On-the-fly Reconstruction for Large-Scale Novel View Synthesis from Unposed Images

ANDREAS MEULEMAN, ISHAAN SHAH, and ALEXANDRE LANVIN, Inria, Université Côte d'Azur, France
BERNHARD KERBL, TU Wien, Austria
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

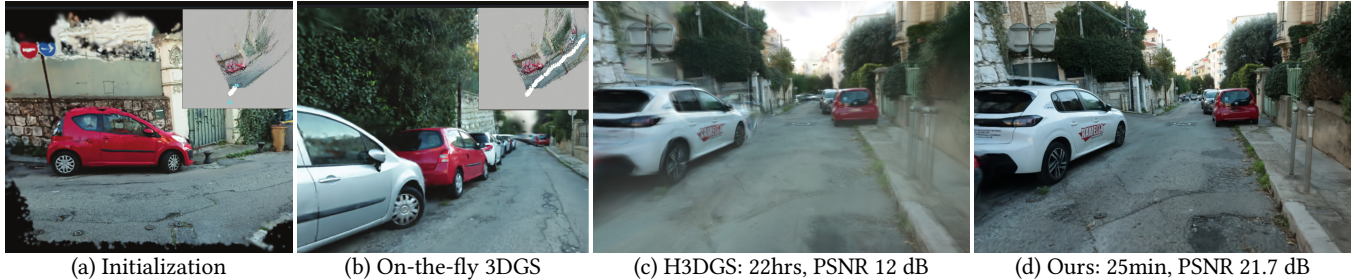| (a) Initialization | (b) On-the-fly 3DGS | (c) H3DGS: 22hrs, PSNR 12 dB | (d) Ours: 25min, PSNR 21.7 dB |

Fig. 1. Our method performs on-the-fly reconstruction from an unposed, ordered image sequence. The total processing time for our method is 30min for this sequence of over 4000 images: with our method, the poses and radiance field are *immediately available* after taking the photos. The scene was captured in a 1km walk in 30min, using a camera in "drive mode" taking photos at 3 images/sec, resulting in around 4000 images. Our method incrementally reconstructs a 3D Gaussian representation along with the camera poses. In the middle right, we show a novel view of the same scene reconstructed with Hierarchical 3DGS [Kerbl et al. 2024], that requires 22 hours of processing for camera pose estimation and 3DGS optimization, in contrast to our method (right) that has completed all processing by the time photos are taken. The camera calibration of hierarchical 3DGS fails in many places later on the path, leading to novel view synthesis failure in some places (please see video and additional results at https://repo-sam.inria.fr/nerphys/on-the-fly-nvs/).

Radiance field methods such as 3D Gaussian Splatting (3DGS) allow easy reconstruction from photos, enabling free-viewpoint navigation. Nonetheless, pose estimation using Structure from Motion and 3DGS optimization can still each take between minutes and hours of computation after capture is complete. SLAM methods combined with 3DGS are fast but struggle with wide camera baselines and large scenes. We present an on-the-fly method to produce camera poses and a trained 3DGS *immediately* after capture. Our method can handle dense and wide-baseline captures of ordered photo sequences and large-scale scenes. To do this, we first introduce fast initial pose estimation, exploiting learned features and a GPU-friendly mini bundle adjustment. We then introduce direct sampling of Gaussian primitive positions and shapes, incrementally spawning primitives where required, significantly accelerating training. These two efficient steps allow fast and robust joint optimization of poses and Gaussian primitives. Our incremental approach handles large-scale scenes by introducing scalable radiance field construction, progressively clustering 3DGS primitives, storing them in anchors, and offloading them from the GPU. Clustered primitives are progressively merged, keeping the required scale of 3DGS at any viewpoint. We evaluate our solution on a variety of datasets and show that it can provide on-the-fly processing of all the capture scenarios and scene sizes we target. At the same time our method remains competitive – in speed, image quality, or both – with other methods that only handle specific capture styles or scene sizes.

Authors' addresses: Andreas Meuleman, andreas.meuleman@gmail.com; Ishaan Shah, ishaan.n.shah@gmail.com; Alexandre Lanvin, laanvin@gmail.com, Inria, Université Côte d'Azur, France; Bernhard Kerbl, kerbl@cg.tuwien.ac.at, TU Wien, Austria; George Drettakis, George.Drettakis@inria.fr, Inria, Université Côte d'Azur, France.

CCS Concepts: • **Computing methodologies** → **Rasterization**; **Point-based models**; **Matching**; **Reconstruction**; **Tracking**.

1 INTRODUCTION

Radiance Field solutions for novel-view synthesis [Barron et al. 2022; Duckworth et al. 2023; Müller et al. 2022] take a multi-view dataset of images as input and create a 3D digital version of a real scene, enabling free-viewpoint navigation. Since the recent introduction of 3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] – a fast radiance field method with high visual quality – there has been an explosion in the application of radiance fields in domains as diverse as e-commerce, extended reality, film and video, robotics, 3D generative modelling etc. Most such solutions first need to perform pose estimation for all cameras upfront [Schönberger and Frahm 2016] followed by 3DGS optimization; for typical wide-baseline datasets (tens of) minutes are still required for each of the two steps, even with the fastest methods [Mallick et al. 2024]. For larger scenes, these steps each require many hours of processing after capture has finished, hindering the adoption of such solutions in applications. We present a method for *on-the-fly* radiance field construction that works on consumer hardware: with our solution, camera poses and the scene representation are immediately available by the time scene capture is finished.

While Structure-from-Motion (SfM) [Schönberger and Frahm 2016] followed by 3DGS [Kerbl et al. 2023; Mallick et al. 2024] are too

slow for on-the-fly processing, SLAM solutions are much faster [Tosi et al. 2024], but typically expect dense, video-like input, and often focus less on visual quality. These solutions often struggle with wide-baseline multi-view captures, typically used for radiance-field reconstruction.

Similarly, 3DGS optimization is still too costly for on-the-fly reconstruction. 3DGS optimization starts from a sparse, noisy set of SfM points, while the *densification* process provides additional points from heuristics. As a result, the optimization needs to proceed carefully, requiring many iterations. This results in long training times, even for the most efficient versions [Mallick et al. 2024]. Finally, while some solutions have been proposed to allow 3DGS for large scenes [Kerbl et al. 2024; Liu et al. 2024a; Ren et al. 2024], they all require the poses of all cameras to be estimated beforehand, for scene subdivision and/or hierarchy construction. Our goals are to provide both camera calibration and usable 3DGS reconstruction immediately after capture, either for dense or wide-baseline capture, and most importantly for large-scale scenes. Other methods have difficulty satisfying all of these goals. Note that we require captured images to be given in an ordered sequence, since we present an *incremental* radiance field construction method.

We observe that accurate pose estimation can be costly; if we relax the need for accuracy in a first step, we can reformulate pose estimation to be GPU-friendly and thus significantly faster even if the initial guess is approximate. For such an approach to be possible, joint pose/3DGS estimation needs to subsequently refine these poses. 3DGS differentiable rendering is well-suited to this task, since the gradients from the rendering loss can flow to the poses, improving their accuracy. Finally, for joint optimization to be effective, we also need to accelerate 3DGS; a good strategy to achieve this goal is to directly sample well placed Gaussians, reducing densification and making the optimization easier.

Given these observations, we first propose lightweight initial pose estimation that leverages learning-based matches with image neighbors. The matches and their currently optimized poses are used to quickly estimate the next camera pose. We also reformulate the problem to allow fast GPU-friendly mini bundle adjustment. Second, instead of densifying, we introduce a method for direct sampling of Gaussian primitives. When adding a new image, we directly choose the positions and sizes of 3D Gaussian primitives created, by estimating the probability that a given pixel should spawn a Gaussian, and also directly sample the size of each primitive. This greatly reduces the need for densification, providing the required optimization speed. Given our incremental pose estimation and training, joint optimization is accelerated and the risk of getting stuck in local minima is reduced. Finally, our efficient incremental optimization is naturally suited to a sliding-window clustering and merging approach which we propose, that stores parts of the scene as *anchors* that are placed in space as image capture advances. Taken together, these elements allow our method to incrementally process images in various capture styles, providing camera poses and 3DGS reconstruction on-the-fly.

In summary, our contributions are:

- A fast initial pose estimation method based on deep feature matching and GPU-friendly mini bundle adjustment.

- Probability-based direct sampling of position and shape of Gaussian primitives, greatly alleviating the need for gradient-driven densification resulting in fast 3DGS optimization Together with fast pose estimation, our sampling allows effective incremental joint optimization of poses and the radiance field.
- A sliding window clustering and merging strategy that allows on-the-fly processing of large-scale scenes using *anchors*.

We run evaluations on a variety of datasets, showing that our method is one of the only solutions that can provide on-the-fly processing of all the capture scenarios we target. At the same time it remains competitive – in speed, image quality, or both – with other methods that only handle specific capture styles or scene sizes.

2 RELATED WORK

Novel View Synthesis (NVS) generates images of a scene from viewpoints not observed during capture, allowing free-viewpoint navigation [Barron et al. 2022; Kerbl et al. 2023]. Our contributions are in pose estimation and radiance field optimization for NVS, in particular for large scenes. We briefly review the directly relevant literature for each of these domains, and refer the reader to more complete surveys, e.g., for 3DGS [Chen and Wang 2024; Fei et al. 2024] and SLAM with radiance fields [Tosi et al. 2024].

