IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

VINGS-Mono: Visual-Inertial Gaussian Splatting Monocular SLAM in Large Scenes

Ke Wu[1], Zicheng Zhang[2], Muer Tie[1], Ziqing Ai[1], Zhongxue Gan[1], Wenchao Ding[1,*]

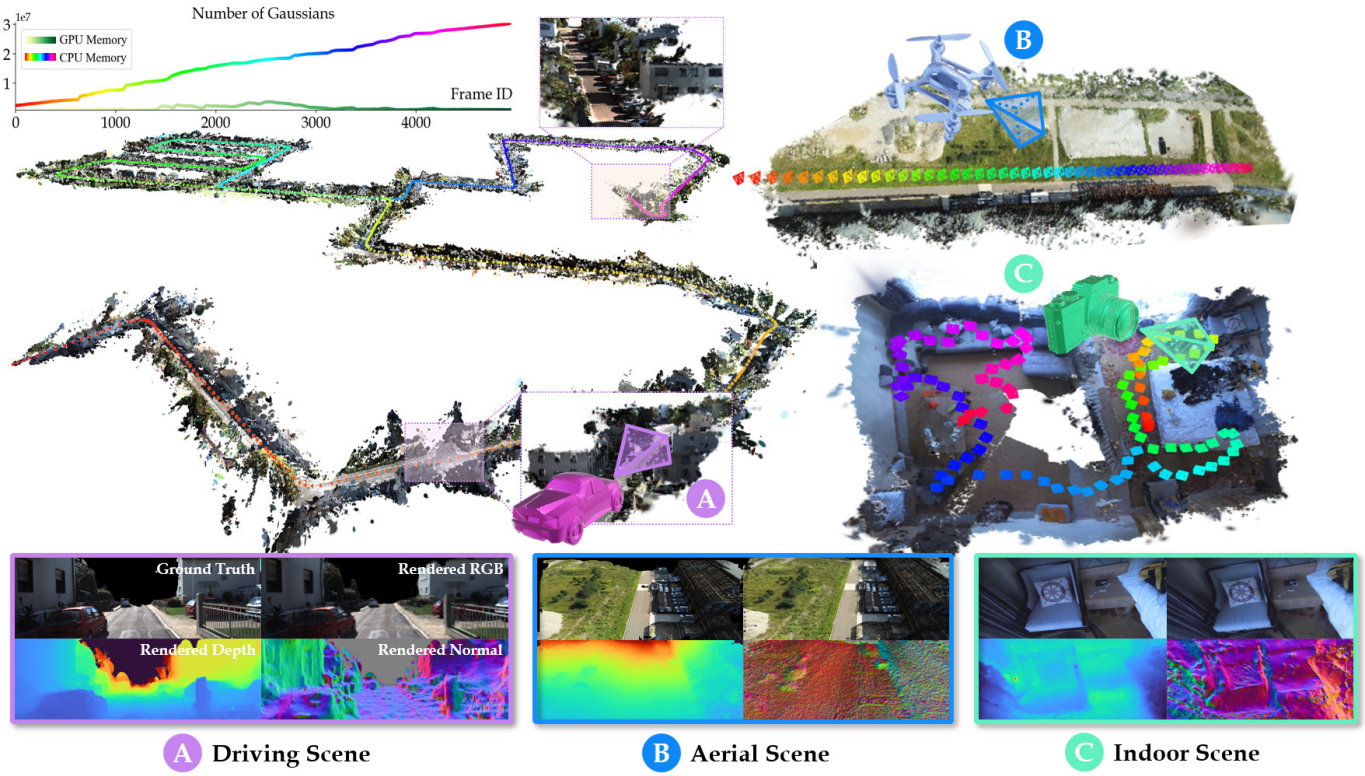


Fig. 1: VINGS-Mono's estimated trajectory and reconstructed gaussian map of three different scenes. Our method effectively estimates poses and reconstructs high-quality Gaussian maps across large-scale driving scenarios, aerial drone views, and indoor environments. Particularly for the **driving scene** on the left, the trajectory spans 3.7 kilometers and includes a Gaussian map containing 32.5 million Gaussian ellipsoids. During training, we track the number of Gaussians and zoom in on specific areas to improve visualization clarity. (Project page with code available: https://vings-mono.github.io)

***Abstract*—VINGS-Mono is a monocular (inertial) Gaussian Splatting (GS) SLAM framework designed for large scenes. The framework comprises four main components: VIO Front End, 2D Gaussian Map, NVS Loop Closure, and Dynamic Eraser. In the VIO Front End, RGB frames are processed through dense bundle adjustment and uncertainty estimation to extract scene geometry and poses. Based on this output, the mapping module incrementally constructs and maintains a 2D Gaussian map. Key components of the 2D Gaussian Map include a Sample-based Rasterizer, Score Manager, and Pose Refinement, which collectively improve mapping speed and localization accuracy. This enables the SLAM system to handle large-scale urban environments with up to 50 million Gaussian ellipsoids. To ensure global consistency in large-scale scenes, we design a Loop Closure module, which innovatively leverages the Novel View Synthesis (NVS) capabilities of Gaussian Splatting for loop closure detection and correction of the Gaussian map. Additionally, we propose a Dynamic Eraser to address the inevitable presence of dynamic objects in real-world outdoor scenes. Extensive evaluations in indoor and outdoor environments demonstrate that our approach achieves localization performance on par with Visual-Inertial Odometry while surpassing recent GS/NeRF SLAM methods. It also significantly outperforms all existing methods in terms of mapping and rendering quality. Furthermore, we developed a mobile app and verified that our**

Manuscript received: January, 10, 2025; Revised: April, 18, 2025; Accepted: August, 21, 2025.

This paper was recommended for publication by Editor Javier Civera upon evaluation of the Reviewers' comments. This work is sponsored by Shanghai Municipal Science and Technology Major Projectunder Grant(2021SHZDZX0103), National Natural Science Foundation of China (NSFC) under Grant 62403142, and the Science and Technology Commission of Shanghai Municipality (No. 24511103100). *(*Corresponding Author: Wenchao Ding)*

[1]Ke Wu, Muer Tie, Ziqing Ai, Zhongxue Gan and Wenchao Ding are with the College of Intelligent Robotics and Advanced Manufacturing, Fudan University, Shanghai, China. (e-mail: dingwenchao@fudan.edu.cn)

[2]Zicheng Zhang is with the College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

framework can generate high-quality Gaussian maps in real time using only a smartphone camera and a low-frequency IMU sensor. To the best of our knowledge, VINGS-Mono is the first monocular Gaussian SLAM method capable of operating in outdoor environments and supporting kilometer-scale large scenes.

***Index Terms*—SLAM, Gaussian Splatting, Sensor Fusion**

I. Introduction

An information-rich, geometrically dense map is essential for a robot's environmental perception and scene understanding. 3D Gaussian Splatting (3DGS) [1] has rapidly gained popularity due to its exceptional rendering speed and high-quality visuals. 3DGS enhances SLAM systems by providing detailed scene information and enabling novel view synthesis. Furthermore, due to Gaussian Splatting's differentiable rendering process, we can construct the dense maps using only low-cost RGB supervision.

Existing 3DGS SLAM systems [2], [3] primarily focus on a limited number of displayed objects or small indoor spaces, using depth cameras as input and leveraging traditional SLAM front end or depth point cloud ICP for localization and Gaussian updates. Outdoor GS-SLAM methods [4] are scarce and restricted to reconstructing scenes within a few hundred meters, relying heavily on high-beam LiDAR sensors. However, depth cameras perform poorly in outdoor settings, and the high cost of LiDAR sensors has limited their adoption in consumer applications. Given constraints in size, weight, and power, a low-cost camera paired with an IMU forms the minimal sensor suite for SLAM implementation. Therefore, developing a robust monocular (inertial) GS-SLAM system capable of handling large-scale environments is both essential and urgent.

Currently, monocular input-supported 3DGS SLAM systems [5], [6], which initialize Gaussians using random or sparse feature points, are unable to handle large-scale, fast-moving scenes due to their vulnerability to pose drift and geometry noise. Furthermore, significant accumulated errors are commonly observed in large-scale environments. These errors are typically mitigated through loop closure. Traditional loop closure methods, relying on descriptors or network feature vectors, require additional encoding and storage of bag-of-words models, which is inefficient and leads to performance degradation as the scene scale increases. GO-SLAM [7], on the other hand, identifies loop closures by maintaining the co-visibility matrix between frames, but this results in quadratic storage demands and increased computational overhead.

Developing an efficient and high-fidelity monocular GS-SLAM for large-scale scenes faces several significant challenges. First, representing large, street-level scenes requires managing tens of millions of Gaussians, which is both storage-intensive and computationally demanding. Second, monocular setups suffer from severe scale drift, which undermines the accuracy and reliability of the reconstructed scenes. Furthermore, significant cumulative errors arise in large-scale environments. While traditional loop closure techniques are effective at optimizing landmark-based maps, correcting a dense Gaussian map after detecting a loop closure is highly challenging and often requires retraining on all historical frames. Lastly, the presence of dynamic objects in large urban environments poses significant challenges, as they generate considerable artifacts and noise in the Gaussian map, further complicating the optimization process.

In this paper, we introduce VINGS-Mono, a monocular (inertial) Gaussian Splatting SLAM framework that supports large-scale urban scenes. The framework consists of four main modules: VIO Front End, 2D Gaussian Map, NVS Loop Closure, and Dynamic Object Eraser. To address challenges in Gaussian map storage and optimization efficiency, we develop a score manager to manage the 2D Gaussian Map by integrating local and global map representations. Additionally, we design a sample rasterizer to accelerate the backpropagation algorithm of Gaussian Splatting, significantly improving its computational efficiency. To enhance tracking accuracy and mitigate the inevitable drift encountered in large-scale scenarios, we propose a single-to-multi pose refinement module. This module back-propagates rendering errors from a single frame to optimize the poses of all frames within the frustum's field of view, improving overall pose consistency. For accumulated errors, we utilize the novel view synthesis (NVS) capability of Gaussian Splatting for loop closure detection. We further propose an efficient loop correction method capable of simultaneously adjusting millions of Gaussian attributes upon detecting a loop. Finally, to address the impact of dynamic objects on mapping, we design a heuristic semantic segmentation mask generation method based on re-rendering loss. This method ensures that dynamic objects are effectively handled, enhancing the robustness of the mapping process.

Our contributions can be summarized as follows:

- We are the first monocular (inertial) GS-based SLAM system capable of operating in outdoors and support kilometer-scale urban scenes.
- We propose a 2D Gaussian Map module, including a sample rasterizer, score manager, and single-to-multi pose refinement, ensuring that our method could achieve accurate localization and build high-quality gaussian maps in real time.
- We introduce a GS-based loop detection method, along with an efficient approach that can correct tens of millions of Gaussian attributes in a single operation upon loop detection, effectively eliminating accumulated errors and ensuring the global consistency of the map.
- Comprehensive experiments on different scenes (indoor environments, aerial drone view and driving scenes) demonstrate that VINGS-Mono outperforms existing approaches in both rendering and localization performance. Furthermore, we developed a mobile app and carried out real-world experiments to demonstrate the practical reliability of our method.

II. Related Works

In this section, we review related works in Gaussian Splatting SLAM, Visual Loop Closure, and Large-Scale Visual SLAM, as they are highly relevant to our framework. These topics cover the core aspects of our method: scene representation with Gaussian splatting, ensuring global consistency

through loop closure, and addressing scale drift and memory consumption in large-scale environments.

A. NeRF and Gaussian Splatting SLAM

Recent research has actively explored integrating NeRF [8] or Gaussian Splatting [1] into SLAM systems [9]. Early works like iMAP [10] pioneered this for RGB-D sensors using a single MLP but faced scalability limitations. Subsequently, NICE-SLAM [11] improved scalability for larger RGB-D scenes by introducing hierarchical feature grids. Addressing the limitations of fixed-resolution grids, Point-SLAM [12] proposed anchoring features in adaptively dense neural point clouds instead, allowing density to match scene detail. Concurrently, others focused on enhancing robustness and scalability for very large environments. NGEL-SLAM [13] integrated robust external tracking (ORB-SLAM3 [14]) and managed multiple implicit submaps with loop closure capabilities, while PLGSLAM [15] introduced a progressive representation, dynamically creating new local tri-plane-based maps coupled with local-to-global bundle adjustment to mitigate long-term drift. Transitioning to RGB-only input, methods like NICER-SLAM [16] enabled end-to-end optimization by incorporating monocular geometric cues.

3D Gaussian Splatting, with its differentiable nature and fast rendering speed, has emerged as a promising scene representation in SLAM systems. Compared with traditional explicit map representation such as voxel grids [17], surfels [18], point-clouds [14], [19], GS representation provides a denser map and novel view synthesis capabilities. The initial GS-SLAM method SplaTAM [2] focused on utilizing depth cameras to initialize Gaussian ellipsoids and optimized camera poses by backpropagating photometric errors from rendering loss to Gaussian positions and then to camera poses. However, this approach was highly sensitive to depth camera noise and demonstrated limited robustness. PhotoSLAM [6] and MonoGS [5] extended GS-SLAM to monocular settings for small indoor scenes, adding Gaussian ellipsoids through ORB feature points or random initialization and directly estimating poses with ORB-SLAM3 [14]. Despite their effectiveness in small-scale environments, these methods showed significant limitations in handling large or dynamic scenarios, causing severe floaters that greatly compromised map quality in the SLAM system. Gaussian-SLAM [20] introduced differential depth rendering and frame-to-model alignment, which enhanced its overall performance. However, it exhibited inefficiencies in frame processing speed and memory usage. GS-ICP-SLAM [21], on the other hand, achieved notable improvements in frame processing rates by employing point cloud matching. More recently, MGS [22] was proposed for monocular settings, leveraging a Multi-View Stereo Network from DPVO [23] as a depth prior. While promising, this method faces challenges in scaling to large-scale scenes, limiting its applicability in extensive environments. LIVGaussMap [4] introduced GS-Mapping to outdoor scenes at a scale of hundreds of meters. However, it relies on high-beam LiDAR, which is typically difficult to obtain for consumer-grade applications.

