaussians being evaluated at a time different that camera's time. The use of our image-based geometric loss better manages to preserve structure.*

the average number of keyframes per scene. The average is well bellow ¼ of the length of the sequence, which is the maximum resulting from our choice of l_{seg}. Because of the large disparity in the length of the sequences we also compute the number of keyframes relative to this length. Fig. 11 displays the histogram – averaged across all scenes – of the relative number of keyframes. Two spikes are present. The tallest one, amassing most of the distribution over small values, corresponds to the fact that the majority of a scene does not exhibit significant motion. The other reflects the leaves and branches that do move throughout the sequence. We additionally visualize the spatial variation of the number of keyframes in Fig. 10, where the same situation is observed, Gaussians lying on the moving leaves have high number of keyframes, while Gaussians on static regions have just 1 or close to that.

6. Discussion and Conclusions

We introduced an explicit spatio-temporal Gaussian representation for the reconstruction of dynamic scenes with incoherent small-scale motion, such as foliage oscillating in the wind. We based this representation on a set of per-Gaussian keyframed deformations with adaptive spatio-temporal refinement. We guided this refinement together with the spatial densification through a new strategy based on the moments of the reconstruction error. Finally, we improved the optimization both with new image-based temporal regularizations and a visibility-aware weighting of the Adam scheme that is especially important for dynamic scenes where keyframes are rarely visible. Our experiments demonstrate that our method provides much faster rendering speeds than all previous methods, which is critical in the adoption of dynamic Gaussian splatting so-

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

	No Frames	Avg. No Keyframes	Avg. Rel. No Keyframes
Urban Scene #1	218	8.4	3.8%
Urban Scene #2	189	17.7	9.3%
Urban Scene #3	256	11.6	4.5%
Urban Scene #4	298	33.8	11.3%
Forest Scene #1	825	9.1	1.1%
Forest Scene #2	323	12.1	3.7%
Forest Scene #3	1088	10.4	1.0%
Forest Scene #4	364	10.7	2.9%
Forest Scene #5	734	24.9	3.4%
Forest Scene #6	743	33.4	4.5%
Average	504	17.2	4.6%

Table 4: *Average number of keyframes across scenes, as well as relative to the number of frames in the sequence.*

lutions, while achieving better visual quality. For larger scenes, the visual improvement is significant, since our method manages to clearly distinguish static background from motion.

Our method shares the limitation of most other methods that attempt to reconstruct dynamic 3D scenes, and that come from the ill-posedness of this problem. Specifically, depth imprecision of SfM points, which is prominent in the background, manifests false motion since keyframes are erroneously added to adapt to the discrepancy. This is a common pitfall of monocular methods that do not employ learned priors as can be seen in the supplemental video.

Figure 10: *Visualization of the spatial variation of the keyframes for a frame from Urban Scene #2 with length 188. Above, the rendered frame. Below, number of keyframes.*

Besides, scenes with complex changes in illumination, reflections, shadows, etc., are particularly challenging. Note, as well, that our method was also focused on small foliage-like dynamics, and can be less efficient for large motions of a single dominant object, for which methods based on tracking and optical flow are still superior.

Finally, future work includes the generalization of our method to broader classes of movements to encompass complex outdoor scenes, with vehicles, humans, etc. Our method outputs a precise displacement for each Gaussian, which could be the basis for inferring the physical properties of the dynamic objects in the scene and ultimately enable physically realistic interactions with the reconstructed objects.

7. Acknowledgements

This work was funded by the European Union, European Research Council (ERC) Advanced Grant NERPHYS, 101141721 https://project.inria.fr/nerphys. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). The authors would also like to thank Adobe and NVIDIA for software and hardware donations.

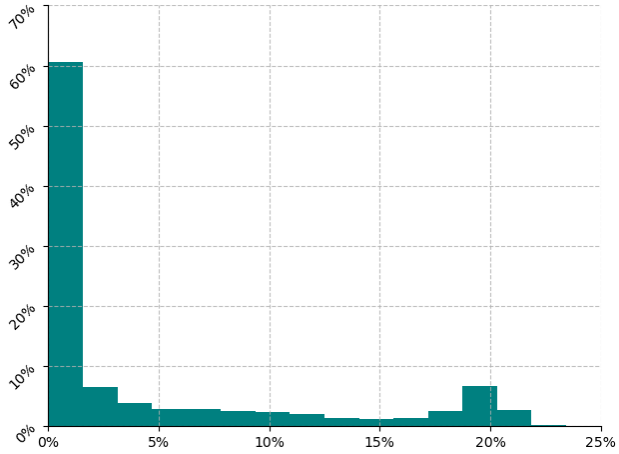

Figure 11: *Histogram of the relative number of keyframes averaged across scenes.*