Camera pose estimation. Pose estimation using SfM is most often a significant part of the overall computation to create the scene representation [Kerbl et al. 2023], given its high computational expense [Kopf et al. 2021; Schönberger and Frahm 2016; Zhao et al. 2022]. Despite recent advances [Brachmann et al. 2024; Duisterhof et al. 2024; Pan et al. 2024; Wang et al. 2024], this process remains too slow for on-the-fly incremental reconstruction. SLAM approaches [Campos et al. 2021; Mur-Artal et al. 2015; Schops et al. 2019] leverage frame-to-frame consistency to estimate camera poses incrementally but struggle with large baselines. Learning-based SLAM methods [Czarnowski et al. 2020; Homeyer et al. 2024; Teed and Deng 2020] improve robustness but are computationally intensive, which quickly becomes prohibitive when optimizing 3D Gaussians simultaneously with pose estimation. Recently, Spann3r [Wang and Agapito 2024] achieved fast pose and point cloud prediction using a transformer-based approach, DUSt3R [Wang et al. 2024]. However, these methods suffer from low-resolution output and substantial pose drift over longer videos. In contrast, we propose a lightweight mini bundle adjustment as a first step by structuring the computation in a GPU-friendly manner; our initial pose estimation is thus fast and does not have to be highly accurate, since we subsequently correct poses during our efficient joint optimization.

Novel view synthesis. Neural volumetric representations [Barron et al. 2021; Lombardi et al. 2019; Mildenhall et al. 2020] have enabled tremendous progress in the field, boasting high visual quality, but suffer from expensive optimization due to ray-marching and large neural networks. More explicit or hybrid representations [Barron et al. 2023; Chen et al. 2022; Hu et al. 2023; Liu et al. 2024b; Müller et al. 2022; Sara Fridovich-Keil and Alex Yu et al. 2022] improve efficiency but remain relatively slow to render and optimize. Recently, 3D Gaussian splatting [Kerbl et al. 2023] represents the scene

(a) Initial Poses   (b) Direct Primitive Sampling   (c) Joint Optimization   (d) Progressive Clustering and Merging
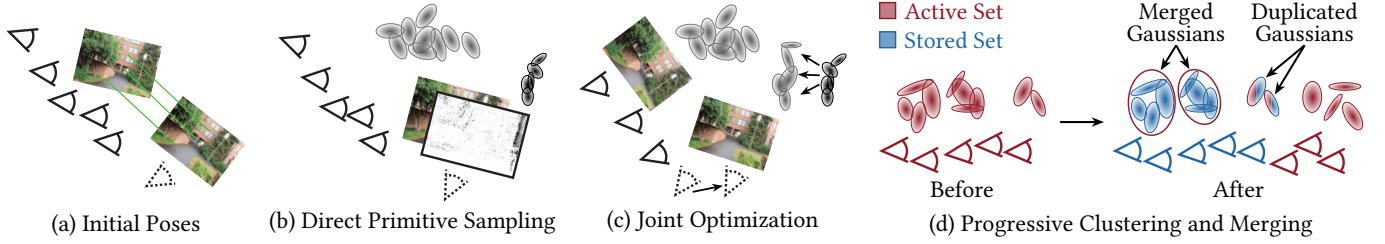
Fig. 2. Overview of our method. As each new image arrives, (a) we first use learning-based feature matching, reformulated in a GPU-friendly manner to achieve fast initial pose estimation (b) we perform direct sampling of Gaussian primitives by estimating the probability that a pixel should generate a Gaussian, and also sample the size, (c) the two first steps enable fast and effective joint optimization, improving both poses and scene representation and (d) we incrementally cluster into *anchors*, and merge nodes allowing us to represent large-scale scenes. When placing an anchor, the Gaussians from the active set are stored for later rendering. We create the new active set by merging primitives that appear small and duplicating the others, reducing the number of Gaussians being optimized.

with 3D Gaussian primitives that can be efficiently projected and rasterized on the GPU. Each primitive stores opacity and Spherical Harmonics (SH) to represent appearance; these are blended together to recreate the appearance of the scene with high fidelity. 3DGS achieves real-time rendering but requires careful initialization and densification, i.e., creating new primitives during optimization to achieve high-quality results. Recent solutions have improved densification and reduced primitive counts, thus speeding up the method with minimal quality loss [Huang et al. 2024; Mallick et al. 2024; Papantonakis et al. 2024]. Several methods propose explicit initialization [Fang and Wang 2024a], but still require additional densification; unpublished concurrent work [Fang and Wang 2024b] refines this idea to further accelerate computation. However, these methods are unsuited for on-the-fly incremental scene reconstruction mainly because they require all camera poses to be computed beforehand, and densification requires many optimization iterations, increasing the overall cost. We propose direct sampling of Gaussian primitives alleviating the need for densification, reducing 3DGS optimization cost, and incremental joint optimization compatible with large scenes.

Joint poses and 3D reconstruction. Jointly optimizing camera poses and 3D scene representations is a natural approach to eliminate dependence on known camera poses [Deng et al. 2024; Jeong et al. 2021; Lin et al. 2021; Liu et al. 2023; Meuleman et al. 2023; Sun et al. 2024b]. Such methods, however, require expensive optimization times—from seconds per frame to days per scene—and often rely on RGB-D data, e.g., [Deng et al. 2024; Sun et al. 2024b]. Near real-time performance has been achieved in recent work that jointly optimize poses and 3D Gaussians [Keetha et al. 2024; Peng et al. 2024; Sun et al. 2024a; Zhu et al. 2025]; Again, most of these require RGB-D input. A few methods handle RGB-only inputs [Li et al. 2024; Matsuki et al. 2024], but typically do not use matching-based pose initialization, increasing the number of required gradient-based optimization iterations, especially when the baseline is large. While some achieve real-time performance by reducing iteration counts [Peng et al. 2024; Sun et al. 2024a], they are suboptimal for datasets with large baselines (Sec. 5.1). Some methods also alternate between pose estimation and reconstruction, further increasing optimization time [Fu et al. 2024; Matsuki et al. 2024].

Some recent methods incorporate frame-to-frame matching to reduce optimization time, but can require hours for complete optimization [Jiang et al. 2024], or while improving speed with DUSt3R-based initialization, have quadratic matching time with the number of frames [Fan et al. 2024]. Photo-SLAM combines ORB-SLAM3 with 3DGS, achieving real-time performance, but given ORB-SLAM3's limitations has difficulty with wide baselines, and requires additional computation time to sufficiently densify the representation. In currently unpublished work, several methods propose SLAM solutions using 3DGS (e.g., [Chen et al. 2024; Feng et al. 2024; Homeyer et al. 2024; Zhang et al. 2024]) with strategies such as monocular depth priors, or focusing more on loop closure. None of these methods supports large-scale scenes, while maintaining both interactive pose estimation and acceptable visual quality.

In contrast to these solutions, our fast initial pose estimation and Gaussian sampling steps strikes a better balance of computational load, simultaneously handling SLAM-like and wide-baseline capture, as well as large-scale scenes.

Representations for large-scale scenes. Some NVS methods handle general captures [Barron et al. 2022; Sara Fridovich-Keil and Alex Yu et al. 2022; Zhang et al. 2020] or naturally extend to larger scenes [Kerbl et al. 2023], but have insufficient capacity for very large trajectories. Several large-scale representations subdivide scenes into local radiance fields [Duckworth et al. 2023; Meuleman et al. 2023; Mi and Xu 2023; Tancik et al. 2022; Turki et al. 2022; Xu et al. 2023], often requiring specific handling of seams [Duckworth et al. 2023; Tancik et al. 2022]. Similar solutions have been developed for 3DGS, requiring divide-and-conquer solutions [Lin et al. 2024] followed by involved steps to construct and adapt a hierarchy to the data [Kerbl et al. 2024; Liu et al. 2024a; Ren et al. 2024]. All of these training approaches are incompatible with incremental reconstruction, since they require prior knowledge of the full scene camera poses. Progressive optimization of poses and local radiance fields has been explored [Meuleman et al. 2023], but optimization is slow, requiring up to 40 hours for 1000 images. Pose estimation methods struggle with large datasets, as shown by COLMAP's [Schönberger and Frahm 2016] failure to produce consistent trajectories in certain public large-scale datasets [Kerbl et al. 2024; Meuleman et al.

2023]. Hierarchical 3D Gaussians [Kerbl et al. 2024] also need time-intensive, per-chunk bundle adjustment, taking up to hours per chunk.

Our efficient joint optimization overcomes all these limitations and is inherently suited to incremental scene construction, using a sliding window solution for large-scale scenes.

3 OVERVIEW

We propose an *on-the-fly* method to estimate camera poses and compute a complete radiance field at the speed of photo acquisition, designed for large scenes. Our method has four main components: 1) A fast – but approximate – initial pose estimation, adopting a careful design to allow GPU-friendly mini bundle adjustment; 2) A direct sampling method to find the position and shape of Gaussian primitives, by estimating the probability of each pixel to generate a Gaussian, significantly reducing the need for densification; 3) A method for joint optimization of poses and 3DGS that is very efficient, thanks to the first two steps, improving the initial versions of both poses and the radiance field; 4) An online scalable optimization using a sliding set of anchors that progressively clusters 3DGS primitives in space, allowing the treatment of large-scale scenes. Figure 2 provides an overview of these steps.

4 METHOD

4.1 Lightweight Initial Pose Estimation

We first compute approximate initial poses, that will later be improved by joint optimization; our design choices thus prefer speed over flexibility. Specifically, to fully exploit the GPU, we first use a limited number of keypoints, reducing expensive memory access on the GPU and second, we formulate our solution as a fixed-size problem thus exploiting GPU core parallelization. Initial pose estimation thus has three stages: feature extraction, bootstrapping and subsequent frame estimation.

Feature extraction. A fast feature keypoint detector and descriptor [Potje et al. 2024] is applied to each input image, generating 6144 keypoints per frame.

Bootstrapping. We first wait until the first N_{init} frames have arrived, then run exhaustive matching between each pair of these (N_{init} is 8 in our experiments). From this set of matches, we optimize the focal length, poses and 3D keypoints' positions by minimizing the reprojection error. Following standard practice, we implement this mini bundle adjustment as Levenberg-Marquardt optimization [Levenberg 1944; Madsen et al. 2004; Marquardt 1963]. Our mini bundle adjustment is lightweight and efficient, compared, e.g., to full 3DGS rendering SGD optimization used by other methods [Matsuki et al. 2024].