Although GS-SLAM has developed rapidly, the visual GS-SLAM systems mentioned above primarily focus on small-scale indoor scenes using datasets like TUM-RGBD [24] and Replica [25]. This significantly limits the applicability of GS-SLAM in larger-scale scenarios. To address this, we designed and developed a robust and efficient monocular (inertial) GS-SLAM method, which was tailored to the challenges of large-scale environments.

B. Visual Loop Closure

Visual Loop Closure consists of two main components, Loop Detection and Loop Correction. In large-scale environments, loop closure is essential, especially in monocular settings that lack scale information, where each segment of the trajectory has a different scale. Detecting and correcting loops enables visual SLAM systems to effectively eliminate cumulative errors and build globally consistent maps.

In terms of loop detection, early methods used hand-crafted features [26]–[28] to capture the general appearance of an image through single vectors or histograms. However, these global features lacked robustness to rotation and scale changes. The advent of local descriptors [29]–[31] enhanced feature robustness, employing visual bag of words (BoW) [32] or vocabulary trees [33] for efficient descriptor management across frames. Despite their effectiveness, hand-crafted features struggled in dynamic conditions with variable lighting or seasonal changes. The shift to deep learning introduced adaptive feature extraction, significantly boosting the reliability of loop closure detection. Pioneering this approach, Chen et al. [34] utilized features from all layers of trained networks for enhanced location recognition. Methods like NetVLAD [35] then improved image descriptor resilience by integrating multiple features, while LoopSplat [36] and hloc [37] further refined this approach. Additionally, advancements in visual foundation models have led to techniques like SALAD [38] using DINOv2 [39], significantly enhancing descriptor quality for loop detection in complex environments.

In terms of loop correction, ORB-SLAM3 [14] performs a map merging operation after detecting a loop. This process mainly consists of four steps, welding window assembly, merging maps, welding BA, and essential-graph optimization. These steps collectively optimize both the poses and the landmark map. VINS-Mono [19] applies an extra pose graph optimization step to guarantee that the past poses are arranged in a globally consistent manner. However, for dense maps (e.g., NeRF [8], 3DGS [1]), correcting the map after detecting a loop is challenging, as these maps are typically generated through training based on given poses and image pairs. Retraining them would be extremely time-consuming. GO-SLAM [7] attempts to correct poses and inverse depths by iteratively optimizing loop edges added to existing local keyframes and optimizing the dense map through training. However, this approach struggles to perform loop correction for large-scale drift in extensive scenes. LoopySLAM [40] and LoopSplat [36] address scene map correction by constructing submaps, but this method also fails to resolve scale drift within the submap, making it difficult to adapt to monocular settings.

To address loop detection and correction in large-scale monocular settings and to explore the potential of 3DGS in

place recognition, our approach innovatively utilizes the novel view synthesis capabilities of Gaussian Splatting, allowing us to perform loop detection using only the gaussian map. Moreover, we provide an efficient method that can correct millions of Gaussians in one go upon detecting a loop, enabling us to construct globally consistent Gaussian maps.

C. Large-Scale SLAM

In this subsection, we focus on visual-based SLAM methods tailored for large-scale environments. It is worth emphasizing that in extensive outdoor, street-level scenarios, visual SLAM is especially susceptible to scale drift and cumulative errors, posing significant challenges.

LSD-SLAM [41] is a pioneer in large-scale SLAM, building globally consistent maps by directly optimizing geometry on image intensities and explicitly modeling scale drift. VINS-Mono [19] and ORB-SLAM3 [14] incorporate IMU data to obtain weak scale information, ensuring localization accuracy. SelectiveVIO [42] and iSLAM [43] further extend this approach by utilizing neural networks to fuse visual and inertial sensor readings. However, these methods primarily focus on localization, resulting in very sparse map reconstructions. NEWTON [44] successfully integrates NeRF into SLAM tasks for large-scale indoor environments (e.g., at the scale of a single floor) by constructing view-centric submaps. However, this method struggles to adapt to outdoor scenarios with fast ego-motion. LIVGaussMap [4] and MMGaussian [45], by using LiDAR point clouds for initialization, extend 3D Gaussian Representation (3DGS) to outdoor SLAM tasks. Nonetheless, these methods rely heavily on LiDAR and are limited to scene scales in the range of hundreds of meters due to the large number of Gaussians involved.

We incorporate dense visual factors and IMU factors into the factor graph for optimization and design a score manager for the 2D Gaussian Map, which includes status control, storage control, and GPU-CPU transfer. This enables the reconstruction of tens of millions of Gaussian ellipsoids across kilometer-scale scenes.

III. System Overview

The pipeline of our framework is illustrated in Fig. 2. Given a sequence of RGB images and IMU readings, we first utilize the Visual Inertial Front End (Sec. IV) to select keyframes and calculate the initial depth and pose information of the keyframes through dense bundle adjustment. Additionally, we compute the depth map uncertainty based on the covariance from the depth estimation process, filtering out geometrically inaccurate regions and sky areas. The 2D Gaussian Map module (Sec. V) incrementally adds and maintains Gaussian ellipsoids using the outputs of the visual front end. We designed a management mechanism based on contribution scores and error scores to effectively prune Gaussians. Furthermore, we propose a novel method to optimize multi-frame poses using single-frame rendering loss. To ensure scalability to large-scale urban scenes, we implemented a CPU-GPU memory transfer mechanism. In the NVS Loop Closure Module (Sec. VI), we leverage the novel view synthesis capability of GS to design an innovative loop closure detection method and correct the Gaussian map through Gaussian-pose pair matching. Additionally, we integrate a Dynamic Object Eraser module (Sec. VII) that masks out transient objects like vehicles and pedestrians, ensuring consistent and accurate mapping under static scene assumptions.

IV. Visual Inertial Front End

The input of our Visual Inertial Front End consists of RGB images $\{I_t\}$ and IMU's acceleration and gyroscope (optional). We extract relevant features of adjacent RGB frames through the correlation volume from RAFT [46] and feed them into the DBA module proposed in DROID-SLAM [47] to estimate inverse depths and poses (Sec. IV-A). To enable fusion of visual data with IMU information, we use a graph optimization approach [48], [49] (Sec. IV-B). To prevent floaters in the map, we calculate the depth map uncertainty based on the information matrix [50] (Sec. IV-C).

A. Dense BA & Vision Factor

The visual constraints are modeled as a Dense Bundle Adjustment (DBA) optimization problem over inverse depth and pose, where the inverse depth $\mathbf{d}_i^{-1}$ is projected (Π_C^{-1}) through pose $\mathbf{T}_{ij}$ onto frame j to estimate the optical flow $\mathbf{u}_{ij}$:

$$\mathbf{u}_{ij} = \Pi_C(\mathbf{T}_{ij} \circ \Pi_C^{-1}(\mathbf{u}_i, \mathbf{d}_i^{-1})). \tag{1}$$

For adjacent RGB frames, we construct a correlation volume using their encoded feature maps. By applying a lookup operator on $\mathbf{u}_{ij}$, we obtain a correlation feature map. Using a GRU-based structure, we take the current image encoding, the correlation feature map, and the GRU's own hidden state as inputs, which then outputs the optical flow residual r_{ij} and the weight $\mathbf{w}_{ij}$ for subsequent upsampling. To ensure real-time processing, we downsample by a factor of eight, So the resolution of the inverse depth $\mathbf{d}_i^{-1}$ is one-eighth of the original RGB image.

The GRU outputs a revision flow field $\mathbf{u}_{ij}$, and we denote the corrected correspondence as $\mathbf{u}_{ij}^* = \mathbf{r}_{ij} + \mathbf{u}_{ij}$. We can then define this DBA problem as follows, iteratively optimizing $\mathbf{d}^{-1}$ and $\mathbf{T}_i, \mathbf{T}_j$:

$$\mathbf{E}(\mathbf{T}, \mathbf{d}^{-1}) = \sum_{(i,j)\in\epsilon} ||\mathbf{u}_{ij}^* - \Pi_C(\mathbf{T}_{ij} \circ \Pi_C^{-1}(\mathbf{u}_i, \mathbf{d}_i^{-1}))||^2_{\Sigma_{ij}}. \tag{2}$$

where Σ_{ij} represents the diagonal matrix of $\mathbf{w}_{ij}$.

Considering a bundle of edges anchored on frame i and projected to N co-visible frames, the combined Hessian is constructed by positionally stacking and summing the blocks as in Eq. 3:

$$\begin{bmatrix} \Sigma\mathbf{v}_{ii} \\ \mathbf{v}_{i1} \\ \vdots \\ \mathbf{v}_{iN} \\ \Sigma\mathbf{z}_{ii} \end{bmatrix} = \begin{bmatrix} \Sigma\mathbf{B}_{ii} & \mathbf{B}_{i1} & \cdots & \mathbf{B}_{iN} & \Sigma\mathbf{E}_{ii} \\ \mathbf{B}_{i1}^\top & \mathbf{B}_{11} & & & \mathbf{E}_{i1} \\ \vdots & & \ddots & & \vdots \\ \mathbf{B}_{iN}^\top & & & \mathbf{B}_{NN} & \mathbf{E}_{iN} \\ \Sigma\mathbf{E}_{ii}^\top & \mathbf{E}_{i1}^\top & \cdots & \mathbf{E}_{iN}^T & \Sigma\mathbf{C}_{ii} \end{bmatrix} \begin{bmatrix} \Delta\xi_i \\ \Delta\xi_1 \\ \vdots \\ \Delta\xi_N \\ \Delta\mathbf{d}_i^{-1} \end{bmatrix} \tag{3}$$

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

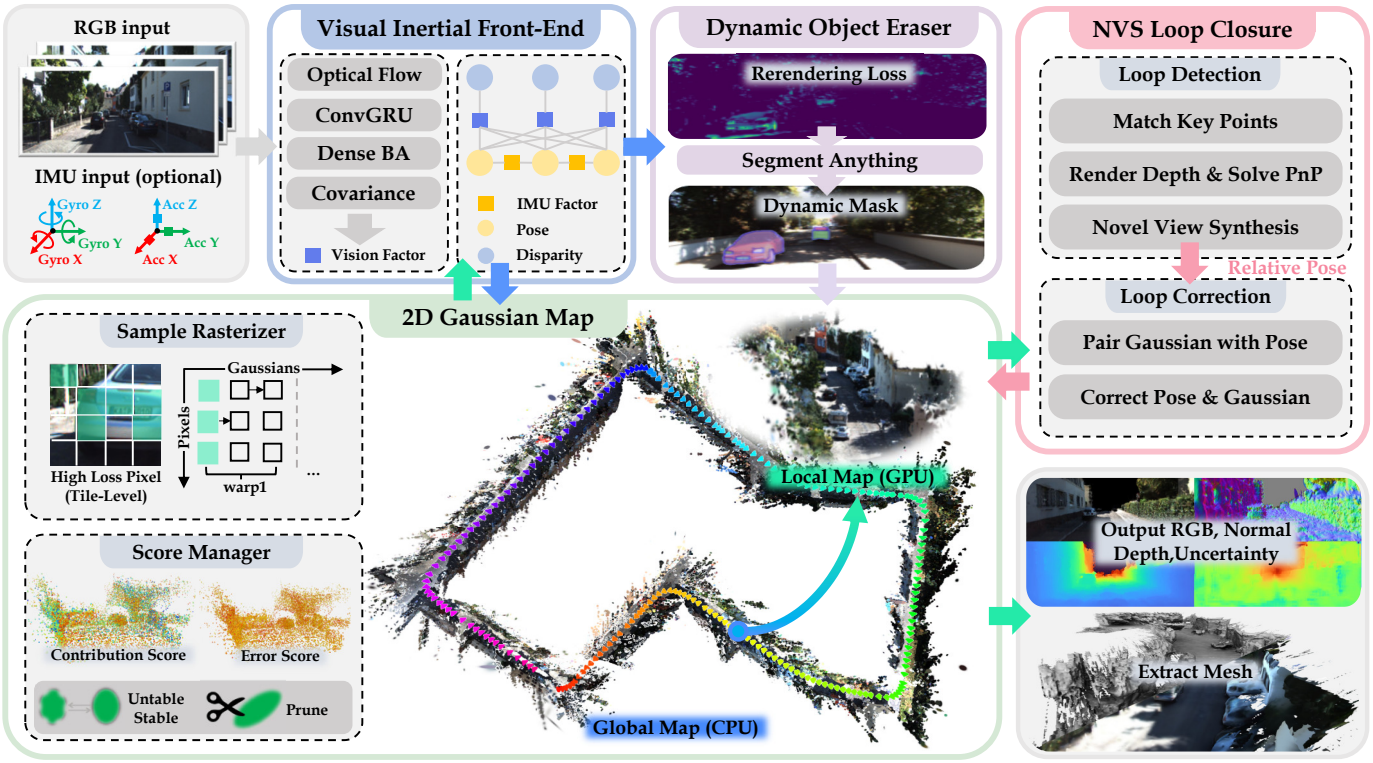


Fig. 2: Pipeline of VINGS-Mono. RGB and IMU readings are processed by the Visual Inertial Frontend to calculate pose and inverse depth. Based on this, the 2D GS Map is incrementally updated, comprising a score manager, sample rasterization, and pose refinement. The NVS Loop Closure employs novel view synthesis for efficient loop detection and correction seamlessly. Furthermore, the Dynamic Object Eraser helps minimize the impact of moving objects on the framework.