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

Appendix A: Derivation for the Second Moment

We utilize properties of the vec operation [Min00], which transforms a matrix to a vector by concatenating its rows:

$$\|A\|_F = \|\operatorname{vec}(A)\| \tag{34}$$

$$\operatorname{vec}(AXB) = \left(B^T \otimes A\right)\operatorname{vec}(X) \tag{35}$$

where $\otimes$ is the Kronecker product. Also, in order to express the fact that S_g is symmetric we can use the operation that vectorizes only the upper triangular part of the matrix vech. For a symmetric $n \times n$ matrix A the two operations relate to each other via the elimination and duplication matrices [MN80], respectively L_n and D_n:

$$\operatorname{vech}(A) = L_n \operatorname{vec}(A) \tag{36}$$

$$\operatorname{vec}(A) = D_n \operatorname{vech}(A) \tag{37}$$

Using the above the objective 19 becomes:

$$\min_{\boldsymbol{S}_g} \sum_{I\in\mathcal{I}} w_{g,I} \left\|\operatorname{vec}(\boldsymbol{S}_{g,I}) - B_{g,I} D_3 \operatorname{vech}(\boldsymbol{S}_g)\right\|^2 \tag{38}$$

$$B_{g,I} = A_{g,I} \otimes A_{g,I} \tag{39}$$

Similarly to the first moments the solution is:

$$\operatorname{vech}(\boldsymbol{S}_g) = \left(\sum_{I\in\mathcal{I}} e_{g,I} D_3^T B_{g,I}^T B_{g,I} D_3\right)^{-1} \sum_{I\in\mathcal{I}} e_{g,I} D_3^T B_{g,I}^T \operatorname{vec}(\boldsymbol{S}_{g,I}) \tag{40}$$

Appendix B: Clamping of Moments

In some cases, due to inconsistencies across views, the solutions for the moments might lie outside what is taken to be the effective extent of the Gaussian (3 times the standard deviation). For this reason we perform a kind of clamping on the moments. Specifically for the first moment, we clamp its norm to be less then 3 times the directional standard deviation:

$$\boldsymbol{r}_g^{\lceil\rceil} = \frac{3\,\boldsymbol{r}_g}{\max\left(3, \sqrt{\boldsymbol{r}_g^T \Sigma_g^{-1} \boldsymbol{r}_g}\right)} \tag{41}$$

For the second moment we make sure that it is smaller than the Gaussian's covariance matrix in the partial ordering of positive semidefinite matrices. If, for a symmetric matrix S, we use $\lfloor S \rfloor$ to denote the operation preserving its non-negative eigenvalues, whilst mapping its negative ones to zero, then the clamped second moment is:

$$\boldsymbol{S}_g^{\lceil\rceil} = \boldsymbol{S}_g - \lfloor \boldsymbol{S}_g - \Sigma_g \rfloor \tag{42}$$

Finally, we centralize the the second moment and make sure it remains positive semidefinite:

$$\boldsymbol{S}_g^c = \left\lfloor \boldsymbol{S}_g^{\lceil\rceil} - \boldsymbol{r}_g^{\lceil\rceil} \left(\boldsymbol{r}_g^{\lceil\rceil}\right)^T \right\rfloor \tag{43}$$

Appendix C: Weighted Adam Updates

Given the weight $\eta_i \in [0,1]$ that expresses how informative a gradient of a parameter is for the iteration i, the update equations become:

$$g_i = \frac{\nabla f_i(\theta_{i-1})}{\eta_i} \tag{44}$$

$$m_i = \left(1-(1-\beta_1)\eta_i\right) m_{i-1} + (1-\beta_1)\eta_i g_i \tag{45}$$

$$\upsilon_i = \left(1-(1-\beta_2)\eta_i\right) \upsilon_{i-1} + (1-\beta_2)\eta_i g_i^2 \tag{46}$$

$$b_i^m = \left(1-(1-\beta_1)\eta_i\right) b_{i-1}^m + (1-\beta_1)\eta_i \tag{47}$$

$$b_i^\upsilon = \left(1-(1-\beta_2)\eta_i\right) b_{i-1}^\upsilon + (1-\beta_2)\eta_i \tag{48}$$

$$\hat{m}_i = \frac{m_i}{b_i^m} \tag{49}$$

$$\hat{\upsilon}_i = \frac{\upsilon_i}{b_i^\upsilon} \tag{50}$$

$$\theta_i = \theta_{i-1} - \gamma\,\eta_i \frac{\hat{m}_i}{\sqrt{\hat{\upsilon}_i} + \epsilon} \tag{51}$$