The key to our efficient solver is to carefully layout the problem so that each 3D point is seen from a fixed number of images. This results in a fixed-size sparse Jacobian J of the reconstruction error, which is easy to build and enables an efficient solve method on the GPU. Specifically, we compute the Jacobian of the reprojection error with respect to the camera pose J_{cam} and 3D point position J_{xyz}. At each iteration, similar to standard solvers, we compute the reprojection error and its Jacobian J, that is built from J_{cam} and J_{xyz},

which are both sparse. Since we fix the non-zero block sizes, we can pre-allocate memory and compute every block independently with fixed-size computation, allowing us to exploit batch-processing on the GPU. This simplified layout avoids the need for flexible solvers such as Ceres [Agarwal et al. 2023] that are typically used for bundle adjustment.

Pose Estimation for Subsequent Frames. In each new frame, we match its keypoints to those in the last N registered frames (N is 6 in our experiments). To establish 3D-2D correspondences, we estimate the 3D positions of keypoints in each of the N past frames using the known (previous) camera poses and triangulation. If this estimation fails, we use rendered depth. We then estimate the camera pose and inliers from these 3D-2D correspondences using GPU-parrallel RANSAC with our mini bundle adjustment as estimator. After this initialization, we run 20 iterations of the mini bundle adjustment with all inliers to refine the pose. Finally, a 3D Gaussian primitive is created for each triangulated keypoint. Although a keypoint can be seen from many images due to transitive matches, we restrict supervision to the last N registered frames to maintain a fixed-size problem.

To ensure the method recovers in challenging scenarios (pure rotations, scale drift), we rerun the bootstrapping when the mean distance between the last twenty cameras is below 0.1/3. If the projection error is below 1 pixel, we update the N_{init} last frames' poses by aligning it to the previously estimated ones.

4.2 Sampling Gaussian Primitives

To avoid the overhead and shortcomings of densification, we introduce a direct sampling method for Gaussian primitives. At each frame, we sample a dense set of 3D Gaussian primitives satisfying two requirements: 1) place primitives to cover previously unseen regions, or to add additional detail in coarsely reconstructed parts of the scene and 2) avoid placing more primitives than actually required in any given region. To satisfy these requirements simultaneously, we introduce a sampling method based on the probability that a pixel of a new frame should generate a primitive. The steps of our sampling method are illustrated in Fig. 5.

✕ Gaussian centres

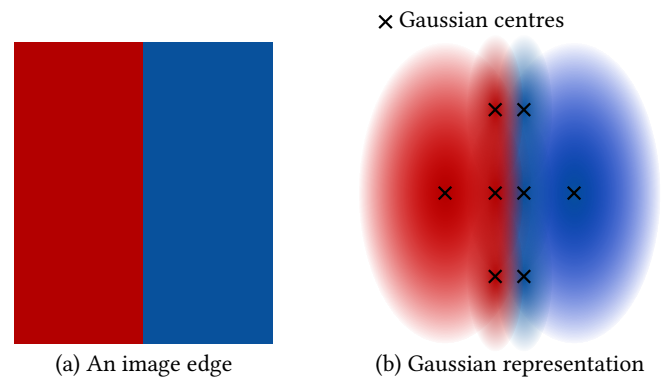(a) An image edge   (b) Gaussian representation

Fig. 3. To represent an image edge (a) accurately, more Gaussians should be placed on both sides of the discontinuity (b), placing them where they are required.

Existing Gaussian splatting SLAM approaches typically handle Gaussian initialization by either uniformly distributing Gaussians across the image [Sun et al. 2024a] or placing them at keypoints [Huang et al. 2024]. However, uniform placement fails to adapt to the specific features of the input image, while keypoints alone tend to be too sparse, requiring further densification with the resulting increase in optimization time and number of primitives.

We define the probability that a primitive should be generated at a given pixel based on two criteria: 1) 3D Gaussian primitives should be concentrated in areas with high-frequency details which cause discontinuities, and 2) Gaussians should be positioned on both sides of each discontinuity, to accurately represent edges (see Fig. 3).
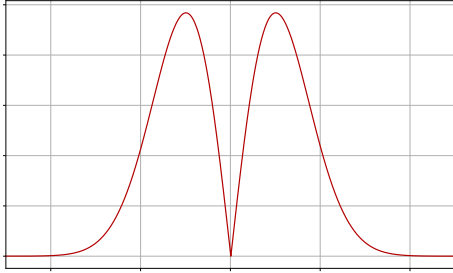

Fig. 4. The response of the norm of the Laplacian of Gaussian (LoG) function to a step function. The LoG operator highlights edges by producing two peaks on either side of a discontinuity.

The probability that a given pixel should generate a primitive is thus based on the local spatial gradient. Using the norm of the Laplacian of Gaussian (LoG) operator [Haralock and Shapiro 1991] as a proxy for our probability fulfills both criteria: areas with high-frequency details yield high LoG norm, and a sharp edge produces two peaks on either side of the discontinuity (see Fig. 4). Thus, we assign an initial probability P_L that a primitive should be generated at pixel (x, y) based on the LoG norm:

$$P_L(x, y) = \min \left(\left\| \nabla^2(n_\sigma) * I(x, y) \right\|, 1 \right). \quad (1)$$

Here, I is the input image, and n_σ is a Gaussian kernel with a standard deviation of σ.

To satisfy our second requirement, i.e., avoid placing excess Gaussians in areas where there are already sufficient primitives to represent the edges, we render a view $\tilde{I}$ from the viewpoint of the new frame. We then compute the same quantity as in Eq. 1 but for the rendered image $\tilde{I}$, providing a pixel-wise penalty $\tilde{P}$ that reduces the probability of placing new Gaussians in already reconstructed areas:

$$\tilde{P}(x, y) = \min \left(\left\| \nabla^2(n_\sigma) * \tilde{I}(x, y) \right\|, 1 \right). \quad (2)$$

The quantity $\tilde{P}$ will be similar to P_L in regions where content has already been reconstructed, thus the final probability for adding a Gaussian at pixel (x, y) is given by:

$$P_s(x, y) = \max \left(P_L(x, y) - \tilde{P}(x, y), 0 \right). \quad (3)$$

This will reduce the probability to spawn primitives in regions that are already well represented by the rendered image and thus the representation.

Depth for Gaussian Primitive Positions. With a set of pixel positions selected for conversion into 3D Gaussians, we now have a set of pixels that will spawn 3D Gaussian primitives; the next step is to estimate their depths. We use Depth-Anything-2 [Yang et al. 2024] to estimate monocular depth, which we align to the triangulated matches using the same procedure as described in Kerbl et al. [2024]. We then estimate depth using a standard correlation volume approach centered around the monocular depth. This guided matching is essential as the monocular depth can exhibit significant errors. Details of this computation are given in the Appendix.

Primitive Size Parameter. In 3DGS, the scale of the primitives is initialized based on the average distance to the approximate 3D 3-nearest neighbors. However, this approach tends to produce overly large Gaussians around discontinuities and is sensitive to outliers, resulting in an initialization that poorly matches the input frame (see Fig. 6).

To address this, we first estimate an appropriate scale in image space. Using the probability from Eq. (1), we compute the expected distance to the nearest neighbor assuming a local 2D Poisson process of intensity $P_L(x, y)$ around the pixel (x, y) [Clark and Evans 1954]:

$$s' = \frac{1}{2\sqrt{P_L(x, y)}}. \quad (4)$$

This calculation leverages the probability P_L before the penalty term, as a high penalty would imply that many Gaussians are already present.

Next, we convert from pixel space to 3D space using the camera's focal length f and the estimated depth z for the pixel: $s = \frac{zs'}{f}$. This approach provides an appropriate scale without requiring a nearest neighbor search, making it efficient. We then assign s to each dimension of the 3D Gaussian's scale vector S.

4.3 Joint Pose and Gaussian Optimization and Scheduling

For each new image, we now have an initial estimation of poses and directly sampled positions and sizes of Gaussian primitives. These two fast initial steps enable efficient joint optimization of the poses and the 3DGS representation.

For each new image received, we only register the frame if the median displacement of the keypoints exceeds 3% of the screen width; these registered frames are the *keyframes*. This ensures that only frames with meaningful parallax are used, improving geometry estimation and avoiding redundant frames.

For each registered image, we run 30 Gaussian splatting optimization iterations, using the fast backpropagation and sparse-Adam optimizer from [Mallick et al. 2024] to enhance iteration speed.

Learning rates are assigned per Gaussian, with decay rates adjusted based on the point at which each Gaussian was introduced. Camera poses are optimized jointly using a 6D rotation representation for rotations [Hempel et al. 2022]. These poses receive gradient updates from Gaussian position and rotation optimization, but not from spherical harmonics, as propagating gradients through view-dependent color information can degrade pose quality [Liu et al. 2023].