We use the Gauss-Newton method to simultaneously optimize both the pose and the inverse depth. $\Delta\xi_i$ represents a update of camera pose and $\mathbf{C}_{ii}$ is a diagonal matrix, the rest variables are sub-blocks of the matrix in Eq. 3.

To eliminate the depth state, we calculate the Schur Complement of the Hessian with respect to $\mathbf{C}$, which effectively constructs an inter-frame pose constraint containing the linearized BA information. The calculations in Eq. 4 can be efficiently parallelized on the GPU:

$$(\mathbf{B}_i - \mathbf{E}_i\mathbf{C}_i^{-1}\mathbf{E}_i^T)\Delta\xi_{i,1,\dots,N} = \mathbf{v}_i - \mathbf{E}_i\mathbf{C}_i^{-1}\mathbf{z}_i. \tag{4}$$

where $\mathbf{B}_i, \mathbf{E}_i, \mathbf{C}_i, \mathbf{v}_i, \mathbf{z}_i$ are blocks in Eq. 3. After updating the pose, we can then update the inverse depth state using:

$$\Delta(\mathbf{d}_i^{-1}) = \mathbf{C}_i^{-1}\left(\mathbf{z}_i - \mathbf{E}_i^T\Delta\xi_{i,1,\dots,N}\right). \tag{5}$$

We use the convex upsampling method from DROID-SLAM, as defined in RAFT, to obtain full-resolution depth. This upsampling approach takes a convex combination of the neighboring depth values, with the upsampling weights estimated by the GRU network. In more complex scenes, we also use monocular depth estimation networks to provide priors for the DenseBA optimization.

B. Visual Inertial Factor Graph

The factor graph we constructed includes visual factors and optionally IMU pre-integration factors, the factor graph optimization is implemented via GTSAM [49]. The state variables in the IMU pre-integration factor are:

$$\mathbf{b}_k = \begin{bmatrix}\mathbf{b}_{a,k} & \mathbf{b}_{g,k}\end{bmatrix}, \mathbf{x}_k = \begin{bmatrix}\mathbf{T}_{b_k}^w & \mathbf{v}_{b_k}^w & \mathbf{b}_k\end{bmatrix}. \tag{6}$$

Where $\mathbf{T}_{b_k}^w$ represents the IMU pose in the world frame, $\mathbf{v}_{b_k}^w$ represents the velocity, and $\mathbf{b}_{a,k}, \mathbf{b}_{g,k}$ are the accelerometer and gyroscope biases. The visual factor is a pose constraint derived through Schur elimination as described above. Note that the poses in our factor graph are defined as transformations from the IMU frame to the world frame. Our visual factor is expressed as:

$$\begin{aligned}\mathbf{T}_w^{c_k} &= (\mathbf{T}_{b_k}^w\mathbf{T}_c^b)^{-1} \\ \mathbf{X}_c &= \begin{bmatrix}\xi_w^{c_0 T} & \xi_w^{c_1 T} & \dots & \xi_w^{c_k T}\end{bmatrix}^T \\ \mathbf{E}_c(\mathbf{X}_c) &= \frac{1}{2}\mathbf{X}_c^T\mathbf{H}_c\mathbf{X}_c - \mathbf{X}_c^T\mathbf{v}_c.\end{aligned} \tag{7}$$

where $\mathbf{T}_c^b$ denotes the camera-to-IMU extrinsic transformation, $\xi_w^{c_k}$ is Lie algebra of $\mathbf{T}_w^{c_k}$, $\mathbf{H}_c, \mathbf{v}_c$ are the information matrix and vector during DBA.

We follow the method of [51] to compute IMU pre-integration. The residual for preintegrated IMU measurement

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

can be defined as $\mathbf{r}_b\left(\mathbf{x}_k, \mathbf{x}_{k+1}\right)$:

$$\begin{bmatrix} \mathbf{R}_w^{b_k}\left(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2}\mathbf{g}^w\Delta t_k^2 - \mathbf{v}_{b_k}^w\Delta t_k\right) - \hat{\alpha}_{b_k}^{b_{k+1}} \\ \mathbf{R}_w^{b_k}\left(\mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w\Delta t_k - \mathbf{v}_{b_k}^w\right) - \hat{\beta}_{b_k}^{b_{k+1}} \\ \mathrm{Log}((\mathbf{R}_{b_k}^w)^{-1}\mathbf{R}_{b_{k+1}}^w(\hat{\gamma}_{b_{k+1}}^{b_k})^{-1}) \\ \mathbf{b}_{a,k+1} - \mathbf{b}_{a,k} \\ \mathbf{b}_{g,k+1} - \mathbf{b}_{g,k} \end{bmatrix} \tag{8}$$

Where $\mathbf{p}_{b_k}^w$ and $\mathbf{R}_{b_k}^w$ represents the translation vector and rotation matrix of $\mathbf{T}_{b_k}^w$, $\hat{\alpha}_{b_k}^{b_{k+1}}, \hat{\beta}_{b_k}^{b_{k+1}}, \hat{\gamma}_{b_k}^{b_{k+1}}$ are the IMU pre-integration terms [19], $\mathbf{g}^w$ is the gravity, Δt_k is the time interval.

C. Depth Uncertainty Estimation

Our probabilistic depth uncertainty is inherently derived from the information matrix of the underlying Dense Bundle Adjustment (DBA) process. The primary goal of depth uncertainty is to suppress noise and mitigate artifacts. Using the sparse form of the Hessian matrix, we can calculate the marginal covariances for per-pixel inverse depth values. The marginal covariance of the inverse depth is formulated as:

$$\begin{aligned} \mathbf{\Sigma}_T &= (\mathbf{H}/\mathbf{C})^{-1} \\ \mathbf{\Sigma}_{d^{-1}} &= \mathbf{C}^{-1} + \mathbf{C}^{-1}\mathbf{E}^T\mathbf{\Sigma}_T\mathbf{E}\mathbf{C}^{-1} \\ &= \mathbf{C}^{-1} + \mathbf{C}^{-1}\mathbf{E}^T(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{E}\mathbf{C}^{-1} \\ &= \mathbf{C}^{-1} + (\mathbf{L}^{-1}\mathbf{E}\mathbf{C}^{-1})^T(\mathbf{L}^{-1}\mathbf{E}\mathbf{C}^{-1}). \end{aligned} \tag{9}$$

where $\mathbf{L}$ is the lower triangular Cholesky factor. Finally, with all the information provided by the visual-inertial front end – including poses $\{T_t\}$, depths $\{D_t\}$, their associated depth uncertainties $\{U_t\}$, and input RGB images $\{I_t\}$, we can incrementally construct and maintain our 2D Gaussian Map.

V. 2D Gaussian Map

We will first give a comprehensive introduction to the online mapping process, followed by a detailed explanation of the score manager, the sample rasterizer, and the pose refinement mechanism.

A. Online Mapping Process

For the initialization of the mapping module, the 2D Gaussian Map is initialized after the VIO front end has processed the first batch of keyframes. For each frame $\{I_t, D_t, U_t, T_{c_t}^w\}$, pixels with excessive depth or high uncertainty are masked out. Then, k points ($k = 50,000$) are randomly sampled and projected to obtain point clouds in the world coordinate system. The Gaussian properties are initialized following the method described in 2DGS [52].

$$\mathcal{G} = \{g_i : (\mu_i, r_i, \alpha_i, c_i) \mid \forall g_i \in \mathcal{G}\}. \tag{10}$$

We follow the rendering approach of 2DGS to render color C, depth D, normal N and accumulation A, as shown in Eq. 11, where z_i represents the depth value of the Gaussian center in the camera coordinate system, and n_i represents the normal vector of the gaussian ellipsoid, with its positive direction aligned with the ray. For simplicity in notation, the rendering process (Eq. 11) will be represented as $\mathcal{R}(\cdot)$.

$$\begin{aligned} f(p) &= \alpha \cdot \exp(-\frac{1}{2}(p-\mu)^T(p-\mu)) \\ (C, D, N, A) &= \sum_{i=1}^{N}(c_i, z_i, n_i, 1) f_i \Pi_{j=1}^{i-1}(1-f_j). \end{aligned} \tag{11}$$

The Mapping module and the VIO Front End operate as two parallel threads. For the subsequent incremental mapping process, we do not adopt the original 3DGS clone-and-split strategy for densification because we found in practice that the reset opacity operation performs unsatisfactorily in the GS-SLAM setting. Instead, we demonstrate that adding a relatively large number of Gaussians first and then pruning unnecessary ones is highly effective.

When the front end adds a new keyframe $\mathbf{T}_{c_t}^w$, new Gaussian ellipsoids are added before training. First, the color and depth of $\mathbf{T}_{c_t}^w$ are rendered. Then, two operations are performed: deleting conflicting Gaussians and adding necessary ones. Gaussians with high depth or RGB losses within the view frustum are removed, as well as those with excessively large projection radii. This is determined by projecting each Gaussian's center onto the image. After deletion, the accumulation map is re-rendered, and new Gaussians are added using depth information from pixels where the accumulation remains low. The number of new Gaussians is proportional to the area of low-accumulation pixels relative to the total pixel area. This redundant addition followed by selective pruning ensures robust performance, outperforming the original densification method in GS-SLAM, especially in forward-view scenarios like driving.

After adding the new Gaussian ellipsoids, we randomly sample frames from the latest keyframe list in the VIO Frontend for training. During training, the loss function $\mathcal{L}$ is computed according to 2DGS [52], as shown in Eq. 12. We add an additional accumulation loss to ensure that the masked-out regions do not contain black Gaussians. We change L_d by using weights normalized by the inverse of the depth uncertainty $\sum_{d^{-1}}$ from the front-end output and we also add an additional accumulation loss to ensure that the masked-out regions do not contain black Gaussians. During each training iteration, we record and update the local contribution score and error score of each Gaussian within the current keyframe list. These variables are used to maintain the Gaussians, which will be explained in detail in the next subsections.

$$\mathcal{L} = \lambda_{rgb}\mathcal{L}_{rgb} + \lambda_{depth}\mathcal{L}_d + \lambda_{norm}\mathcal{L}_n + \lambda_{acc}\mathcal{L}_{acc}. \tag{12}$$

B. Score Manager

We propose a scoring mechanism to manage each Gaussian ellipsoid. This management involves status control (stable/unstable), storage control, and GPU-CPU transfer. The detailed algorithm flow is illustrated in Algorithm. 1.

For a given keyframe list, we define a contribution score and an error score for each Gaussian. Our goal is to ensure that each Gaussian achieves the highest contribution score while causing the smallest error score. In a set of keyframes, a

Gaussian contributes a weight to every pixel it touches. As shown in Eq. 13, for a Gaussian g, we accumulate the weights over P pixels it touches to compute its total contribution to frame t. We then sum these contributions over the K keyframes in the list to obtain the Gaussian's total contribution score $S_C(g)$. Each pixel has a loss value $\mathcal{L}_{rgb}(u)$, and we compute the weighted sum of pixel losses over the PP pixels the Gaussian touches to get its total loss for frame t. Finally, we select the highest error score among the K keyframes in the list as the Gaussian's error score $S_E(g)$.

$$
\begin{aligned}
S_C(g) &= \sum_{t=0}^{K}\sum_{u=0}^{P} f_i \Pi_{j=1}^{i-1}(1-f_j) \\
S_E(g) &= \max(\{\sum_{u=0}^{P} \mathcal{L}_{rgb}(u) f_i \Pi_{j=1}^{i-1}(1-f_j)\}_{t=0,\dots,K}) \\
ID(g) &= \mathrm{argmax}_t(\{\sum_{u=0}^{P} f_i \Pi_{j=1}^{i-1}(1-f_j)\}_{t=0,\dots,K}).
\end{aligned} \tag{13}
$$

The calculation logic for these two scores is consistent. The contribution of a Gaussian to the current keyframe list is calculated as the sum of its contributions across all frames. However, if a Gaussian causes a large error in even a single frame, we consider its properties to be incorrect and the Gaussian should be removed. Therefore, we compute contribution scores using a summation, while error scores are computed using the maximum value. Furthermore, while calculating the contribution score, we also compute each Gaussian's contribution to individual frames. To capture this, we introduce a new variable for each Gaussian, denoted as $ID(g)$. This variable represents the specific frameID where the contribution of Gaussian g is the highest. After calculating S_E, S_C, and ID, we manage the Gaussians based on these variables and the current pose through three processes: status control, storage control, and GPU-CPU transfer.

During status control, we handle the transitions of Gaussians on the GPU between two states: stable and unstable. The unstable Gaussians are mainly those located near adjacent frames with relatively incorrect parameters, while the stable Gaussians primarily consist of the majority from historical frames and well-optimized ones near the current frame. The purpose of defining these two states is to speed up optimization by masking unstable Gaussians during sparse Adam updates and to identify which Gaussians should be removed during storage control. Every Δn_{status} $(= 400)$ iterations, which corresponds to a full replacement of the local keyframe set, unstable Gaussians with contribution scores S_C lower than $S_C^{status}(= 10^{-4})$ are transitioned to the stable state, ensuring that unnecessary Gaussians are effectively removed, thereby reducing storage and computational overhead. Conversely, stable Gaussians with error scores S_E exceeding a predefined threshold $S_E^{status}(= 0.5)$ are transitioned to the unstable state, with their S_C and S_E reset to zero. This ensures that if the system revisits previously explored areas, these stable Gaussians can be reintroduced into the optimization process, allowing for further refinement of historical Gaussians when they become relevant again.