Note that the bias correction terms b_i^m and b_i^υ are not given by a closed form expression as in the original Adam and need to be accumulated as well.

For our purposes we use the average visibility defined in equation 28 as the weight η.

Parameter	Learning rate
Centers* (μ)	$3.2e^{-4} \cdot s_e$
Covariance quaternions	$4e^{-3}$
Covariance scales	$2e^{-2}$
Opacities (o)	$1e^{-1}$
0th order spherical harmonics coefficients	$1e^{-2}$
Other spherical harmonics coefficients	$5e^{-4}$
Translations* ($\Delta\mu$)	$3.2e^{-4} \cdot s_e$
Rotation quaternions (q)	$4e^{-3}$

Table 5: *Learning rates used for optimization. For the centers and the translations (*) the unmodulated learning rate is reported.*

Description	Value
Spatial splitting threshold (τ_{space})	$5e^{-7}$
Temporal splitting threshold (τ_{time})	$8e^{-1}$
Minimum segment size (l_{seg})	4
Temporal densification neighbors' number (K)	10
Kernel size for image-based geometric loss	9
Sigma for color kernel $k_{c,c'}$	$2e^{-2}$
Sigma for depth kernel $k_{d,d'}$	$1e^{-2} \cdot s_e$
Sigma for image space kernel $k_{p,p'}$	5
SSIM loss multiplier (λ_{SSIM})	$2e^{-1}$
Image based geometric loss multiplier (λ_{geo})	15
Velocity distortion loss multiplier (λ_{vel})	15

Table 6: *Values of parameters used in our method.*

Appendix D: Learning rates and other parameter values

We summarize the learning rates used for our experiments in Table 5. Note that for center and the translations the unmodulated

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

Scene	Ours	4D-Scaffold-GS [CCK*25]
	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑	PSNR↑ / SSIM↑ / LPIPS↓ / VMAF↑
coffee_martini	29.65 / 0.92 / 0.13 / 70.50	28.50 / 0.92 / 0.14 / 70.32
cook_spinach	32.49 / 0.95 / 0.13 / 68.78	32.81 / 0.96 / 0.13 / 69.58
cut_roasted_beef	30.85 / 0.94 / 0.13 / 62.01	33.33 / 0.95 / 0.15 / 66.50
flame_salmon	29.85 / 0.93 / 0.13 / 71.77	27.72 / 0.93 / 0.13 / 71.71
flame_steak	32.70 / 0.96 / 0.12 / 66.64	32.54 / 0.96 / 0.13 / 65.48
sear_steak	33.12 / 0.96 / 0.12 / 73.04	32.39 / 0.96 / 0.12 / 69.53
Average	**31.44** / 0.94 / **0.13** / **68.79**	31.21 / **0.95** / **0.13** / 68.24

Table 7: *Quantitative evaluation of our method on the Neural 3D Video Datasets [LSZ*22] compared to 4D-Scaffold-GS [CCK*25]*

learning rate is reported. The values of other hyperparameters are given in Table 6. Similar to the original method, specific hyperparameters that affect spatial quantities are multiplied by the extent of the scene s_e.

Appendix E: Additional results

Even though we developed our method with monocular input in mind, it can easily be modified to be applied in a multi-view context. Specifically, during densification we can sample one of the cameras for each timestep to compute the error moments.

Applying this modification, we test our method against 4D-Scaffold-GS [CCK*25] on the Neural 3D Video Dataset [LSZ*22]. The quality metrics are reported in Table 7. Our method does not explicitly handle transient effects like fire that are present in this dataset, in contrast to 4D-Scaffold-GS which allows opacity to vary over time. Nevertheless, our method achieves competitive results, even surpassing 4D-Scaffold-GS in four out of six scenes.