To capture low-frequency scene details first, accelerate optimization and avoid local minima, we employ a coarse-to-fine strategy
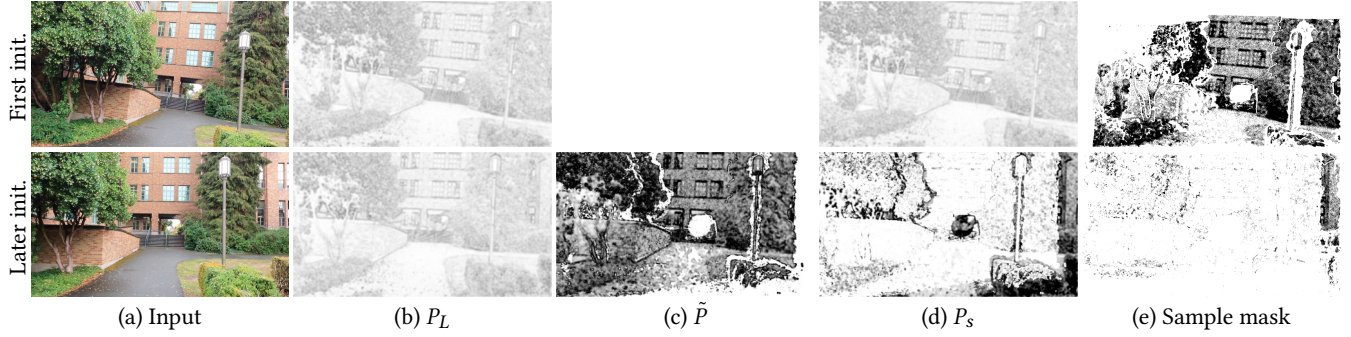
(a) Input	(b) P_L	(c) $\tilde{P}$	(d) P_s	(e) Sample mask

Fig. 5. Direct sampling to place new primitives during joint optimization: When adding a new frame (a), we compute an intitial probability map P_L based on the Laplacian of Gaussian (LoG) norm (Eq. (1)) (b). We then compute a penalty map $\tilde{P}$ based on the LoG norm of the rendered image (c); for the first image, this is empty, since there are no Gaussians to render. The final probability map P_s is computed by subtracting the penalty from the initial probability (Eq. (3), (d)). From this probability, we create the mask to place new Gaussians (e). We observe that this method places more Gaussians in high-frequency areas (leaves), while texture-less regions feature less samples (road). Also, while the first initialization places primitives over the full image, the penalty map helps to avoid placing Gaussians in already well-represented areas. For example, only the unseen right part of the later initialization has many new Gaussians.

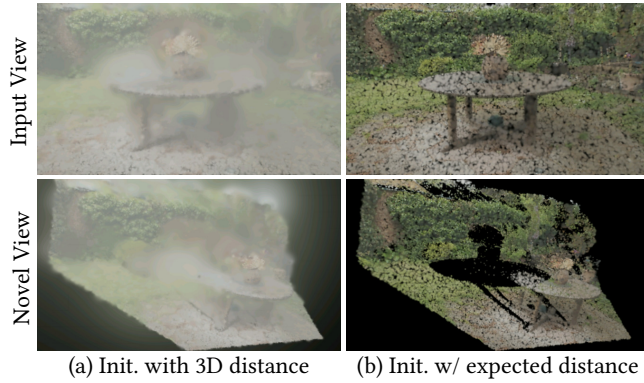|	
(a) Init. with 3D distance	(b) Init. w/ expected distance

Fig. 6. Scale initialization. (a) With the initial scale based on the 3D distance to the 3-nearest neighbors, some Gaussians are too large. (b) Scale initialization based on the expected distance to the nearest neighbor is more appropriate, fitting the input image better.

[Huang et al. 2024; Sun et al. 2024a]. Specifically, every time an image is added, it is used for training with 2^l downsampling ($l = 3$ in our experiments). Then, every five iterations, we decrement l until we reach full image size. We use proper filtering [Yu et al. 2024] to ensure correct multi-scale training. While we do not perform densification, we do apply opacity culling as in the original 3DGS, removing primitives with very low opacity.

Our initial pose estimation prevents local minima when optimizing poses and Gaussians jointly, allowing us to handle wider-baseline than SLAM-based methods. The incremental nature of our approach is well-suited to handling large environments, using our scalable incremental method we describe next.

4.4 Scalable Incremental Gaussian Construction

Given our incremental pose and radiance field optimization, our final goal is to allow processing of large-scale environments. Previous solutions for large environments [Kerbl et al. 2024; Liu et al. 2024a;

Ren et al. 2024] incur significant overhead of creating, optimizing and maintaining hierarchical data structures, often requiring many hours for large scenes. This overhead is unacceptable for our goal of on-the-fly radiance field construction.

As images are processed, we maintain a set of *Active Gaussian* containing primitives currently being optimized and rendered. After some time, primitives placed and optimized earlier may appear very small or even sub-pixel-sized from the current camera location, contributing little to the rendered images. These primitives are offloaded from GPU to CPU RAM and stored at an *anchor*. This gives a scene representation as a set of clusters that can be loaded back onto the GPU when required. The clustering process has three steps: 1) detecting when to create an anchor, 2) clustering and primitive merging 3) Incremental optimization with a sliding window.

Detecting when to create an anchor. We define the size $\mathcal{S}$ of a primitive from a camera i as S/D where D is the distance of the primitive center to the camera, and S is the scale of the Gaussian. When we are at camera i in the sequence, we check if more than 40% of the *Active Gaussians* have a size $\mathcal{S} < \tau_{\min}$ from camera $i-1$'s point of view ($\tau_{\min} = 1$ pixel). If they do, we trigger an update to create an anchor and merge these Gaussians.

Clustering and Primitive Merging. The set of *Active Gaussians* before the update is copied to the newly created anchor. The anchor stores a location, the set of Gaussian primitives, their optimization state and the keyframes they were optimized with.

We next merge Gaussians that have a minor contribution to obtain a coarser representation of distant regions. To do this, we randomly select $\frac{1}{k+1}$ th of primitives deemed too fine in the detection step. We then find the k nearest neighbors for each of the selected primitives using the method of Papantonakis et al. [2024] and merge them following the approach adopted by Kerbl et al [2024] ($k = 3$). All other primitives are kept unchanged.

Sliding window incremental optimization. The merging process leaves us with a coarser representation of the scene, which becomes

the new *Active Gaussian* set. This set is optimized in the next iteration. Subsequent clustering steps will create new anchors, and distant content will become progressively coarser due to merging. At the end of the capture path, the *Active Gaussian* set is stored in a last anchor. The use of anchors is illustrated in the supplemental video.

Once we have created the full scene representation of the scene, where different scale representations are stored with anchors, we can navigate freely in space. To perform rendering of novel views with our representation, we select the closest anchor to the camera's current position and render the Gaussians it contains. When two anchors have a similar distance to the camera, we blend the Gaussians from both. If the two closest anchors from the camera's point of view are at distances d_1 and d_2 (with $d_1 \leq d_2$), we define the overlap parameter as $o \in (0, 0.5)$ (e.g., $o = 0.1$) and compute a ratio $r = \frac{d_1}{d_2}$. If $r \leq 1 - o$, then the blending weight for the closest anchor is 1 and 0 for the other anchor. Otherwise, we linearly blend the weight:

$$w(r) = \begin{cases} 1, & \text{if } r < 1 - o, \\ 1 - \left(r - (1 - o)\right)\frac{0.5}{o}, & \text{otherwise.} \end{cases} \quad (5)$$

5 RESULTS AND EVALUATION

We implemented our method building on the 3DGS codebase [Kerbl et al. 2023], adding a python based interactive viewer for training and online visualization after optimization. Source code of our method and the viewer, as well as additional material are available at https://repo-sam.inria.fr/nerphys/on-the-fly-nvs/.

Our method is robust to different capture styles, ranging from SLAM-like dense video to the more wide-baseline captures typically used for NVS. To demonstrate this, we evaluate on datasets taken with a variety of different capture styles. For SLAM-like capture, we evaluate on the densely captured TUM dataset [Sturm et al. 2012], often used for evaluation of SLAM-based approaches. For intermediate, somewhat wider baseline and larger scale capture we evaluate on Static Hikes [Meuleman et al. 2023]. For NVS wide-baseline capture, we test on a selection of scenes from the MipNeRF360 dataset [Barron et al. 2022]. For large-scale scenes, we evaluate on the SmallCity* and Wayve* scenes, adapted from H3DGS [Kerbl et al. 2024] by using only the front camera. We have selected scenes from these datasets that have ordered image sequences, which is a requirement for our method. Finally, we have also captured a large dataset CityWalk with a Canon EOS R6 camera in drive mode at 3 images per second. The average number of images for the TUM datasets (fr1, fr2, fr3) is 2289, for MipNeRF360 (garden, counter, bonsai) 239 and for StaticHikes (forest1, forest2, university2) 972. For the H3DGS the average is 2285; our self-captured CityWalk scene has 4055 images, but has by far the largest spatial length of 1.1km. We use image resolution of 1200-1600 width in all tests, unless stated otherwise, that usually corresponds to half resolution of the raw input data.

We ran all tests and evaluations on a workstation with an Intel Core i9 14900K CPU, 128GB of RAM, and an NVIDIA RTX 4090 GPU, or if we use a different configuration (e.g. when the method requires more GPU memory) we scale the timings to this setup by

running 1000 calls to our CUDA rasterizer on each machine. We use the same set of parameters for all scenes.

5.1 Methodology

We present comparisons to two sets of methods. First we compare to state-of-the-art methods that do not require camera poses as input. These are mainly SLAM/3DGS and pose-free 3DGS solutions. We selected methods for comparison based on code availability, reported performance, and the ability to handle as many scenes types as possible. For pose-free approaches that create a 3DGS scene, we compare to Photo-SLAM [Huang et al. 2024], DROID-Splat [Homeyer et al. 2024] and MonoGS [Matsuki et al. 2024], and finally CF-3DGS [Fu et al. 2024], all presented in 2024.

We also present two baselines. First, standard 3DGS (i.e., the release from the official github repo, using standard COLMAP parameters) for 7K and 30K iterations; the time reported is the *total* of all the COLMAP processing and the 3DGS optimization. The second baseline uses Taming 3DGS [Mallick et al. 2024] which is the fastest current 3DGS optimization, coupled with a best effort approach to accelerate SfM pose estimation using GLOMAP [Pan et al. 2024]. Specifically we run the COLMAP feature extractor, sequential matcher and then the GLOMAP mapper to get poses and SfM points. This latter baseline can be considered the current fastest best-practice, non-incremental solution to pose estimation and 3DGS optimization. Since the TUM dataset features denser capture, we run methods without keyframe selection (Taming 3DGS, 3DGS, and COLMAP Free 3DGS) on fewer images. Specifically, we select every 3rd, 15th, and 10th frame for fr1, fr2, and fr3, respectively. This approach aligns the total number of frames more closely with our number of registered keyframes while retaining all images from the test set. Note that the reported time for our method includes automatic keyframe selection.