During storage control, we remove unnecessary Gaussians to optimize memory and computational resources. Every $\Delta n_{\text{storage}}$ $(= 200)$ iterations, we prune Gaussians with contribution scores S_C below a certain threshold $S_C^{storage}(= 0.5)$ and a status marked as unstable. This approach is necessary because, in experiments involving multi-room environments or complex road structures, we noticed that previously visited positions (historical Gaussians) can reappear in the view frustum. However, due to occlusion or distance, these Gaussians have a low contribution (S_C) to the current keyframe list. When pruning, it's important to distinguish these historical Gaussians within the frustum from those with genuinely low contribution. Relying solely on projection radius, distance, or S_C is insufficient and risks pruning important historical Gaussians, which would be disastrous. Our stable status resolves this issue by effectively preserving these Gaussians. This is particularly relevant in turning points, where historical Gaussians often reenter the view frustum. In addition, storage control is highly effective and adaptable, and it can be applied to all Gaussian-based methods. According to our experiments, using storage control to reduce the number of Gaussian ellipsoids can cut their count by half without compromising rendering quality.

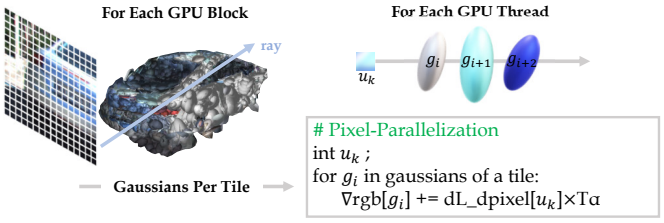
During GPU-CPU transfer, we address the memory limitations of GPU when handling large-scale street-level scenes containing tens of millions of Gaussians. We transfer Gaussians between CPU memory (RAM) and GPU memory. Every ΔK $(= 8)$ keyframes, Gaussians are transferred between CPU and GPU memory based on their distance from the current pose. To reduce computational overhead, we use the pose index $\mathrm{ID}(g)$ to calculate distances between poses instead of directly computing distances for all Gaussian centers. Gaussians linked to poses within a specified distance threshold τ are transferred from CPU to GPU memory for faster access, while those beyond τ are moved from GPU to CPU memory and removed from GPU storage. This strategy dynamically balances memory usage, ensuring relevant Gaussians stay on the GPU while less relevant ones are offloaded to the CPU, maintaining both efficiency and rendering quality.

C. Sample Rasterizer

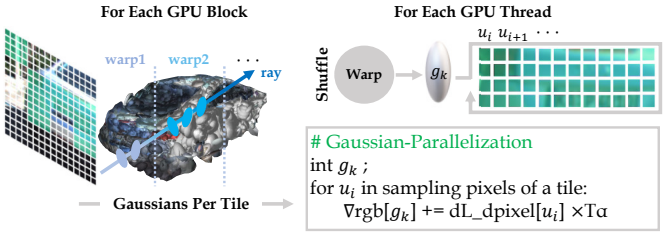
In the original Gaussian Splatting method, the process for backpropagation mirrors the forward propagation in a symmetrical fashion [53]. Each GPU block [54] is responsible for one tile, with each tile containing 16×16 pixels. The 256 threads within a block each handle the backpropagation for one pixel, specifically propagating the loss L_i of pixel p_i back to the Gaussians corresponding to that pixel. Consequently, the number of iterations each thread performs depends on the number of Gaussians associated with that pixel. This results in a bottleneck effect, where the overall backpropagation time is determined by the pixel with the maximum number of Gaussians.

Inspired by Taming 3DGS [55] and prior NeRF approach [8], which achieve backpropagation by sampling pixels, we examined why pixel sampling did not accelerate backpropagation in the vanilla Gaussian Splatting's rasterization pipeline. The issue was that pixel sampling merely reduced

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



(a) 2DGS Backward Process.



(b) Ours Backward Process.

Fig. 3: Sample Rasterizer. In our backpropagation process, each thread is responsible for one Gaussian, and the number of iterations depends on the number of sampled pixels.

the number of active threads per block, without changing the number of iterations each thread performs (still dependent on the number of Gaussians associated with each pixel).

To address this, we introduced a modification during forward propagation. As shown in Fig. 3, for each thread, we store intermediate variables in a buffer at intervals of 32 Gaussians. During backpropagation, we divided the GPU into multiple warps [56] for computation, where each warp, consisting of 32 threads, performs backpropagation on the Gaussians within the warp. This change reduces the number of iterations per thread to match the number of pixels associated with the current tile. We further optimized by selecting a subset of pixels with the highest loss rates r within each tile for backpropagation. With this approach, each thread's iteration count is reduced to $256 \times r$. In our experiments, we set $r = 0.5$, which resulted in a backpropagation speedup of 273% compared to the original method. Detailed experimental results are documented in Sec. VIII-E1.

D. Single-to-Multi Pose Refinement

Existing GS-based SLAM [2], [5], [57], optimizes localization by propagating gradients to the positional property of Gaussians, which then pass these gradients on to the current frame's pose. However, this method is relatively inefficient. In Eq. 13, we associated Gaussians with their respective keyframes. Based on this, we implemented a system where gradients of different Gaussian poses are propagated as pairs to different keyframe poses, thereby enabling the rendering of a single frame to optimize multiple frame poses. The optimized poses replace the visual frontend's pose buffer, facilitating further rounds of optimization.

As in Eq. 14, for the kth keyframe, the pose is represented as $\mathbf{T}_{c_k}^{w}$. From S_C, we obtain the subset of Gaussians associated

Algorithm 1 Score Manager

Input: $\Delta n_{\text{status}} = 400$, $\Delta n_{\text{storage}} = 200$, $\Delta K = 8$
Thresholds: $S_C^{\text{status}} = 10^{-4}$, $S_E^{\text{status}} = 0.5$, $S_C^{\text{storage}} = 0.5$, distance threshold τ
Initialize: $j = 0$
for each keyframe k **do**
  for each iteration i **do**
    $j = j + 1$
    if $j \mod \Delta n_{\text{storage}} = 0$ **then**
      for each Gaussian g **do**
        if g is unstable **and** $S_C(g) < S_C^{\text{status}}$ **then**
          Set g to stable
        else if g is stable **and** $S_E(g) > S_E^{\text{status}}$ **then**
          Set g to unstable, reset $S_E(g)$, $S_C(g)$
        end if
      end for
    end if
    if $j \mod \Delta n_{\text{storage}} = 0$ **then**
      for each Gaussian g **do**
        if g is unstable **and** $S_C(g) < S_C^{\text{storage}}$ **then**
          Prune g
        end if
      end for
    end if
    if $k \mod \Delta K = 0$ **then**
      for each Gaussian g **do**
        if Pose distance $d(T_{ID(g)}) < \tau$ **then**
          Transfer g to GPU storage
        else
          Move g to CPU, remove from GPU storage
        end if
      end for
    end if
  end for
end for

with this frame, denoted as $\{g_{c_k}\}$. We introduce the camera pose transformation matrix $\mathbf{T}_{c_k}^{\hat{c_k}}$ as an optimization variable. Subsequently, we render RGB image $\hat{I}_k$ and perform backpropagation to optimize the poses of all keyframes within the visible range by minimizing the rendering loss. This process effectively adjusts the keyframe poses based on their rendering performance to enhance the overall quality of the visualization.

$$\begin{aligned} \hat{\mu_k} &= \mathbf{T}_{c_k}^{w}\mathbf{T}_{c_k}^{\hat{c_k}}\mathbf{T}_{w}^{c_k}\mu_k, \mathbf{T}_{c_k}^{\hat{w}} = \mathbf{T}_{c_k}^{w}(\mathbf{T}_{c_k}^{\hat{c_k}})^{-1} \\ \hat{I}_k &= \mathcal{R}(\{\hat{\mu_k}, s_k, c_k, r_k\}, \mathbf{T}_{c_k}^{\hat{w}}) \\ &\min_{\{\mathbf{T}_{c_k}^{\hat{c_k}}\}} \mathcal{L}_{rgb}(\hat{I}_k, I_k). \end{aligned} \tag{14}$$

VI. NVS Loop Closure

In monocular setups that lack scale information, loop closure is essential to eliminate accumulated errors, especially in large-scale environments. We propose a novel Gaussian Splatting based loop detection and correction method. Instead of using the Bag of Words (BoW) approach for loop detection, we leverage the novel view synthesis capabilities of gaussian splatting from new viewpoints to determine if a loop has

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

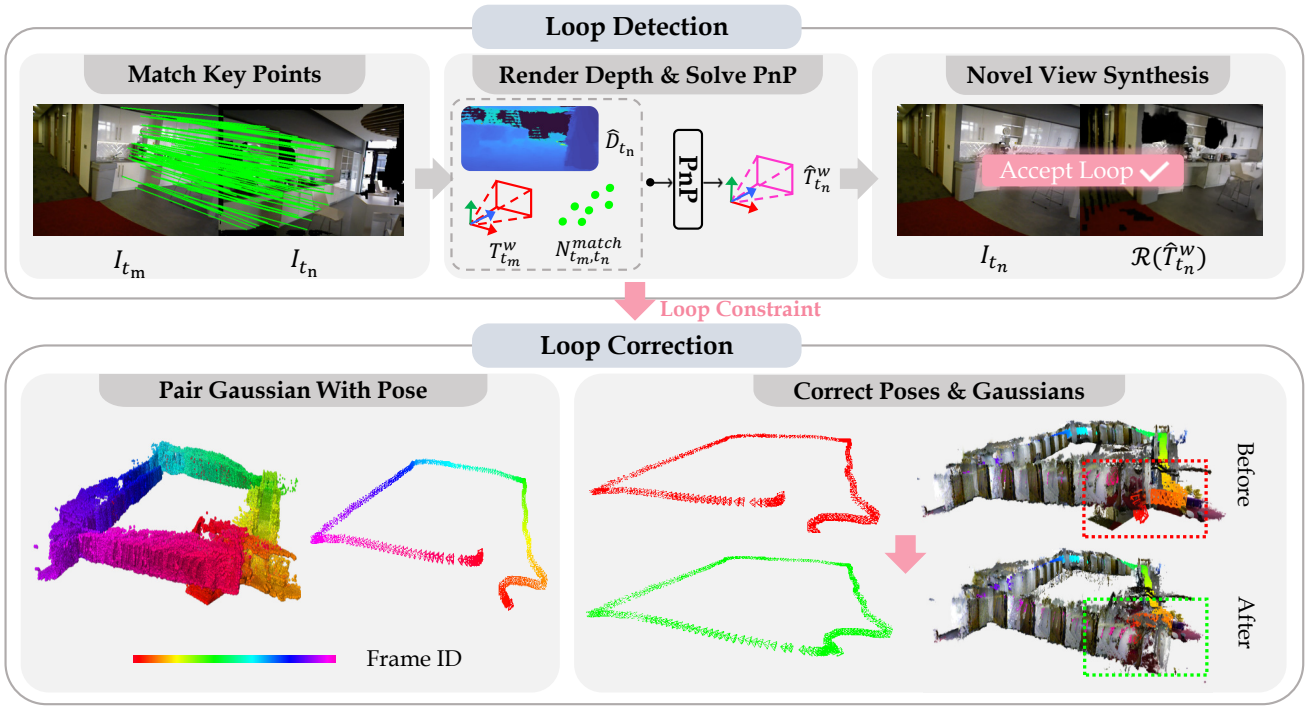


Fig. 4: Pipeline of NVS Loop Closure. We perform feature matching, filtering, and novel view synthesis on keyframes that meet the distance threshold requirements to achieve loop detection. Once a loop is detected, we implement loop correction of the pose and Gaussian map through pairwise Gaussian with pose alignment and graph optimization.

been detected (Sec. VI-A). Following this, we use graph optimization to correct poses and use gaussians' frame index $ID(g)$ to correct the 2D Gaussian Map (Sec. VI-B).

A. Loop Detection

As illustrated in the upper section of Fig. 4, our loop detection process comprises three key steps: matching feature points with historical frames, deriving the relative poses of the two frames from the matched feature points and rendered depths, and synthesizing a novel view using the new poses to ascertain the presence of a loop closure.

1) Match Key Points: We extract and match feature points [58] with historical frames $\{I_{t_k}\}$ located within a specified range of the current pose $T^w_{t_n}$ and with a frame ID difference exceeding ten from the current frame. The number of feature points successfully matched between historical frame I_{t_k} and current frame, denoted by $N_{match}(t_k, t_n)$, is systematically recorded. Frames whose match counts exceed the threshold $N^{th}_{match}(= 50)$ are then organized in descending order based on the number of matches and we denote this set as $\{I_{t_k}\}^{filt}$.

2) Render Depth & Solve PnP: We sequentially check $\{I_{t_k}\}^{filt}$ in descending order based on the number of key-points. First, we use $T^w_{t_n}$ to render the depth map $\hat{D}_{t_n}$ of current frame. For frame $I_{t_m} \in \{I_{t_k}{}^{filt}\}$, we perform a Perspective-n-Point (PnP) computation using the matched feature points $N^{match}_{t_m,t_n}$ to estimate the relative pose $T^{\hat{t}_m}_{t_n}$ between t_m and t_n. Subsequently, the global pose is computed as $\hat{T}^w_{t_n} = T^w_{t_m} \hat{T}^{t_m}_{t_n}$. It is important to note that, due to the instability of PnP when using feature points with distant depth values, we restrict the selection to points with depth values below a fixed threshold (setting 20m in KITTI and 25m in Waymo).