References

[AHR*23] ATTAL B., HUANG J.-B., RICHARDT C., ZOLLHOEFER M., KOPF J., O'TOOLE M., KIM C.: HyperReel: High-fidelity 6-DoF video with ray-conditioned sampling. In *CVPR* (2023). 2

[Ale02] ALEXA M.: Linear combination of transformations. *ACM Trans. Graph. 21*, 3 (July 2002), 380–387. URL: https://doi.org/10.1145/566654.566592, doi:10.1145/566654.566592. 7

[BMV*22] BARRON J. T., MILDENHALL B., VERBIN D., SRINIVASAN P. P., HEDMAN P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR* (2022). 2, 7

[BMV*23] BARRON J. T., MILDENHALL B., VERBIN D., SRINIVASAN P. P., HEDMAN P.: Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV* (2023). 2

[BPK24] BULÒ S. R., PORZI L., KONTSCHIEDER P.: Revising densification in gaussian splatting, 2024. arXiv:2404.06109. 2, 4, 5

[CCK*25] CHO W. O., CHO I., KIM S., BAE J., UH Y., KIM S. J.: 4d scaffold gaussian splatting with dynamic-aware anchor growing for efficient and high-fidelity dynamic scene reconstruction. In *Arxiv* (2025). 2, 8, 9, 14

[CCZ*24] CHEN Y., CHEN Z., ZHANG C., WANG F., YANG X., WANG Y., CAI Z., YANG L., LIU H., LIN G.: Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In *CVPR* (2024). 2

[DDL*25a] DENG X., DIAO C., LI M., YU R., XU D.: Improving densification in 3d gaussian splatting for high-fidelity rendering, 2025. arXiv:2508.12313. 2

[DDL*25b] DENG X., DIAO C., LI M., YU R., XU D.: Improving densification in 3d gaussian splatting for high-fidelity rendering, 2025. URL: https://arxiv.org/abs/2508.12313, arXiv:2508.12313. 2, 6, 9

[DWD*24] DUAN Y., WEI F., DAI Q., HE Y., CHEN W., CHEN B.: 4d-rotor gaussian splatting: Towards efficient novel-view synthesis for dynamic scenes. In *Proc. SIGGRAPH* (2024). 1, 2

[FCC*24] FENG Q.-Y., CAO G.-C., CHEN H.-X., XU Q.-C., MU T.-J., MARTIN R., HU S.-M.: Evsplitting: An efficient and visually consistent splitting algorithm for 3d gaussian splatting. In *SIGGRAPH Asia 2024 Conference Papers* (New York, NY, USA, 2024), SA '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3680528.3687592, doi:10.1145/3680528.3687592. 2, 6, 9

[FRF*23] FRANKE L., RÜCKERT D., FINK L., INNMANN M., STAMMINGER M.: Vet: Visual error tomography for point cloud completion and high-quality neural rendering. In *ACM SIGGRAPH Asia Conference Proceedings* (2023). 4

[GSKH21] GAO C., SARAF A., KOPF J., HUANG J.-B.: Dynamic view synthesis from dynamic monocular video. In *ICCV* (2021). 3

[HSY*23] HUANG Y.-H., SUN Y.-T., YANG Z., LYU X., CAO Y.-P., QI X.: Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. *arXiv preprint arXiv:2312.14937* (2023). 1, 8, 9

[HYC*24] HUANG B., YU Z., CHEN A., GEIGER A., GAO S.: 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers* (2024), Association for Computing Machinery. doi:10.1145/3641519.3657428. 7

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3 (July 2005), 1134–1141. URL: https://doi.org/10.1145/1073204.1073323, doi:10.1145/1073204.1073323. 7

[JKK*23] JAMBON C., KERBL B., KOPANAS G., DIOLATZIS S., LEIMKÜHLER T., DRETTAKIS G.: Nerfshop: Interactive editing of neural radiance fields". *Proceedings of the ACM on Computer Graphics and Interactive Techniques 6*, 1 (2023). 2

[KB14] KINGMA D., BA J.: Adam: A method for stochastic optimization. *International Conference on Learning Representations* (12 2014). 7

[KKLD23] KERBL B., KOPANAS G., LEIMKÜHLER T., DRETTAKIS G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics 42*, 4 (2023). 1, 2, 3, 8, 9