DROID-Splat and CF-3DGS cannot handle full resolution images. As a result, we present a separate table (Tab. 2), with these methods at a resolution where they work both as well as possible for each dataset (446x336 for TUM and 640 width for MipNeRF360 and StaticHikes). Our method requires higher resolution inputs, as XFeat [Potje et al. 2024] performs optimally within the 1 to 2 megapixel range. For comparisons, we upsample the resized images by a factor of two in both dimensions before processing them with our method. Metrics are then reported on the appropriately downsampled images. Another issue is the specification of the set of test images for evaluation. Different approaches are used for different methods, since in some cases not all images have an estimated pose, and thus the test set is often different for each method, even for the same scene. We defined a single evaluation protocol, using every n^{th} image as a test view, where n is 8 and 10 for MipNeRF360 and StaticHikes, as proposed by their authors, and 30 for TUM, as the baseline between frames is small. This required specific modifications for each method (please see the Appendix).

The second comparison is for methods treating large-scale scenes that cannot be handled by standard 3DGS. Specifically, we compare to H3DGS [Kerbl et al. 2024]. For this comparison, we use the front camera from the SmallCity and Wayve datasets as well as our CityWalk dataset. We provide the COLMAP calibration using

Table 1. Reconstruction time and novel view quality results for different methods. The first section compares our method with others that use unposed images, while the second section employs Structure from Motion (SfM). Reported runtimes include pose optimization. COLMAP intrinsics are used for both Photo-SLAM and MonoGS. The best and second best are color coded for pose-free methods.

	TUM				MipNeRF360				StaticHikes			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
Photo-SLAM	19.30	0.700	0.382	0:02:12	16.54	0.505	0.603	0:02:11	14.13	0.316	0.660	0:02:01
MonoGS	16.60	0.682	0.381	0:16:18	14.46	0.436	0.663	0:04:05	15.46	0.301	0.659	0:09:19
Ours	23.02	0.821	0.250	0:00:50	24.31	0.775	0.300	0:01:02	20.40	0.589	0.365	0:01:30
GLOMAP + Taming 3DGS (7k)	25.29	0.868	0.191	0:03:33	27.52	0.866	0.226	0:08:50	20.23	0.537	0.465	1:02:34
COLMAP + 3DGS (7k)	24.75	0.874	0.175	0:04:11	27.82	0.877	0.212	0:09:52	20.40	0.538	0.463	2:33:57
COLMAP + 3DGS (30k)	25.34	0.881	0.157	0:09:44	29.66	0.906	0.170	0:26:05	24.08	0.757	0.267	2:47:11

Table 2. Novel view quality results for different methods that require low-resolution input. We use COLMAP intrinsics for COLMAP Free 3DGS.

	TUM				MipNeRF360				StaticHikes			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
DROID-Splat	19.49	0.721	0.325	0:06:35	25.87	0.776	0.258	0:11:18	19.65	0.470	0.506	0:09:26
CF-3DGS	15.05	0.578	0.405	1:10:27	13.52	0.295	0.621	1:08:14	15.21	0.301	0.560	7:52:54
Ours	22.45	0.815	0.225	0:01:11	25.80	0.834	0.182	0:00:55	21.93	0.673	0.270	0:01:32

the method from H3DGS for all these scenes, since it is one of the few ways to get camera poses with SfM for scenes of this size. We also evaluate pose estimation quality for the test views, comparing to ground truth poses when these exist (i.e., TUM) and using the COLMAP poses as "pseudo-ground truth" following standard practice and using the RMSE APE and RPE metrics [Grupp 2017].

5.2 Novel View Synthesis Quality

In Tab. 1, 2, we show the average results for each method broken down by dataset type for the SLAM set of methods. For each method we show the standard metrics PSNR, SSIM and LPIPS as well as the average time for full processing, i.e., the total time for pose estimation and 3DGS optimization. The average times for GLOMAP pose optimization are 0:02:02, 0:07:17, and 1:00:26 for TUM, MipNeRF360, and StaticHikes, respectively. For TUM, we use a subset of the images for CF-3DGS and the SfM methods, as they do not feature keyframing, whereas other methods process all images. Additionally, SfM-based approaches require the entire dataset before processing, as their mappers reorder the images. This prevents live feedback and obtaining the reconstruction immediately at the end of the capture.

We also show qualitative results visually comparing the different methods in Figures 7 and 8. We see that the visual quality of our solution is on par or better than all competitors, for all types of scenes. DROID-Splat has good visual quality; our method tends to be sharper, but can have slightly lower fidelity. SLAM methods perform well on the dense captures they are designed for, but visual quality can degrade or the method can even fail as the camera baseline becomes wider. Taming 3DGS and standard 3DGS have better visual quality, and work well for all scenes, but with the high computational overhead discussed previously, making them unsuitable for our on-the-fly reconstruction scenario.

After the training has processed all views, we can fine-tune Gaussians and cameras using the identified keyframes. To do this, we load anchors one-by-one. For each anchor, the associated cameras and

Gaussians parameters are optimized further by randomly sampling all of the anchor's keyframes. Since only 3DGS optimization is performed, this process is fast enough, and we can repeat it for multiple *epochs* to find an ideal overhead/quality tradeoff. Tab. 3 shows that we reach Taming 3DGS (7k)'s quality. However, achieving quality beyond this requires a more involved solution; we discuss this as future work Sec. 6.

We next show results for the large-scale scenes, where we compare to H3DGS. In Tab. 4 we see that the overhead of camera calibration using SfM approaches grows significantly with the scale of the scene. Capturing the scene CityWalk took 30min which is more than the 25min our method requires to process the scene; using H3DGS (one of the few methods that can handle captures this big) requires 22 hours of processing after capture is finished. In addition, the quality of the pose estimation is very low, leading to failure for novel view synthesis in several segments of the path.

5.3 Pose Estimation Quality

We evaluate pose estimation quality using the APE and RPE metrics in Tab. 5. Our method performs well for the MipNeRF360 dataset but has difficulty with TUM. This is due to the low quality of this video capture, where many frames are blurry, and there is significant rolling shutter which we do not explicitly handle, resulting in bad outlier poses. SLAM methods are often tuned to perform well on this dataset. We also compare to Spann3r, a transformer-based approach, whose poses are of lower quality than our method due to substantial pose drift.

5.4 Runtimes

Tab. 6 details the runtime of each step of the algorithm on the Garden dataset. Each step is executed for every keyframe. Feature detection and extraction are performed for every input frame to determine whether it should be retained as a keyframe. We process the input

Table 3. Results with additional optimization. Our method achieves quality similar to Taming 3DGS (7k).

| # epochs	TUM				MipNeRF360				StaticHikes			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
10	24.38	0.843	0.224	0:00:58	25.88	0.815	0.265	0:01:20	21.06	0.620	0.338	0:02:00
25	25.09	0.853	0.210	0:01:09	26.51	0.830	0.247	0:01:47	21.26	0.635	0.322	0:02:42
50	25.63	0.862	0.198	0:01:28	27.08	0.842	0.233	0:02:34	21.43	0.649	0.308	0:03:50
100	26.08	0.866	0.189	0:02:05	27.17	0.848	0.224	0:04:09	21.57	0.660	0.296	0:06:08
Taming 3DGS (7k)	25.29	0.868	0.191	0:03:33	27.52	0.866	0.226	0:08:50	20.23	0.537	0.465	1:02:34
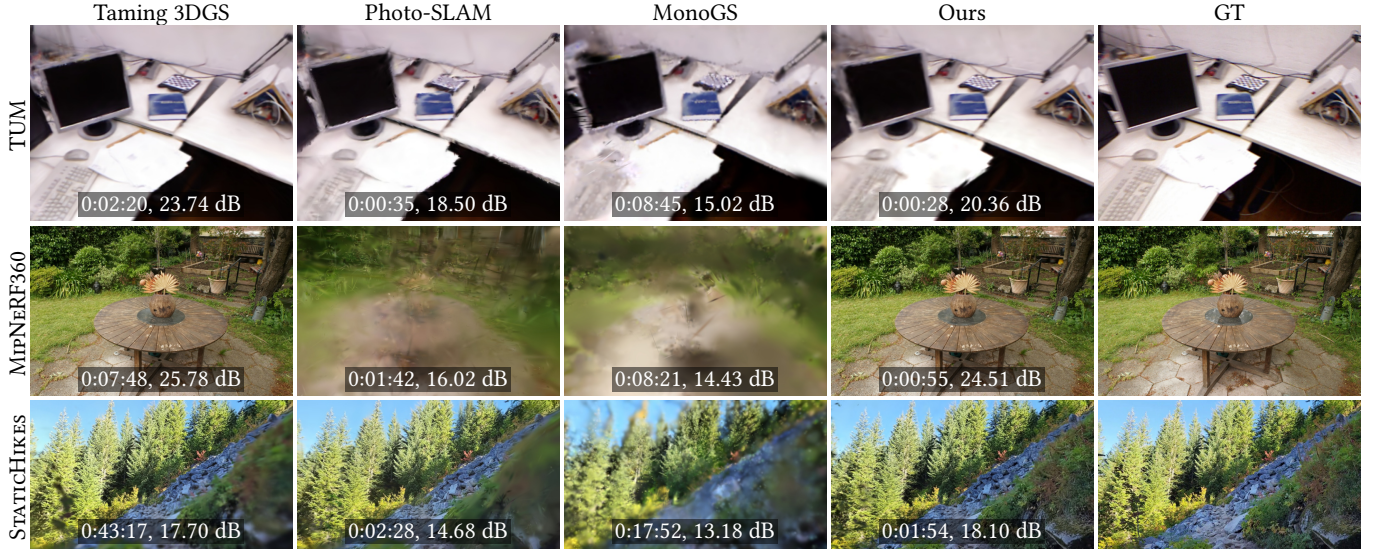
Fig. 7. Qualitative comparison for the three datasets used, for Taming 3DGS, Photo-Slam, MonoGS. We include the images of these methods for the test views in Fig. 8 in supplemental. We show the scene reconstruction time and PSNR. Note that Taming 3DGS requires significantly more time since it uses offline SfM for camera pose estimation. Our approach provides better visual quality for these scenes compared to other methods that take unposed images as input.