3) Novel View Synthesis: The core of the loop detection problem is determining whether two images capture the same scene. With the inherent novel view synthesis capability of 3DGS, this problem transforms into verifying whether the newly captured image can serve as a novel viewpoint of the Gaussian Map. Loop detection can be directly assessed by calculating the L1 Loss between the newly synthesized view $\mathcal{R}(\hat{T}^w_{t_n})$ and the original image I_{t_n}. The criterion is as follows: if the color loss is below a specific threshold or less than one-tenth of the median color loss among other frames in $\{I_{t_k}\}^{\text{filt}}$, a loop is considered detected.

B. Loop Correction

After detecting a loop closure and the corresponding loop closure constraints, we correct the poses of historical keyframes and the Gaussian Map. It is not feasible to directly optimize the Gaussian Map based on loop closure constraints because, in large-scale environments, loop closure errors tend to be significant. Directly retraining the Gaussian Map with corrected poses may not converge effectively. Therefore, we first associate each Gaussian with a historical keyframe pose to ensure consistency between our Gaussian Map and poses. On this basis, we then proceed with fine-tuning.

1) Pair Gaussian with Pose: For all historical keyframes, we forward propagate and record the contribution score of each Gaussian in each frame sequentially. Each Gaussian selects the pose corresponding to its highest score as the matched pose.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

Due to the extremely fast rendering speed, this process takes approximately two seconds for one thousand frames.

2) Correct Pose & Gaussians: We construct the pose graph for all historical frames $\{T_{c_k}^w\}$ and add loop closure constraints to it. Then, we perform graph optimization to obtain the updated global poses of the historical keyframes $\{T_{c_k}^{w\prime}\}$. For each historical keyframe, we calculate its scale using the ratio of the translation vector norms before and after the transformation. As described in Eq. 15, k is $ID(g_i)$ and $R(\cdot)$ represents the transformation from a quaternion to a rotation matrix. We compute each Gaussian's new position μ' and rotation r_i, while keeping other attributes unchanged. Subsequently, we retrain the model for one hundred iterations on the global set of historical keyframes and record the S_C. In large-scale scenes, only the Gaussians on the GPU participate in retraining process, while those on the CPU for distant regions are updated only in terms of position and rotation. This strategy is crucial, as it ensures that during revisiting, both the camera pose and Gaussian positions remain consistent. Significant deviations in Gaussian positions could cause severe issues, leading to catastrophic training divergence. Finally, we perform an additional step to prune Gaussians based on S_C to further optimize storage overhead.

$$\begin{aligned} \mu_i{}' &= T_{c_k}^{w\prime} T_w^{c_k} \mu_i \\ r_i{}' &= R^{-1}(T_{c_k}^{w\prime} T_w^{c_k} R(r_i)). \end{aligned} \tag{15}$$

VII. Dynamic Object Eraser

The underlying assumption of Gaussian Splatting is that scenes are static. However, in real-world applications, especially in large-scale environments, dynamic distractors like vehicles or pedestrians are common. Previous dynamic Gaussian Splatting methods [59]–[62] were implemented in offline training settings. These approaches model the 4D space and train the relationships between Gaussian properties and time across the entire dataset in an offline manner. However, such methods are not suitable for SLAM, which requires incrementally loading data. Considering that SLAM's mapping is an online process and that Gaussian Splatting has the capability for novel view synthesis, we designed a heuristics-guided segmentation method to distinguish masks of dynamic objects.

First, we apply an accelerated open-set semantic segmentation model [63] on the entire image to generate a set of semantic masks, denoted as $\{M_k\}_{k=0,1,\ldots,K}$. When a new keyframe I_t arrives, we render the color of current frame $R(T_{c_t}^w)$ before adding new Gaussians. Next, we calculate the SSIM loss and the L1 loss separately with respect to the new keyframe. We observed that SSIM is particularly sensitive to textures, whereas the L1 loss is more sensitive to color value differences. By multiplying these two losses, we compute the re-rendering Loss, $\mathcal{L}_{\text{re}}$. Note that this loss is initially calculated at pixel level before taking the overall average, we denote the 90% percentile of this pixel-level loss as $\mathcal{L}_{\text{re}}^{90\%}$. However, for dynamic objects with relatively smooth textures, the re-rendering loss is only primarily noticeable around the edges. To address this issue, we modify the re-rendering loss calculation by incorporating depth uncertainty as mention in Eq. 9, $\mathcal{L}_{\text{re}} = \mathcal{L}_{\text{SSIM}} \cdot \mathcal{L}_1 \cdot \Sigma_{d^{-1}}$. This enables us to more effectively identify and determine the mask for moving objects M_{dyn}.

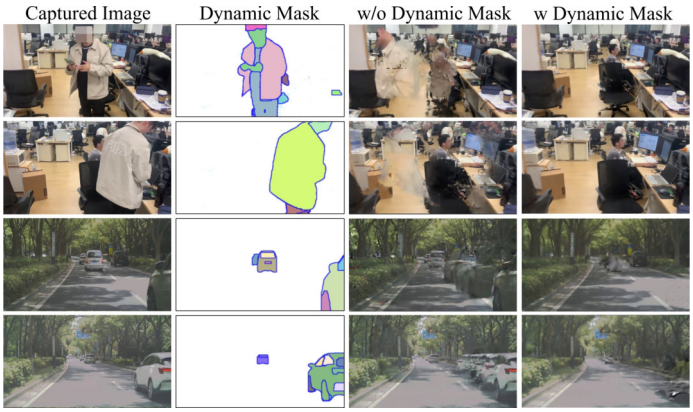


Fig. 5: Effect of Dynamic Object Eraser. Our dynamic eraser can filter out moving people indoors and fast-moving vehicles outdoors, preventing the Gaussian map from being affected by dynamic floaters.

$$\begin{aligned} M_{dyn,k} &= \left(\frac{\sum \mathbf{1}(\mathcal{L}_{re}(M_k) > \mathcal{L}_{re}^{90\%})}{\sum \mathbf{1}(M_k)} > \gamma\right) \wedge \left(\overline{\mathcal{L}_{re}}(M_k) > \mathcal{L}_{re}^{th}\right) \\ M_{dyn} &= \bigcup_{k=0}^{K} M_{dyn,k}. \end{aligned} \tag{16}$$

Where $\sum \mathbf{1}(\cdot)$ represents pixel number of the mask and γ is set to 20%. We identify a semantic mask as a dynamic mask when the rerendering loss in the region covered by the mask exceeds a certain threshold in more than γ of the mask's pixels, and the color rendering loss is relatively high. We filter them out during the addition of new Gaussians and in the subsequent rendering process.

VIII. Experimental Evaluation

We conducted comprehensive comparative experiments to evaluate our framework's tracking performance and mapping performance emphasized by Gaussian Splatting. Additionally, we carried out comparative experiments on the dynamic object eraser and performed ablation studies on the individual components of our system. Finally, we analyzed the runtime of the system and introduced the mobile app we developed, along with real-world experiments.

A. Experimental Setup

1) Datasets and Metrics: To validate the effectiveness and robustness of our algorithm, we exclusively use real-world datasets rather than simulated data. Our experiments include two large-scale indoor scenarios, a classic dynamic indoor SLAM dataset, and five different outdoor scenarios characterized by varying lighting conditions, movement speeds, and capture devices. Additionally, we collect real-world data using consumer-level smartphone's sensors.

For indoor scenarios, we evaluated ScanNetV1 [64] and BundleFusion [65]. ScanNetV1 [64] is a widely used RGB-D dataset in SLAM research, offering over 1500 indoor scenes

with 3D camera pose annotations. We selected six large-scale scenes with significant lighting variations for our experiments. The official BundleFusion dataset was tested on five scenes (apt0, apt2, copyroom, office0, office2), with reference trajectories provided by the official dataset.

For outdoor scenarios, we conducted experiments on three driving-view datasets: KITTI [66], KITTI-360 [67], and Waymo [68], a drone-view dataset: MegaNeRF [69], and a cycling-view dataset: Hierarchical dataset [70]. The KITTI dataset was collected in urban, rural, and highway settings operating at 10 Hz, along with LiDAR scans captured by a Velodyne HDL-64 and ground-truth trajectories recorded by an OXTS 3003 GPS/IMU. We use KITTI Odom splits in our Experiments. The KITTI-360 [67] dataset consists of 9 sequences covering over 73 km, with data captured by a stereo pair at 10 Hz (rectified resolution: 1408×376) and ground-truth poses derived through large-scale optimization combining OXTS data, laser scans, and multi-view images. The Waymo [68] dataset was collected in San Francisco, Mountain View, and Phoenix using five LiDAR sensors and five high-resolution cameras (rectified resolution: 1920×1280) operating at 10 Hz, with official vehicle poses provided for each range image. For MegaNeRF [69], we evaluated two scenes: Building, which features grid-pattern footage of a 500×250m^2 industrial area, and Rubble, both with GPS-derived camera poses. The Hierarchical 3DGS [70] dataset includes walking and cycling data; the Campus subset was captured with five GoPro HERO 6 cameras (resolution: 1444×1080), while the Small City subset was recorded at 7 km/h. Reference trajectories were generated using COLMAP [71] with a hierarchical mapper and per-chunk bundle adjustment.

To assess global consistency between estimated and ground-truth trajectories, we calculate Absolute Trajectory Error (ATE) [72] for indoor datasets and Relative Pose Error (RPE) following [66] for large scale outdoor datasets via the evo toolkit [73]. For rendering quality, we applied metrics like Peak Signal-to-Noise Ratio (PSNR), Learned Perceptual Image Patch Similarity (LPIPS), and Structural Similarity Index (SSIM).

2) Parameters and Implementation Details: All experimental results were recorded using a single RTX 4090 GPU and an Intel Xeon 6133 CPU (2.50GHz). The preset parameters are divided into three components: the front end, Gaussian maps, and loop closure detection.

For the VIO front end, we set the optical flow threshold between two frames to be greater than 2.4 pixels for a new keyframe to be considered. The length of the local keyframe list is set to eight. For Gaussian maps, each keyframe in the ScanNet and BundleFusion datasets undergoes 80 iterations of rendering training and 10 iterations of pose optimization. For MegaNeRF, each new keyframe undergoes 50 iterations of rendering training without pose refinement. For other datasets and real-world experiments, 100 iterations of training are performed per keyframe. Additionally, in the monocular settings of real-world experiments, we use [74] to obtain depth priors, enhancing the geometric quality of the results.

Following the advice of the original 3D Gaussian approach, we reduce the learning rate for positional attributes in outdoor scenes. The weights of the loss function are $\lambda_{rgb} = 1.0, \lambda_{depth} = 0.5, \lambda_{normal} = 0.1, \lambda_{\alpha} = 0.1$ For loop closure detection, the filtering radius is set to 15 meters for indoor scenes and 50 meters for outdoor scenes. The threshold for filtering by the number of matching points is set to 50, and the re-rendering loss threshold is set to 0.15.

Some methods in the comparison experiments face difficulties running successfully in outdoor scenarios. To ensure that the comparison experiments are reasonable and meaningful, we provide pseudo ground-truth depth or increase the number of iterations where necessary. Specific configurations will be explained in detail in the experimental section.

B. Localization Performance

In this section, we compare the pose estimation performance of VINGS-Mono with both traditional SLAM methods and Gaussian Splatting based methods.

1) VO Comparison: We conducted experiments on two large-scale indoor scene datasets including ScanNet [64] and BundleFusion [65], as well as two outdoor scene datasets Hierarchical [70] and Waymo [68]. For our evaluation, we selected representative traditional SLAM methods ORB-SLAM3 [14] and DROID-SLAM [47]. To ensure a fair comparison for DROID-SLAM, we omitted the global BA post-processing step after the full run. It's worth noting that these methods do not have the capability to construct Gaussian maps. Additionally, we included two NeRF-based SLAM methods [16], [75] and two state-of-the-art monocular Gaussian-based SLAM methods MonoGS [5] and Photo-SLAM [6]. For indoor scenes, as shown in Tab. I, our method performs on par with traditional SLAM methods but significantly outperforms existing monocular GS-based SLAM methods.

For the outdoor scenes which are our main focus, as shown in Tab. II, our method achieves better localization accuracy compared to existing approaches. Additionally, due to the faster movement speed (resulting in insufficient overlap between consecutive frames) and the large ground area in outdoor large-scale scenes like Hierarchical-Smallcity, which causes weaker textures, both ORB-SLAM and PhotoSLAM (which uses ORB as its frontend) relocated back to the origin. MonoGS, on the other hand, displayed a completely black image. All three methods successfully tracked less than half of the trajectory, as shown in Fig. 6. Therefore, for ORB-SLAM and PhotoSLAM, we only recorded the ATE for the first fifty frames and marked with an asterisk(*) in Tab II. However, our method was still able to handle large-scale scenes with relatively faster movement speeds effectively.