[KRS*24] KHERADMAND S., REBAIN D., SHARMA G., SUN W., TSENG Y.-C., ISACK H., KAR A., TAGLIASACCHI A., YI K. M.: 3d gaussian splatting as markov chain monte carlo. In *Advances in Neural Information Processing Systems (NeurIPS)* (2024). Spotlight Presentation. 2

[KVN24] KATSUMATA K., VO D. M., NAKAYAMA H.: A compact dynamic 3d gaussian representation for real-time dynamic view synthesis. In *ECCV* (2024). 3

[LCLX24] LI Z., CHEN Z., LI Z., XU Y.: Spacetime gaussian feature splatting for real-time dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 8508–8520. 2

[LGM*23] LIU Y.-L., GAO C., MEULEMAN A., TSENG H.-Y., SARAF A., KIM C., CHUANG Y.-Y., KOPF J., HUANG J.-B.: Robust dynamic radiance fields. In *CVPR* (2023). 3

[LHND25] LIU Y.-C., HÖLLEIN L., NIESSNER M., DAI A.: Quicksplat: Fast 3d surface reconstruction via learned gaussian initialization. In *ICCV* (2025). 2

[LHY*24] LIU J., HU W., YANG Z., CHEN J., WANG G., CHEN X., CAI Y., GAO H.-A., ZHAO H.: Rip-nerf: Anti-aliasing radiance fields with ripmap-encoded platonic solids. In *ACM SIGGRAPH Conference Proceedings* (2024). 2

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410

[LKL*25] LIANG Y., KHAN N., LI Z., NGUYEN-PHUOC T., LANMAN D., TOMPKIN J., XIAO L.: Gaufre: Gaussian deformation fields for real-time dynamic novel view synthesis. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2025). 2

[LKLR24] LUITEN J., KOPANAS G., LEIBE B., RAMANAN D.: Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV* (2024). 1, 7

[LLLK13] LIU T.-J., LIN Y.-C., LIN W., KUO C.-C. J.: Visual quality assessment: recent developments, coding applications and future trends. *APSIPA Transactions on Signal and Information Processing 2* (2013), e4. doi:10.1017/ATSIP.2013.5. 8

[LSS*19] LOMBARDI S., SIMON T., SARAGIH J., SCHWARTZ G., LEHRMANN A., SHEIKH Y.: Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics* (2019). 2

[LSS*21] LOMBARDI S., SIMON T., SCHWARTZ G., ZOLLHOEFER M., SHEIKH Y., SARAGIH J.: Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics 40*, 4 (2021). 2

[LSZ*22] LI T., SLAVCHEVA M., ZOLLHOEFER M., GREEN S., LASSNER C., KIM C., SCHMIDT T., LOVEGROVE S., GOESELE M., NEWCOMBE R., LV Z.: Neural 3d video synthesis from multi-view video. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 5511–5521. doi:10.1109/CVPR52688.2022.00544. 14

[LWC*23] LI Z., WANG Q., COLE F., TUCKER R., SNAVELY N.: Dynibar: Neural dynamic image-based rendering. In *CVPR* (2023). 3

[LWH*24] LEI J., WENG Y., HARLEY A., GUIBAS L., DANIILIDIS K.: Mosca: Dynamic gaussian fusion from casual videos via 4d motion scaffolds, 2024. arXiv:2405.17421. 2, 3, 7

[LWJ*24] LEE J., WON C., JUNG H., BAE I., JEON H.-G.: Fully explicit dynamic gaussian splatting. In *Proceedings of the Neural Information Processing Systems* (2024). 3, 4

[MGK*24] MALLICK S. S., GOEL R., KERBL B., STEINBERGER M., CARRASCO F. V., DE LA TORRE F.: Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers* (2024). 2, 7

[Min00] MINKA T. P.: Old and new matrix algebra useful for statistics. URL: https://api.semanticscholar.org/CorpusID:15971655. 5, 13

[MN80] MAGNUS J. R., NEUDECKER H.: The elimination matrix: Some lemmas and applications. *SIAM Journal on Algebraic Discrete Methods 1*, 4 (1980). 13

[MSL*25] MEULEMAN A., SHAH I., LANVIN A., KERBL B., DRETTAKIS G.: On-the-fly reconstruction for large-scale novel view synthesis from unposed images. *ACM Transactions on Graphics 44*, 4 (2025). 2

[PBB*25] PARK J., BUI M.-Q. V., BELLO J. L. G., MOON J., OH J., KIM M.: Splinegs: Robust motion-adaptive spline for real-time dynamic 3d gaussians from monocular video. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)* (2025). 2, 3