Table 4. Results for large scale scenes. For other methods time includes total time for COLMAP using the H3DGS process and the actual 3DGS optimization of each method.

	SmallCity*				Wayve*				CityWalk			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
H3DGS	21.17	0.679	0.285	2:55:28	20.80	0.737	0.227	7:29:45	11.78	0.557	0.560	22:09:20
Ours	23.59	0.789	0.323	0:01:45	20.29	0.739	0.303	0:04:29	21.71	0.712	0.395	00:25:03

Table 5. Pose estimation results for different methods using absolute and relative error metrics. The best and second best are color coded.

	TUM				MipNeRF360			
	T.APE↓	R.APE↓	T.RPE↓	R.RPE↓	T.APE↓	R.APE↓	T.RPE↓	R.RPE↓
DROID-Splat	1.0	0.033	1.2	0.016	11.7	0.052	19.1	0.074
Photo-SLAM	9.0	0.034	8.9	0.021	314.0	2.016	318.9	1.213
MonoGS	33.5	0.197	23.3	0.063	315.5	2.373	278.3	0.983
CF-3DGS	73.1	2.817	15.0	0.187	161.5	2.777	59.5	0.197
Spann3r	89.4	0.507	33.4	0.210	32.9	0.130	42.2	0.150
Ours	40.2	0.313	5.7	0.045	11.4	0.035	16.6	0.047

images at 40, 4, and 9 FPS and retain 9%, 86%, and 31% of them as keyframes for TUM, MipNeRF360, and StaticHikes, respectively.

Enabling CUDA graphs has a significant impact on the runtime, bringing the bootstrapping optimization time from 625 to 145 and the incremental mini bundle adjustment time from 95 to 4 milliseconds. The performance discrepancy shows that setting a fixed-sized problem for mini bundle adjustment is essential to obtain real-time performance in our implementation.

5.5 Ablations

We perform ablation studies to illustrate the importance of each step; results are shown in Tab. 8. First we replace our direct sampling with a uniform 0.5 probability of a pixel spawning a Gaussian (NoSampling). Second, we keep position sampling, but disable shape sampling (NoShape), using the standard 3DGS kNN-based scale initialization. Third, we use the monocular depth directly instead of the guided matching (NoGuided). Finally, we do not refine the
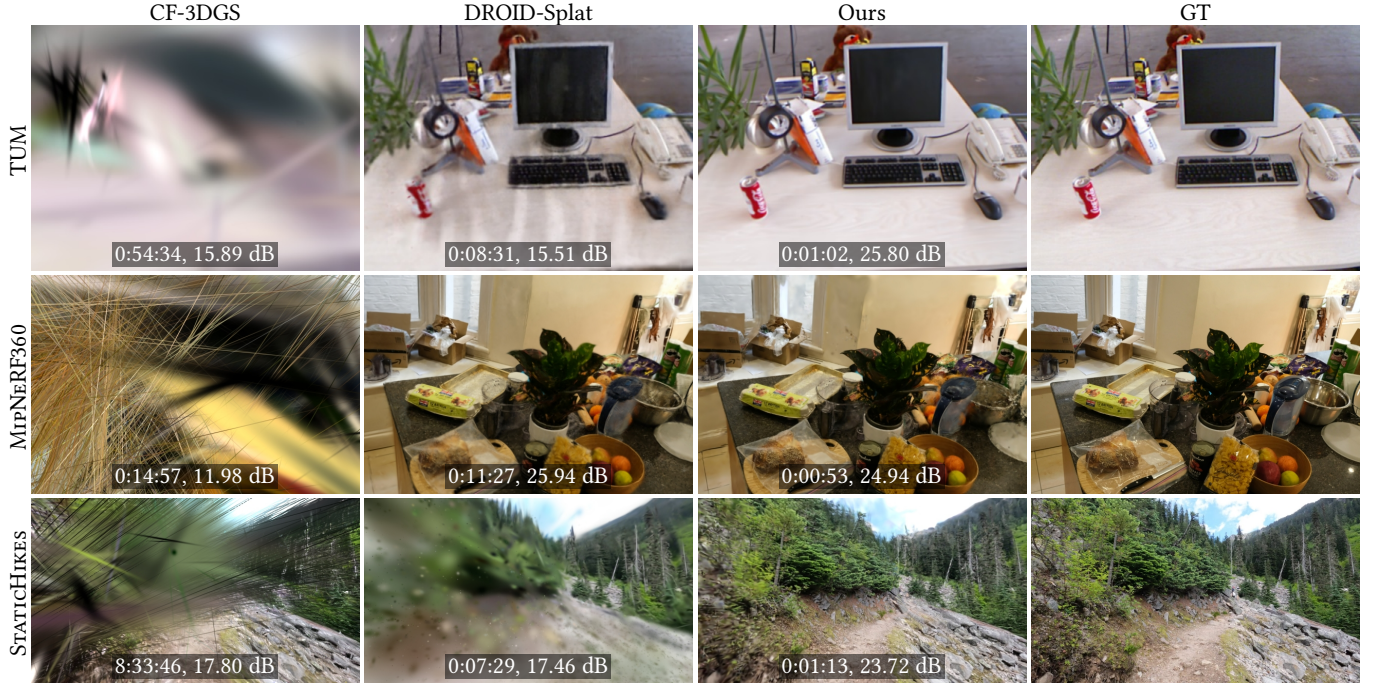
Fig. 8. Qualitative comparison of pose-free methods for the three datasets used, for CF-3DGS and DROID-Splat that only handle low resolution. We include the images of these methods for the test views in Fig. 7 in supplemental. We show the scene reconstruction time and PSNR. We see that our method is competitve to previous approaches, and is robust to different capture styles.

Fig. 9. Qualitative comparison of large-scale methods for the three datasets used. We show the scene reconstruction time and PSNR.

Table 6. Per keyframe runtime breakdown. The pose initialization (Section 4.1) and primitive sampling and placement (Section 4.2) are relatively efficient and joint pose and Gaussian optimization (Section 4.3) is the longest step.

Step	Time (ms)
Feature detection and extraction	3.4
Matching + Outlier removal	8.0
MiniBA incremental	4.0
Sum pose initialization	**15.4**
Monocular depth estimation	12.9
Triangulation and depth map alignment	23.2
Dense feature extraction	2.3
Probability estimation	0.6
Guided matching	11.2
Sum GS sampling	**50.2**
Joint optimization	**223.8**
Sum per keyframe	**289.4**

poses (NoJoint), i.e., we do not perform joint optimization and just use the initial pose estimation.

We perform the ablations on two scenes: garden from MipNeRF360 and Forest2 from StaticHikes.

The results show that the each component contributes to the visual quality of our solution. The expectancy-based scale is the most important factor for the MipNeRF360 garden scene. Our probability-based sampling contributes over 1dB in PSNR to overall quality for these cases, since it adapts to the input image or the current

Table 7. Impact of the anchors on Forest2. Enabling the anchors enhances quality and reduces the maximum number of Gaussians that need to be loaded during optimization.

	PSNR↑	SSIM↑	LPIPS↓	peak # GS↓
Without Anchors	21.56	0.594	0.386	1432671
With Anchors	22.22	0.616	0.370	976688

Table 8. Ablation results. NoSampling shows results using uniform sampling instead of our direct sampling, NoShape without our expectancy-based shape parameters for the primitives, NoGuided with monocular depth instead of guided matching, NoJoint without pose refinement when optimizing Gaussians.

Ablation	PSNR↑	SSIM↑	LPIPS↓
NoSampling	21.75	0.564	0.390
NoShape	19.12	0.422	0.541
NoGuided	21.91	0.561	0.388
NoJoint	21.61	0.559	0.379
OursFull	23.01	0.649	0.328

reconstruction state. In addition, using uniform sampling instead of our Laplacian-based approach is inefficient, bringing the number of primitives from 1.0M to 2.0M on Garden. Figures 10 and 11 show qualitative results for the ablations.

Table 7 shows the impact of the anchors on a medium-sized scene. They have a small positive impact on quality while significantly reducing the peak number of Gaussians rendered. For larger scenes, they are required to perform optimization within reasonable GPU memory and runtime: they allow the GPU memory usage to stabilize at 22GB after 150 images in CityWalk. This shows that there is (theoretically) no technical limit to the number of images that can be processed. Note that, for smaller scenes, the anchor creation criterion is never met, hence not impacting the results.

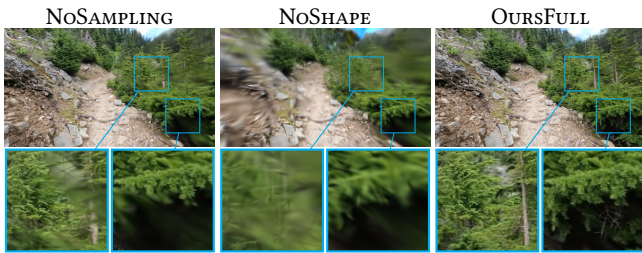NoSampling   NoShape   OursFull

Fig. 10. Qualitative evaluation of the various components via ablation studies on Forest2.

6 DISCUSSION

Our on-the-fly method for radiance field reconstruction provides immediate feedback that is central in ensuring user-friendly 3D reconstruction. Our method will greatly simplify and accelerate 3D capture from photos. We next discuss limitations and future work.