2) VIO Comparison: We compared the pure odometry accuracy on the competitive KITTI [66] and KITTI360 [67] datasets. Due to the significant storage and computational demands of kilometer-scale urban scenes, no existing NeRF/3DGS-based SLAM methods can run on both datasets. Therefore, we selected two feature-based methods, VINS-Mono and ORB-SLAM3, as well as two advanced learning-based methods, iSLAM [43] and Selective-VIO [42], for comparison. To test whether our algorithm can robustly adapt to different IMU frequencies, we used the 10Hz KITTI sync

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

TABLE I: Monocular Localization results (ATE [cm]) on the indoor datasets ScanNet and BundleFusion. Red, orange, and yellow represent the best, second-best, and third-best performance, respectively. For all evaluation scenarios, the same dataset with ground truth values was used as a reference to compute the average metrics. ("Tra" represents traditional SLAM methods, "DL" refers to deep learning–based SLAM algorithms, while "NeRF" and "GS" represent SLAM algorithms based on Neural Radiance Fields and Gaussian Splatting.)

ATE (cm) ↓	ScanNet						BundleFusion				
	0054	0059	0106	0169	0233	0465	apt0	apt2	copyroom	office0	office2
ORB-SLAM3^{Tra}	243.26	90.67	178.13	60.15	25.01	181.86	89.38	148.04	19.70	31.41	73.91
DROID-SLAM^{DL}	161.22	67.26	11.20	17.39	69.85	116.42	87.37	265.64	27.59	116.33	49.32
NeRF-SLAM^{NeRF}	147.20	26.95	18.75	13.53	37.23	73.32	85.50	241.72	59.20	59.08	83.57
NICER-SLAM^{NeRF}	46.80	207.51	76.04	231.77	78.17	91.70	99.75	230.32	74.87	118.45	81.45
MonoGS^{GS}	70.189	97.24	150.89	191.98	62.45	113.19	122.59	142.54	53.41	62.67	127.02
PhotoSLAM^{GS}	332.03	205.01	359.85	151.61	195.71	294.20	247.19	320.91	54.03	271.87	298.98
Ours (VO)	44.08	15.96	16.13	16.84	60.71	92.83	44.22	136.69	39.10	44.44	39.10

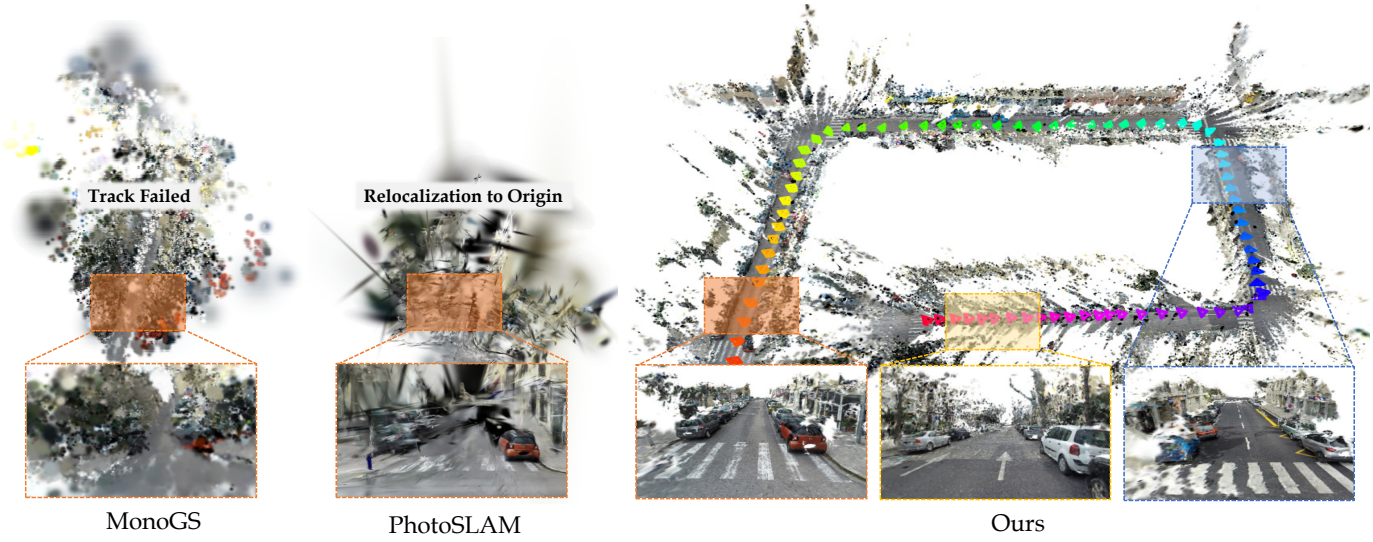


Fig. 6: VO Performance on SmallCity of Hierarchical [70]. MonoGS fails in tracking due to being obscured by large floaters, and Photoslam cannot match feature points to relocate to the starting point due to the lack of complex textures in and ego fast motion. In contrast, our method robustly and stably achieves localization and constructs high-quality Gaussian maps.

TABLE II: Monocular Localization results (ATE [m]) on the outdoor datasets Waymo and Hierarchical3DGS. "-" indicates that the system failed to track in this scenario, "*" indicates only the first 50 frames were tested due to tracking failure.

ATE [m] ↓	Waymo			Hierarchical3DGS	
	Scene01	Scene03	Scene14	SmallCity	Campus
ORB-SLAM3^{Tra}	1.21	2.49	2.48	-	-
DROID-SLAM^{DL}	2.38	2.94	3.98	5.83	1.87
NeRF-SLAM^{NeRF}	2.05	5.87	6.43	4.58	1.44
GO-SLAM^{NeRF}	3.15	3.07	5.13	5.79	3.50
MonoGS^{GS}	2.73	10.73	6.59	6.05*	20.81*
PhotoSLAM^{GS}	3.15	6.41	7.30	47.72*	34.04*
Ours (VO)	0.91	2.67	2.27	2.82	1.03

data and the 100Hz KITTI360 unsync data. The original KITTI unsync data had issues with out-of-order timestamps. Following the recommendations in the issue reports, we organized and corrected the data. Notably, ORB-SLAM3 failed to run successfully on KITTI's Mono&IMU configuration, so we recorded the results using the Stereo&10Hz IMU setup instead. As shown in Tab. III, for low-frequency IMU data, our method significantly outperforms feature-based SLAM algorithms and, in many scenarios, surpasses learning-based SLAM methods. Under the high-frequency IMU settings of the KITTI360 dataset, our method demonstrates a notably better localization performance compared to other approaches. Moreover, we are the first to propose a GS-based VIO odometry.

C. Rendering Performance

In this section, we compare the rendering performance of VINGS-Mono with both NeRF and 3DGS based methods.

1) Indoor Rendering Results: We evaluated the rendering quality of our method in comparison to two NeRF-based methods GO-SLAM [7], NICER-SLAM [16] and two state-of-the-art 3DGS-based monocular SLAM methods, MonoGS [5] and PhotoSLAM [6], using the ScanNet [64] and BundleFusion [65] datasets. Due to frequent issues with out-of-memory errors and not a number gradients encountered in NICER-SLAM, we only evaluated the metrics within the valid frames. For MonoGS, we initialized the process with 1000 iterations and performed 150 iterations for each subsequent frame. In the case of PhotoSLAM, due to its subpar performance under monocular settings (PSNR below 10), we used [74] predicted depth as a prior for the ScanNet dataset and recorded the

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

TABLE III: Visual inertial localization results (t_{rel} in $\%$ and r_{rel} in $^\circ/100m$) on KITTI and KITTI360.

	KITTI Sync										KITTI360 Unsync									
$t_{rel}\downarrow$ $r_{rel}\downarrow$	02		06		07		08		09		00		02		05		06		10	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
VINS-Mono^{Tra}	2.08	1.68	4.27	0.32	2.08	0.63	3.22	0.33	4.72	0.65	1.89	0.17	1.01	0.20	1.19	0.22	1.35	0.18	3.61	0.22
ORB-SLAM3^{Tra}	3.51	1.42	4.01	0.94	4.41	0.95	3.36	0.87	4.30	0.89	2.39	0.12	1.31	0.22	1.41	0.23	1.69	0.18	5.34	0.21
Selective-VIO^{DL}	2.41	0.78	1.90	0.52	1.72	1.01	2.23	0.91	2.83	0.80	-	-	-	-	-	-	-	-	-	-
iSLAM^{DL}	2.08	0.53	2.40	0.32	2.22	0.47	2.78	0.43	2.51	0.41	7.75	0.36	38.46	0.56	9.36	1.01	32.18	1.46	4.74	0.36
Ours (VIO)	2.64	0.44	2.01	0.40	1.01	0.80	1.90	0.23	2.84	0.38	0.76	0.10	0.58	0.17	1.16	0.23	0.73	0.16	4.23	0.42

TABLE IV: Depth Metric Results. We record the A.R.(absolute relative error) and δ_1 following DepthAnything [76] to evaluate the quality of rendered depths.

	Scan.-0059		Bundle.-apt0		Waymo-01		KITTI-02	
	A.R.↓	δ_1↑	A.R.↓	δ_1↑	A.R.↓	δ_1↑	A.R.↓	δ_1↑
GO-SLAM	0.241	0.740	0.335	0.401	0.504	0.404	0.714	0.214
MonoGS	0.274	0.709	0.271	0.674	0.406	0.378	0.727	0.186
Ours	0.154	0.855	0.125	0.898	0.236	0.538	0.107	0.844

rendered results, training for 100 iterations per image. Our method, designed for monocular settings, trained for just 80 iterations per image.

We present the qualitative and quantitative analysis of rendering quality for indoor scenes respectively. As illustrated in Figure 8, even though PhotoSLAM utilizes depth priors under RGB-D settings, our method still outperforms it across most scenes. During experiments, we observed that both PhotoSLAM and MonoGS frequently encounter issues where floaters cover the entire frame during later stages of tracking. Compared to the clone-and-split and reset-opacity-and-prune mechanisms used in PhotoSLAM and MonoGS which tend to be unstable under the SLAM setting where each frame is optimized for only a few dozen iterations, our method achieves more robust and reliable performance by simply removing Gaussians with large error score S_E values and reinitializing new ones based on depth. We used the poses estimated by each method as the viewpoints for image rendering and computed the average rendering quality for each scene over its trajectory sequence. Tab. VI shows that our method achieves the best quantitative performance on both ScanNet and BundleFusion datasets, with the highest SSIM (0.79), lowest LPIPS (0.22 on ScanNet and 0.29 on BundleFusion), and best PSNR (22.43 dB on ScanNet and 20.97 dB on BundleFusion). These results validate the comprehensive superiority of our approach in terms of PSNR, LPIPS and SSIM.

TABLE V: Localization results on several dynamic scene sequences in the BONN dataset [77].

ATE [cm] ↓	ball	ps_tk	ps_tk2	mv_box2	Avg.
ReFusion	17.5	28.9	46.3	17.9	27.65
RodynSLAM	7.9	14.5	13.8	12.6	12.2
Ours (wo Eraser)	11.75	37.48	48.31	23.44	30.25
Ours (w Eraser)	4.08	4.63	5.05	3.58	4.34

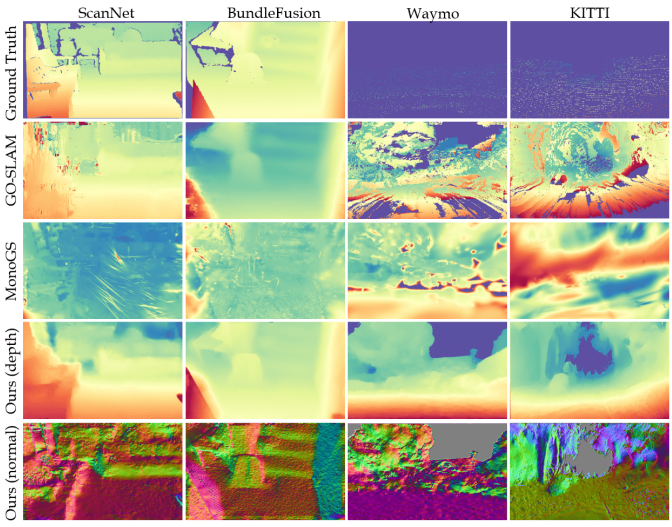


Fig. 7: Qualitive geometry results. Since PhotoSLAM does not support depth rendering, we compared it with GO-SLAM and MonoGS. (Red indicates nearby regions, while green denotes distant regions.)