[PPGT*23] PETITJEAN A., POIRIER-GINTER Y., TEWARI A., CORDONNIER G., DRETTAKIS G.: Modalnerf: Neural modal analysis and synthesis for free-viewpoint navigation in dynamically vibrating scenes. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 42*, 4 (2023). 2

[PSB*21] PARK K., SINHA U., BARRON J. T., BOUAZIZ S., GOLDMAN D. B., SEITZ S. M., MARTIN-BRUALLA R.: Nerfies: Deformable neural radiance fields. *ICCV* (2021). 7

[RWL*22] RÜCKERT D., WANG Y., LI R., IDOUGHI R., HEIDRICH W.: Neat: Neural adaptive tomography. *ACM Transactions on Graphics 41*, 4 (2022). 4

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Goslar, DEU, 2007), SGP '07, Eurographics Association, p. 109–116. 7

[SF16] SCHÖNBERGER J. L., FRAHM J.-M.: Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). 4

[SHK*24] SHIH M.-L., HUANG J.-B., KIM C., SHAH R., KOPF J., GAO C.: Modeling ambient scene dynamics for free-view synthesis. In *ACM SIGGRAPH Conference Papers* (2024). 2, 3, 8, 9

[SHU*24] STEARNS C., HARLEY A. W., UY M., DUBOST F., TOMBARI F., WETZSTEIN G., GUIBAS L.: Dynamic gaussian marbles for novel view synthesis of casual monocular videos. In *SIGGRAPH Asia Conference Papers* (2024). 3

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)* (1998), 839–846. URL: https://api.semanticscholar.org/CorpusID:14308539. 7

[TTG*21] TRETSCHK E., TEWARI A., GOLYANIK V., ZOLLHÖFER M., LASSNER C., THEOBALT C.: Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *ICCV* (2021). 3

[WYF*24] WU G., YI T., FANG J., XIE L., ZHANG X., WEI W., LIU W., TIAN Q., WANG X.: 4d gaussian splatting for real-time dynamic scene rendering. In *CVPR* (2024). 1, 2, 8, 9, 10

[WYG*25] WANG Q., YE V., GAO H., ZENG W., AUSTIN J., LI Z., KANAZAWA A.: Shape of motion: 4d reconstruction from a single video. In *International Conference on Computer Vision (ICCV)* (2025). 2

[WYX*25] WANG Y., YANG P., XU Z., SUN J., ZHANG Z., CHEN Y., BAO H., PENG S., ZHOU X.: Freetimegs: Free gaussian primitives at anytime anywhere for dynamic scene reconstruction. In *CVPR* (2025). 2

[XXY*24] XU Z., XU Y., YU Z., PENG S., SUN J., BAO H., ZHOU X.: Representing long volumetric video with temporal gaussian hierarchy. *ACM Transactions on Graphics 43*, 6 (November 2024). URL: https://zju3dv.github.io/longvolcap. 2

[XZQ*23] XIE T., ZONG Z., QIU Y., LI X., FENG Y., YANG Y., JIANG C.: Physgaussian: Physics-integrated 3d gaussians for generative dynamics, 2023. arXiv:2311.12198. 2

[YGZ*24] YANG Z., GAO X., ZHOU W., JIAO S., ZHANG Y., JIN X.: Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *CVPR* (2024). 1, 2, 3

[YSL*22] YUAN Y.-J., SUN Y.-T., LAI Y.-K., MA Y., JIA R., GAO L.: Nerf-editing: geometry editing of neural radiance fields. In *CVPR* (2022), pp. 18353–18364. 2

[YXL*25] YAO W., XIE S., LI L., ZHANG W., LAI Z., DAI S., ZHANG K., WANG Z.: Sd-gs: Structured deformable 3d gaussians for efficient dynamic scene reconstruction, 2025. URL: https://arxiv.org/abs/2507.07465, arXiv:2507.07465. 2

[YYPZ24] YANG Z., YANG H., PAN Z., ZHANG L.: Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)* (2024). 1, 2, 8, 9

[ZHL*24] ZHANG Z., HU W., LAO Y., HE T., ZHAO H.: Pixel-gs: Density control with pixel-aware gradient for 3d gaussian splatting. In *ECCV* (2024). 2

This is the author version of the paper, the final version is published here:
https://doi.org/10.1111/cgf.70410