Our method currently relies on ordered image sequences. This is often the case in radiance field captures, but the assumption does not always hold. For example, several scenes in the 360 dataset [Barron et al. 2022] are unordered. An interesting future avenue of research
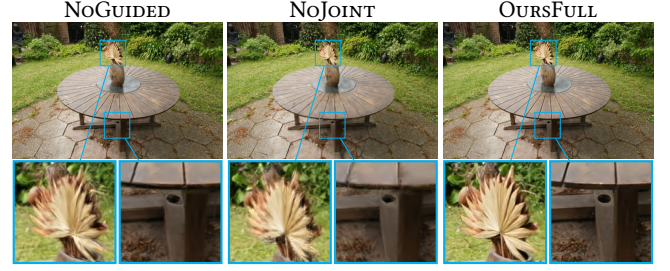
NoGuided   NoJoint   OursFull

Fig. 11. Qualitative evaluation of the various components via ablation studies on Garden.

is the addition of loop closure that would solve this issue and contribute to an even more robust solution. Such a solution would allow the user to directly identify regions that are not well reconstructed, and immediately take more pictures. This removes one of the most problematic aspects of 3D capture today, where returning to a capture site to take additional photos is always time-consuming and sometimes logistically impossible. The challenge is how to achieve this and maintain performance. We also require at least 1000 pixel width resolution for the method to work; given todays cameras, we believe this is not a significant limitation.

To achieve state-of-the-art visual quality, additional optimization is required. While existing optimization schedules can improve quality a bit (Sec. 5.2), they are not designed to work with the 3DGS representation we create incrementally with direct sampling. A new approach is required, that takes into account the fact that primitives are already quite dense and well-placed, but is able to escape local minima. This is an exciting direction of future research.

Like many other radiance field methods, we do not explicitly handle casual capture artifacts such as blur, saturation, lens flare or moving objects or people in the scene. This is visible in the blurry results we obtain for example in the TUM scenes, see Fig. 7, 8. A plethora of methods have been proposed to address some of these in 3DGS (e.g., [Sabour et al. 2024; Wenbo and Ligang 2024]); an interesting avenue of future work is to see how such solutions can be adapted to our framework. The main challenge is incorporating these additional solutions while maintaining performance.

Our method is suitable for a phone/camera application that can take photos and transmit them to a workstation for immediate reconstruction and feedback. Such an application will broaden the impact of our approach.

7 CONCLUSION

We have introduced a novel approach for on-the-fly large-scale pose estimation and 3D reconstruction from casually captured ordered photos, enabling high-quality novel view synthesis with immediate feedback. Our method can handle a wide variety of capture styles, varying from dense SLAM-style video, typical wider radiance field captures all the way to large sequential captures of thousands of images over more than 1km distance. We have shown that our approach is competitive with the best solutions that specialize in a specific capture style and/or scene size, and is one of the few method that works for all of them.

Our versatile method is an important step towards real-time 3D capture, with many potential applications in a wide variety of domains. Since our method reduces the computational overhead for pose estimation and optimization from 3DGS allowing immediate feedback, it can only increase the already widespread adoption of radiance fields.

ACKNOWLEDGMENTS

REFERENCES

Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. 2023. Ceres Solver. https://github.com/ceres-solver/ceres-solver

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *ICCV* (2021).

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR* (2022).

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV* (2023).

Eric Brachmann, Jamie Wynn, Shuai Chen, Tommaso Cavallari, Áron Monszpart, Daniyar Turmukhambetov, and Victor Adrian Prisacariu. 2024. Scene Coordinate Reconstruction: Posing of Image Collections via Incremental Learning of a Relocalizer. In *ECCV*.

Carlos Campos, Richard Elvira, Juan J. Gòmez, José M. M. Montiel, and Juan D. Tardós. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Transactions on Robotics* (2021).

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *ECCV*.

Guikun Chen and Wenguan Wang. 2024. A Survey on 3D Gaussian Splatting. arXiv:2401.03890 [cs.CV] https://arxiv.org/abs/2401.03890

Yu Chen, Rolandos Alexandros Potamias, Evangelos Ververas, Jifei Song, Jiankang Deng, and Gim Hee Lee. 2024. ZeroGS: Training 3D Gaussian Splatting from Unposed Images. arXiv:2411.15779 [cs.CV] https://arxiv.org/abs/2411.15779

Philip J. Clark and Francis C. Evans. 1954. Distance to nearest neighbor as a measure of spatial relationships in populations. *Ecology* 35 (1954).

J Czarnowski, T Laidlow, R Clark, and AJ Davison. 2020. DeepFactors: Real-time probabilistic dense monocular SLAM. *IEEE Robotics and Automation Letters* (2020).

Tianchen Deng, Yaohui Chen, Leyan Zhang, Jianfei Yang, Shenghai Yuan, Jiuming Liu, Danwei Wang, Hesheng Wang, and Weidong Chen. 2024. Compact 3D Gaussian Splatting For Dense Visual SLAM. arXiv:2403.11247 [cs.CV]

Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T. Barron. 2023. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. *ACM Transactions on Graphics* (2023).

Bardienus Duisterhof, Lojze Zust, Philippe Weinzaepfel, Vincent Leroy, Yohann Cabon, and Jerome Revaud. 2024. MASt3R-SfM: a Fully-Integrated Solution for Unconstrained Structure-from-Motion. arXiv:2409.19152 [cs.CV] https://arxiv.org/abs/2409.19152

Zhiwen Fan, Wenyan Cong, Kairun Wen, Kevin Wang, Jian Zhang, Xinghao Ding, Danfei Xu, Boris Ivanovic, Marco Pavone, Georgios Pavlakos, Zhangyang Wang, and Yue Wang. 2024. InstantSplat: Unbounded Sparse-view Pose-free Gaussian Splatting in 40 Seconds. arXiv:2403.20309 [cs.CV]

Guangchi Fang and Bing Wang. 2024a. Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXXVII* (Milan, Italy). Springer-Verlag, Berlin, Heidelberg, 165–181. doi:10.1007/978-3-031-72980-5_10

Guangchi Fang and Bing Wang. 2024b. Mini-Splatting2: Building 360 Scenes within Minutes via Aggressive Gaussian Densification. *arXiv preprint arXiv:2411.12788* (2024).

Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 2024. 3d gaussian splatting as new era: A survey. *IEEE Transactions on Visualization and Computer Graphics* (2024).

Dapeng Feng, Zhiqiang Chen, Yizhen Yin, Shipeng Zhong, Yuhua Qi, and Hongbo Chen. 2024. CaRtGS: Computational Alignment for Real-Time Gaussian Splatting SLAM. arXiv:2410.00486 [cs.CV] https://arxiv.org/abs/2410.00486

Yang Fu, Sifei Liu, Amey Kulkarni, Jan Kautz, Alexei A. Efros, and Xiaolong Wang. 2024. COLMAP-Free 3D Gaussian Splatting. In *CVPR*.

Michael Grupp. 2017. evo: Python package for the evaluation of odometry and SLAM. https://github.com/MichaelGrupp/evo.

Robert M Haralock and Linda G Shapiro. 1991. *Computer and robot vision*. Addison-Wesley Longman Publishing Co., Inc.

Thorsten Hempel, Ahmed A. Abdelrahman, and Ayoub Al-Hamadi. 2022. 6d Rotation Representation For Unconstrained Head Pose Estimation. In *IEEE International Conference on Image Processing (ICIP)*.

Christian Homeyer, Leon Begiristain, and Christoph Schnörr. 2024. DROID-Splat: Combining end-to-end SLAM with 3D Gaussian Splatting. *arXiv e-prints* (2024), arXiv–2411.

Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. 2023. Tri-MipRF: Tri-Mip Representation for Efficient Anti-Aliasing Neural Radiance Fields. In *ICCV*.

Huajian Huang, Longwei Li, Cheng Hui, and Sai-Kit Yeung. 2024. Photo-SLAM: Real-time Simultaneous Localization and Photorealistic Mapping for Monocular, Stereo, and RGB-D Cameras. In *CVPR*.

Yoonwoo Jeong, Seokjun Ahn, Christopher Choy, Animashree Anandkumar, Minsu Cho, and Jaesik Park. 2021. Self-Calibrating Neural Radiance Fields. In *ICCV*.

Kaiwen Jiang, Yang Fu, Mukund Varma T, Yash Belhe, Xiaolong Wang, Hao Su, and Ravi Ramamoorthi. 2024. A Construct-Optimize Approach to Sparse View Synthesis without Camera Pose. *ACM SIGGRAPH Conference Proceedings* (2024).

Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. 2024. SplaTAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM. In *CVPR*.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* (2023).

Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. 2024. A Hierarchical 3D Gaussian Representation for Real-Time Rendering of Very Large Datasets. *ACM Transactions on Graphics* (2024).

Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. 2021. Robust consistent video depth estimation.

Kenneth Levenberg. 1944. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quart. Appl. Math.* (1944).

Hao Li, Yuanyuan Gao, Chenming Wu, Dingwen Zhang, Yalun Dai, Chen Zhao, Haocheng Feng, Errui Ding, Jingdong Wang, and Junwei Han. 2024. GGRt: Towards Generalizable 3D Gaussians without Pose Priors in Real-Time. In *ECCV*.

Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. 2021. BARF: Bundle-Adjusting Neural Radiance Fields. In *ICCV*.

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. 2024. VastGaussian: Vast 3D Gaussians for Large Scene Reconstruction. In *CVPR*.

Junchen Liu, Wenbo Hu, Zhuo Yang, Jianteng Chen, Guoliang Wang, Xiaoxue Chen, Yantong Cai, Huan-ang Gao, and Hao Zhao. 2024b. Rip-NeRF: Anti-aliasing Radiance Fields with Ripmap-Encoded Platonic Solids. In *SIGGRAPH Conference Proceedings*.

Yang Liu, He Guan, Chuanchen Luo, Lue Fan, Junran Peng, and Zhaoxiang Zhang. 2024a. CityGaussian: Real-time High-quality Large-Scale Scene Rendering with Gaussians. In *ECCV*.