2) Outdoor Rendering Results: Outdoor scenes present significantly greater challenges compared to indoor scenes due to longer trajectory lengths (ranging from hundreds to thousands of meters) and faster movement speeds. These factors considerably impact the rendering quality and map storage requirements of baseline methods. We conducted experiments on five datasets: KITTI [66], KITTI-360 [67], Waymo [68], Hierarchical [70], and MegaNeRF [69], which vary in lighting conditions, camera angles, and capturing devices. Due to the trajectories in KITTI and KITTI-360 often spanning several kilometers, no existing NeRF/3DGS-based SLAM method, apart from ours, has been successfully applied to them. To ensure fair comparisons, we trained the baseline methods on the first 500 frames, using the same settings as those for indoor experiments. Since the sky lacks depth, we used SegFormer to mask out the sky and computed the rendering quality metrics based on the masked results. Importantly, the same

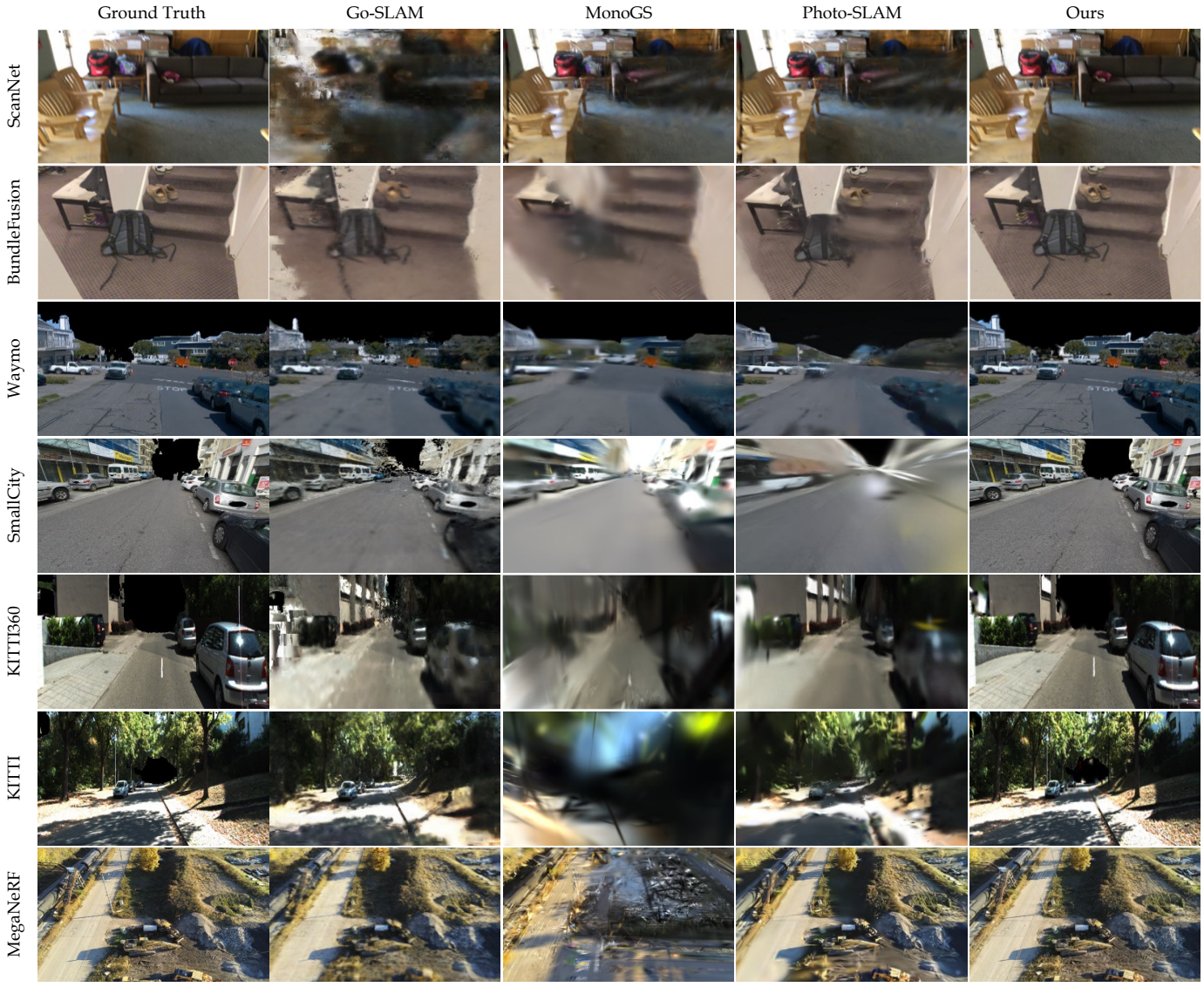


Fig. 8: Qualitative Rendering Results.We compared our method on two indoor [64], [65] and five outdoor scenes [66]–[70], with three advanced monocular SLAM algorithms, including the NeRF-based GO-SLAM [7] and two GS-based methods, MonoGS [5] and PhotoSLAM [6]. VINGS-Mono significantly outperforms existing methods in rendering quality.

filtered datasets were used for all methods during evaluation to maintain fairness. In addition, we also evaluated the quality of novel view synthesis on the KITTI stereo images, and the average PSNR only decreased by 3%.

As shown in Tab. VII and Fig. 8, our method excels in rendering high-quality details across large-scale environments and extended trajectories, even for long-distance, high-speed autonomous driving datasets such as KITTI, KITTI-360, and Waymo. For handheld datasets like Hierarchical, which are characterized by random motion trajectories and limited capture ranges, our approach achieves high-precision modeling of building edges and surface details. In drone datasets with significant depth variations and highly complex hierarchical scenes, our method demonstrates superior reconstruction quality, particularly in sparse regions. Across all scenarios, our system consistently achieves state-of-the-art (SOTA) performance in PSNR, SSIM, and LPIPS metrics, highlighting its robustness and versatility.

We visualized the Gaussian map generated by our method on KITTI-360 from both a bird's eye view and top-down view, we record the number of Gaussians on the GPU and CPU throughout the training process. As shown in Fig. 9, our method robustly adapts to large-scale urban scenes. Unlike existing GS-SLAM methods, which struggle with inevitable floaters, our approach, supported by our score manager, produces clean and accurate geometry even in tree-dense regions, as illustrated by the green dashed boxes in Fig. 9. Our method is capable of handling kilometer-scale scenes with 51.7 million Gaussians using a single RTX 4090 GPU.

In terms of map's global consistency. We demonstrated the impact of our loop closure on poses and the Gaussian map, as shown in Fig. 11. For large-scale scenes, our method can directly correct the Gaussian map without retraining, ensuring the construction of a globally consistent map.

Additionally, our method supports rendering color and depth from interpolated poses and exporting a mesh using TSDF-

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

TABLE VI: Quantitative results on the indoor datasets Scannet and BundleFusion. We mark the best two results with first and second. All quantitative metrics are computed as averages based on renderings at the same keyframes.

		ScanNet						BundleFusion				
		0054	0059	0106	0169	0233	0465	apt0	apt2	copyroom	office0	office2
GO-SLAM^{NeRF}	SSIM↑	0.59	0.32	0.47	0.42	0.48	0.09	0.52	0.34	0.61	0.23	0.51
	LPIPS↓	0.53	0.60	0.59	0.57	0.55	0.75	0.54	0.59	0.49	0.72	0.55
	PSNR↑	19.70	13.15	14.58	14.49	17.22	8.65	17.24	12.24	18.40	12.60	17.31
NICER-SLAM^{NeRF}	SSIM↑	0.79	0.67	0.73	0.75	0.68	0.67	0.69	0.59	0.66	0.64	0.61
	LPIPS↓	0.37	0.49	0.39	0.37	0.48	0.49	0.43	0.55	0.41	0.42	0.57
	PSNR↑	20.68	16.80	16.54	17.17	20.07	17.59	19.13	16.73	18.05	17.47	16.94
MonoGS^{GS}	SSIM↑	0.83	0.74	0.76	0.78	0.74	0.69	0.74	0.39	0.78	0.68	0.67
	LPIPS↓	0.61	0.59	0.60	0.61	0.67	0.74	0.62	0.82	0.57	0.68	0.67
	PSNR↑	21.37	18.55	17.58	19.15	19.73	17.19	18.80	11.50	17.83	16.76	18.98
PhotoSLAM^{GS}	SSIM↑	0.83	0.772	0.78	0.79	0.78	0.74	0.66	0.59	0.73	0.43	0.33
	LPIPS↓	0.35	0.41	0.37	0.39	0.37	0.45	0.56	0.60	0.36	0.63	0.68
	PSNR↑	20.54	17.17	16.09	17.46	23.95	19.88	11.46	11.68	16.96	9.21	8.55
Ours	SSIM↑	0.84	0.775	0.83	0.80	0.77	0.69	0.75	0.63	0.74	0.65	0.68
	LPIPS↓	0.20	0.24	0.18	0.22	0.22	0.25	0.28	0.41	0.33	0.39	0.23
	PSNR↑	26.31	20.51	23.10	22.27	23.67	21.27	20.45	18.61	18.47	19.85	22.23

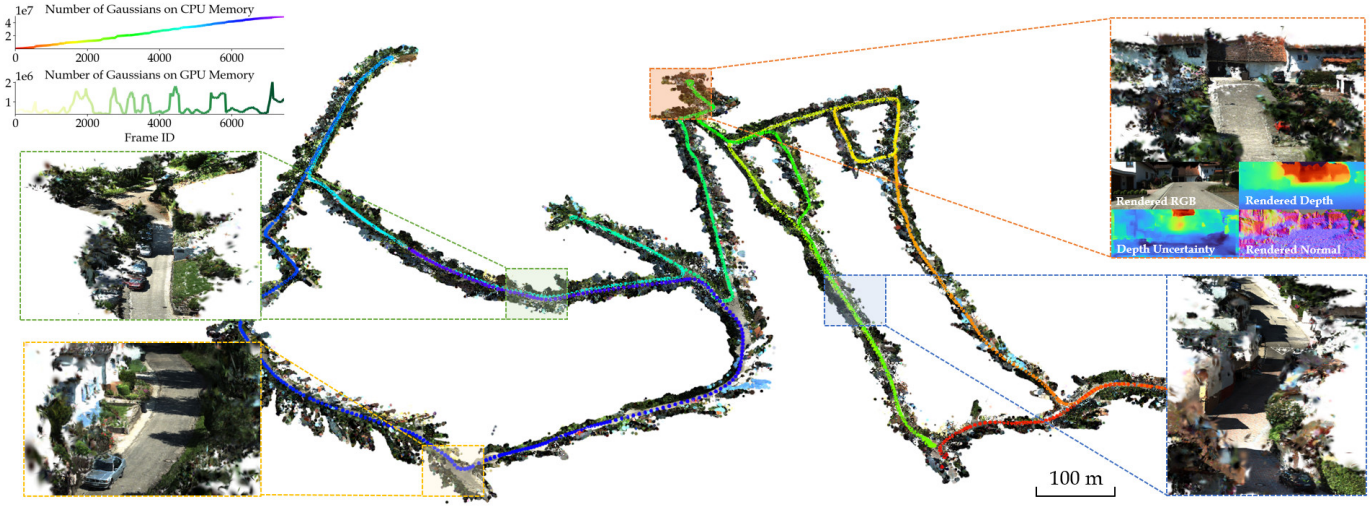


Fig. 9: Visualization of KITTI360's gaussian map. The trajectory length of scene 2013_05_28_drive_0006 is 8.05 km, and the entire Gaussian map contains 51.73 million ellipsoids. We recorded the number of Gaussians throughout the training process and zoomed in on different parts of the map for clearer visualization.

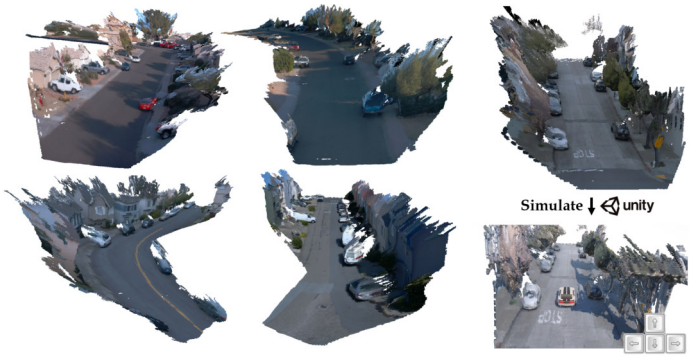


Fig. 10: Mesh Extraction of Waymo Dataset. We extract meshes based on rendered depths and load them into Unity for simulation to narrow the gap of simulation and reality .

Fusion [78]. This further expands the application scope of our approach. As shown in Fig. 10, we exported a mesh for the Waymo dataset and performed simulation in Unity.

3) Geometry Results: To validate the geometric accuracy of the map, we compared the rendered depth with ground truth depth or LiDAR data. Since monocular systems lack absolute scale information, we adopted the evaluation metrics from DepthAnything [76], namely the A.R. (absolute relative error) and δ_1 metrics. As Photo-SLAM does not support depth rendering, we conducted comparisons with GO-SLAM and MonoGS. As shown in the Table. IV, our approach significantly outperforms the baselines, thanks to the 2DGS representation. From the qualitative comparison in the Figure. 7, we can observe that 3DGS-based methods encounter floater occlusion issues in driving scenarios with forward views, which severely compromises geometric accuracy. Our method addresses this problem by removing Gaussians with large S_E and large projection radius when adding new keyframes.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

TABLE VII: Quantitative analysis results on the outdoor datasets KITTI, KITTI360, Waymo, Hierarchical, and MegaNeRF. "-" indicates that the system failed to track and render images in the whole scenario.