Yu-Lun Liu, Chen Gao, Andreas Meuleman, Hung-Yu Tseng, Ayush Saraf, Changil Kim, Yung-Yu Chuang, Johannes Kopf, and Jia-Bin Huang. 2023. Robust Dynamic Radiance Fields. In *CVPR*.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Transactions on Graphics* (2019).

Zeyu Ma, Zachary Teed, and Jia Deng. 2022. Multiview Stereo with Cascaded Epipolar RAFT. In *ECCV*.

Kaj Madsen, Hans Nielsen, and O Tingleff. 2004. Methods for Non-Linear Least Squares Problems (2nd ed.). (2004).

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. 2024. Taming 3DGS: High-Quality Radiance Fields with Limited Resources. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. doi:10.1145/3680528.3687694

Donald W. Marquardt. 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Indust. Appl. Math.* (1963).

Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. 2024. Gaussian Splatting SLAM. (2024).

Andreas Meuleman, Hyeonjoong Jang, Daniel S. Jeon, and Min H. Kim. 2021. Real-Time Sphere Sweeping Stereo from Multiview Fisheye Images. In *CVPR*.

Andreas Meuleman, Yu-Lun Liu, Chen Gao, Jia-Bin Huang, Changil Kim, Min H. Kim, and Johannes Kopf. 2023. Progressively Optimized Local Radiance Fields for Robust View Synthesis. In *CVPR*.

Zhenxing Mi and Dan Xu. 2023. Switch-NeRF: Learning Scene Decomposition with Mixture of Experts for Large-scale Neural Radiance Fields. In *ICLR*.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics* (2022).

Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. 2015. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics* (2015).

Linfei Pan, Dániel Baráth, Marc Pollefeys, and Johannes Lutz Schönberger. 2024. Global Structure-from-Motion Revisited. In *ECCV*.

Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. 2024. Reducing the Memory Footprint of 3D Gaussian Splatting. In *Proceedings of the ACM on Computer Graphics and Interactive Techniques*.

Zhexi Peng, Tianjia Shao, Liu Yong, Jingke Zhou, Yin Yang, Jingdong Wang, and Kun Zhou. 2024. RTG-SLAM: Real-time 3D Reconstruction at Scale using Gaussian Splatting. (2024).

Guilherme Potje, Felipe Cadar, Andre Araujo, Renato Martins, and Erickson R. Nascimento. 2024. XFeat: Accelerated Features for Lightweight Image Matching. In *CVPR*.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. 2024. Octree-GS: Towards Consistent Real-time Rendering with LOD-Structured 3D Gaussians. arXiv:2403.17898 [cs.CV] https://arxiv.org/abs/2403.17898

Sara Sabour, Lily Goli, George Kopanas, Mark Matthews, Dmitry Lagun, Leonidas Guibas, Alec Jacobson, David J. Fleet, and Andrea Tagliasacchi. 2024. SpotLessSplats: Ignoring Distractors in 3D Gaussian Splatting. *arXiv:2406.20055* (2024).

Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*.

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *CVPR*.

Thomas Schops, Torsten Sattler, and Marc Pollefeys. 2019. BAD SLAM: Bundle Adjusted Direct RGB-D SLAM. In *CVPR*.

Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. 2012. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Shuo Sun, Malcolm Mielle, Achim J. Lilienthal, and Martin Magnusson. 2024a. GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting. In *CVPR*.

Shuo Sun, Malcolm Mielle, Achim J. Lilienthal, and Martin Magnusson. 2024b. High-Fidelity SLAM Using Gaussian Splatting with Rendering-Guided Densification and Regularized Optimization. In *IROS*.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. 2022. Block-NeRF: Scalable Large Scene Neural View Synthesis. *arXiv* (2022).

Zachary Teed and Jia Deng. 2020. DeepV2D: Video to Depth with Differentiable Structure from Motion. In *ICLR*.

Fabio Tosi, Youmin Zhang, Ziren Gong, Erik Sandström, Stefano Mattoccia, Martin R Oswald, and Matteo Poggi. 2024. How nerfs and 3d gaussian splatting are reshaping slam: a survey. *arXiv preprint arXiv:2402.13255* 4 (2024).

Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. 2022. Mega-NERF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs. In *CVPR*.

Hengyi Wang and Lourdes Agapito. 2024. 3D Reconstruction with Spatial Memory. *arXiv preprint arXiv:2408.16061* (2024).

Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. 2024. DUSt3R: Geometric 3D Vision Made Easy. In *CVPR*.

Chen Wenbo and Liu Ligang. 2024. Deblur-GS: 3D Gaussian Splatting from Camera Motion Blurred Images. *Proc. ACM Comput. Graph. Interact. Tech. (Proceedings of I3D 2024)* (2024).

Linning Xu, Yuanbo Xiangli, Sida Peng, Xingang Pan, Nanxuan Zhao, Christian Theobalt, Bo Dai, and Dahua Lin. 2023. Grid-guided Neural Radiance Fields for Large Urban Scenes. In *CVPR*.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024. Depth Anything V2. In *Advances in Neural Information Processing Systems*.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024. Mip-Splatting: Alias-free 3D Gaussian Splatting. In *CVPR*.

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. arXiv:2010.07492 [cs.CV]

Wei Zhang, Qing Cheng, David Skuddis, Niclas Zeller, Daniel Cremers, and Norbert Haala. 2024. HI-SLAM2: Geometry-Aware Gaussian SLAM for Fast Monocular Scene Reconstruction. arXiv:2411.17982 [cs.RO] https://arxiv.org/abs/2411.17982

Wang Zhao, Shaohui Liu, Hengkai Guo, Wenping Wang, and Yong-Jin Liu. 2022. ParticleSfM: Exploiting Dense Point Trajectories for Localizing Moving Cameras in the Wild.

Liyuan Zhu, Yue Li, Erik Sandström, Shengyu Huang, Konrad Schindler, and Iro Armeni. 2025. LoopSplat: Loop Closure by Registering 3D Gaussian Splats.

A APPENDIX: IMPLEMENTATION DETAILS

We present various implementation details of the different steps of our method.

A.1 Initial Pose Estimation

We describe here more details of the first step in our pipleine.

Feature extraction. For faster feature extraction and matching, we run the feature extractor model with half precision and CUDA graphs.

Bootstrapping. We initialize the focal length as 0.7 times the image width, poses as identity and the 3D points with depth 1.

We then run 200 iterations of Levenberg-Marquardt optimization with initial $\lambda = 1 \cdot 10^{-5}$.

We use initial damping $\lambda_{\text{init}} = 10^{-5}$ with factor $\nu = 2$ applied at each iteration such that $\lambda_{i+1} = \lambda_i/\nu$ if $\|r_i + 1\| < \|r_i\|$, $\lambda_{i+1} = \nu\lambda_i$ otherwise.

We employ the Huber loss and discard residuals whose errors exceed the sum of the median and four times the median absolute deviation, ensuring robustness to potential outliers.

The fixed-size layout also allows us to use CUDA graphs, so all iterations run with a single CPU call, further accelerating computation.

A.2 Depth for Gaussian Primitives

We construct a correlation volume with respect to neighboring frames by adapting the dense feature extractor from [Ma et al. 2022] and applying it to each frame, using half precision and CUDA graphs, similar to the feature detector, to optimize performance. This produces a per-pixel feature map $\mathcal{F}$ for each frame.

Next, we select the most suitable neighboring frame for each pixel position, following the adaptive matching approach in [Meuleman et al. 2021]. This adaptive selection ensures that only one neighboring frame per pixel is used, maximizing depth disambiguation while remaining efficient.

We then construct a candidate depth vector, $z_0, \ldots, z_{N-1}$. We restrict the search to a region around the estimated monocular depth: we uniformly sample in inverse depth over the range:

$$\left[\frac{1}{Z^*} - 10^{-1}, \frac{1}{Z^*} + 10^{-1} \right]. \tag{6}$$

For each candidate z_k, we reproject the pixel (x, y) from the current frame i to the selected neighboring frame j, producing coordinates $\left(x_k^{i \to j}, y_k^{i \to j} \right)$.

This allows us to build a correlation vector as:

$$C_k = \left\langle \mathcal{F}_i(x, y), \mathcal{F}_j \left(x_k^{i \to j}, y_k^{i \to j} \right) \right\rangle \tag{7}$$

where the inner product measures the similarity between feature vectors. We then determine the optimal depth candidate via quadratic fitting.

A.3 Joint Optimization

During joint optimization, we initialize all learning rates to a fixed value when the Gaussians are introduced, then when we run the sparse Adam optimizer, we multiply by the decay ratio for each Gaussian processed.

A.4 Evaluation Methodology Details

Using a test/train split. We aim to optimize poses for all test views without using them to optimize the 3DGS representation. For our method, we enforce keyframe addition for the test images. Photo-SLAM uses ORB-SLAM3 as its SLAM backend. We modify ORB-SLAM3's keyframing logic to ensure test frames are registered. If the local mapper performing bundle adjustment is busy, the frame is added to the queue. This approach is successful for TUM and StaticHikes, but ORB-SLAM3 registers only about two-thirds of the test images for MipNeRF360. We evaluate the method based on the registered subset. MonoGS registers all frames in the sequence, providing poses for all test frames regardless of keyframe selection. For DROID-Splat, we update the keyframing criteria to always include test frames. In our experiments, DROID-Splat successfully tracked and registered all test frames. Additionally, all methods were modified to ensure that test frames were not used to optimize the scene; specifically, we skip the Gaussian optimizer for test frames.

Photo-SLAM runtime. The time taken by Photo-SLAM's optimization is determined by the frames' timestamps. For TUM, we use the provided timings. For StaticHikes and MipNeRF360, we assume intervals of 0.1s and 0.5s between images, respectively, to maintain a similar runtime order of magnitude as ours.

Focal length. We use COLMAP's focal length for Photo-SLAM, MonoGS, and COLMAP Free 3DGS, as these methods cannot estimate intrinsics.