		KITTI			KITTI360			Waymo			Hierarchical		MegaNeRF	
		02	07	08	05	06	10	01	03	14	SmallCity	Campus	Building	Rubble
GO-SLAM^{NeRF}	SSIM↑	0.39	0.46	0.51	0.44	0.43	0.38	0.78	0.70	0.63	0.33	0.33	0.53	0.63
	LPIPS↓	0.49	0.45	0.43	0.47	0.45	0.20	0.20	0.30	0.34	0.57	0.54	0.40	0.32
	PSNR↑	15.01	12.81	14.62	14.27	14.24	21.07	21.07	21.22	19.54	14.30	13.41	20.71	20.81
MonoGS^{GS}	SSIM↑	0.34	0.43	0.52	0.53	0.55	0.20	0.83	0.74	0.82	-	0.52	0.23	0.24
	LPIPS↓	0.85	0.78	0.75	0.68	0.61	0.85	0.40	0.63	0.56	-	0.72	0.96	0.94
	PSNR↑	10.63	12.59	15.01	16.08	15.63	10.20	22.63	19.29	23.00	-	14.49	11.06	11.50
PhotoSLAM^{GS}	SSIM↑	0.44	0.52	0.48	0.51	0.56	0.51	0.74	0.69	0.76	0.39	0.57	0.31	0.27
	LPIPS↓	0.66	0.56	0.65	0.55	0.49	0.65	0.39	0.47	0.42	0.71	0.56	0.76	0.67
	PSNR↑	15.25	15.03	14.25	15.57	15.81	14.78	15.08	15.35	15.99	11.57	11.40	15.47	14.09
Ours	SSIM↑	0.68	0.73	0.79	0.80	0.80	0.82	0.85	0.86	0.85	0.81	0.78	0.82	0.82
	LPIPS↓	0.26	0.29	0.27	0.17	0.17	0.16	0.18	0.16	0.19	0.22	0.21	0.15	0.15
	PSNR↑	19.96	20.15	20.93	24.52	22.82	24.47	23.48	24.72	23.76	22.07	21.46	25.45	25.21

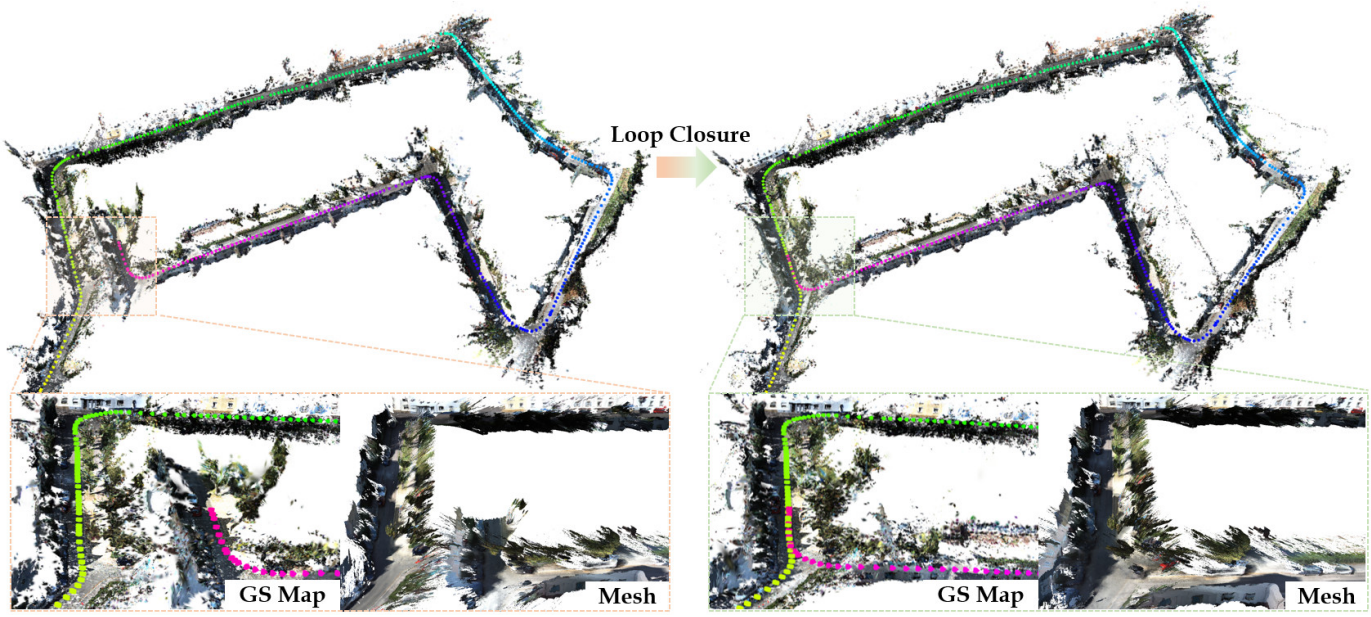


Fig. 11: Performance of NVS Loop Closure in urban scenes. Our NVS Loop Closure can correct the Gaussian map without time-consuming retraining. We zoomed in on the Gaussian map at the loop closure location and export the mesh, our method effectively ensuring the global consistency of the gaussian map.

TABLE VIII: Ablation on Score Manager.

S_C^{th}	ScanNet-0106		Waymo-Scene13	
	Avg. PSNR	GS Number	Avg. PSNR	GS Number
0.0 (wo)	22.98	4,041,325	23.67	1,777,807
0.8	22.58	3,104,080	23.60	1,376,648
12.8	23.07	2,675,419	23.47	1,321,745
25.6	23.13	2,265,721	23.16	1,308,828
102.4	23.02	1,964,771	22.47	1,059,930

D. Dynamic Eraser Performance

To evaluate the effectiveness of Dynamic Eraser in enhancing tracking performance in dynamic environments, we masked out dynamic objects in the frontend BA using Dynamic Eraser. We selected the dynamic SLAM dataset, BONN Dataset [77], and conducted comparative experiments with two SLAM methods tailored for dynamic scenes, as shown in Tab. V. Note that, since the baseline methods require depth information, all results presented in the table are based on RGB-D input. Compared with ReFusion [79] and RodynSLAM [80], our approach achieves superior results. Additionally, we performed ablation studies, and the results demonstrate that for datasets where dynamic objects dominate the scene, Dynamic Eraser significantly improves tracking accuracy. Our dynamic object eraser is a plug-and-play module and remains active in BONN Dataset and real world experiments.

E. Ablation Studies

1) Ablation on Sample Rasterizer: We selected KITTI dataset to profile the rendering during forward propagation and backpropagation for each training iteration. As shown in Fig. 12, we recorded the number of Gaussians, PSNR values, and the time taken for forward and backward propagation.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

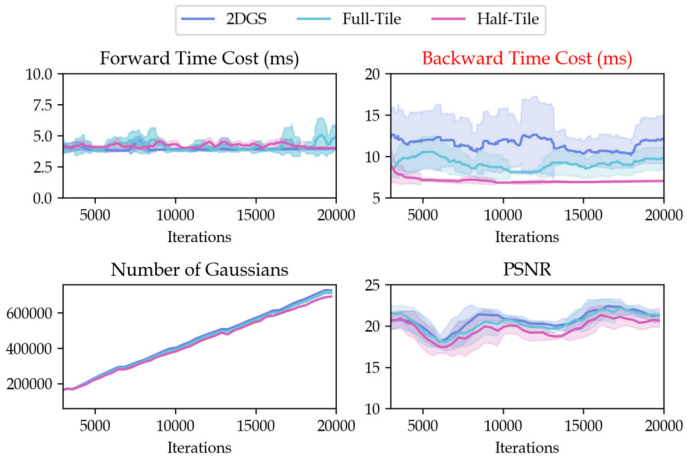


Fig. 12: Ablation on sample rasterizer.

We compared our approach with the original 2DGS pixel-parallel method and the Gaussian-parallel method used in Taming3DGS. Since our acceleration strategy does not affect the score manager, the number of Gaussians showed no significant change. In terms of training speed, compared to performing backpropagation on all pixels within a tile (Full-Tile), our sample-based backpropagation strategy achieved a 170% acceleration in backpropagation time (from 7.21 ms to 4.23 ms), despite a 0.56 drop in average PSNR. Compared to the original 2DGS approach, our method accelerated back-propagation by 273% (from 11.55 ms to 4.23 ms).

2) Ablation on Score Manager: The primary function of our score manager is to remove unnecessary Gaussians and facilitate data transfer between the CPU and GPU. This is achieved by reducing the number of Gaussians while minimizing any impact on rendering quality. To evaluate its effectiveness, we conducted ablation studies on one indoor and one outdoor scenes by changing S_c^{th}, the delete threshold of S_C. As shown in Tab. VIII, our score manager maintains the rendering quality while significantly reducing the number of the Gaussians. For outdoor scene, it achieves a 22% reduction in Gaussian ellipsoids while maintaining a PSNR decrease of only 0.3%.

3) Ablation on Pose Refinement: Our method binds Gaussians to different keyframe poses, the proposed pose refinement strategy enables rendering a frame while simultaneously optimizing the poses of all visible keyframes. Existing GS-based SLAM [2], [5] methods typically optimize the pose of the current frame using rerendering losses. To validate the effectiveness of our pose refinement strategy, we conducted ablation studies, as shown in Tab. IX. Our method outperforms the single-frame optimization strategy in both indoor and outdoor scenes. As shown in the Tab IX, our approach achieves significant improvements in scenes where the frontend tracking performance was initially poor, consistently outperforming existing single-frame optimization methods.

F. Runtime Analysis

We report the runtime and model size on three datasets with varying frame counts: Waymo, Hierarchical, and KITTI.

TABLE IX: Ablation on Pose Refinement.

ATE [m]↓	ScanNet-0106	Copyroom	Campus
wo pose refine	0.25	0.83	1.83
w refine current pose ([2], [5])	0.19	0.64	1.19
w refine visible poses (ours)	0.16	0.39	1.03

TABLE X: Runtime Analysis.

Dataset-Scene	Frame Number	Total Runtime	Tracking /Frame	FPS	Model Size
Waymo-Scene01	198	117s	214ms	1.69	386Mb
Hiera.-SmallCity	877	739s	247ms	1.18	1817Mb
KITTI-Odom08	5177	4560s	273ms	1.13	10366Mb

Our VIO Frontend and Mapping modules run as two parallel threads, with the mapping speed being slower than tracking. First, we independently tested the runtime of the tracking module. Then, we disabled the visualization of the BEV (bird's-eye-view) map to measure the overall runtime of our framework. The Model Size refers to the file size of the final Gaussian point cloud that is saved, which differs from the GPU memory usage during runtime. All results were profiled using an RTX 4090 GPU, as shown in Tab. X. Our method demonstrates the capability of running online for both shorter trajectories (e.g., around 300 meters, as in the Waymo dataset) and longer trajectories (3.2km, as in the KITTI dataset).

G. Real world Experiments

1) Large-scale Environment: To test the stability and robustness of our method, we collected a large outdoor dataset. This dataset covers our entire Campus and was recorded using an iPhone device. It spans approximately 1.02 km in length and 0.4 km in width. The data collection was conducted on a bike, riding at a speed of around 10 km/h. The dataset contains only 20Hz RGB image data (1280×720) and 1Hz GPS data. We validated the performance of our algorithm in large-scale scenarios under monocular settings. As shown in Fig. 13, we visualized the GPS data in the top-right corner of the figure. It can be observed that even with only monocular input data, our method exhibited almost no scale drift in large-scale scenarios during long-duration tests.

2) Mobile App on Smartphone: We ran VINGS-Mono in a mobile phone setup. We developed an mobile app which collects images with a resolution of 480×720 at 30 Hz, along with IMU data of our phone. This data is then transferred to our server. On the server, VINGS-Mono processes the data and generates real-time visual outputs displayed on the phone screen. These outputs include a bird's-eye view Gaussian map, the captured images, rendered images, rendered depth maps, and rendered normal maps. To achieve better geometry in real-world scenarios, we utilize [74] to provide depth priors. It is important to note that this depth information is noisy and exhibits scale inconsistencies across multiple views. To evaluate the robustness of VINGS-Mono, we conducted experiments in both indoor and outdoor scenes under various lighting conditions. The experimenter held the phone and walked through our lab and around the square. As shown

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

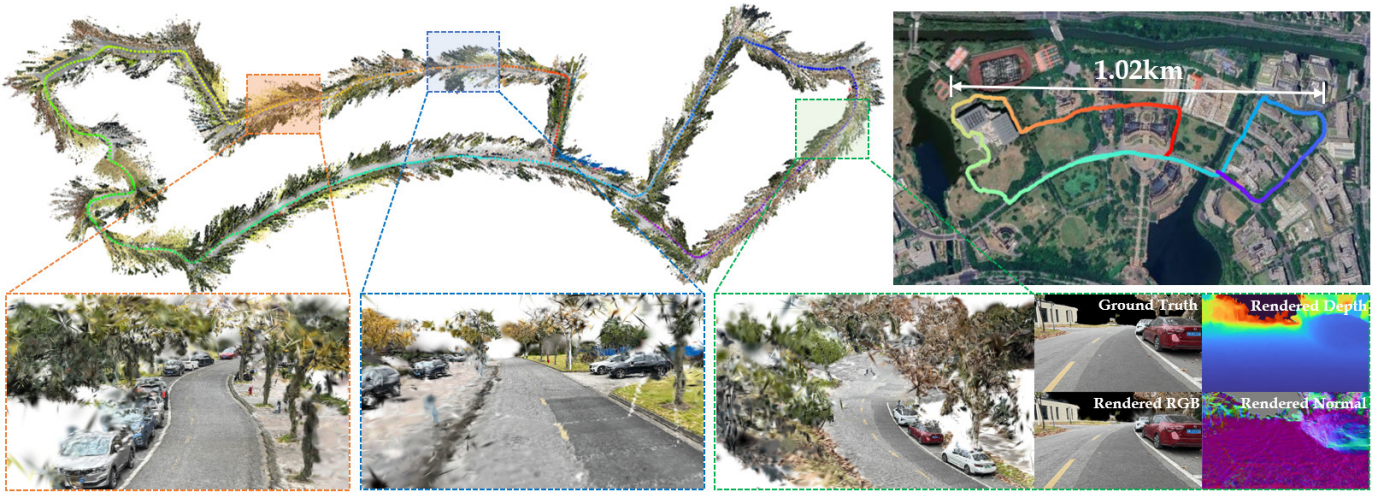


Fig. 13: Monocular SLAM results of large scale self-collected data. The collected data covers our campus. The trajectory and map on the left represent the results estimated by VINGS-Mono, while the top-right shows the smartphone GPS data recorded during data collection, aligned with Google Map.

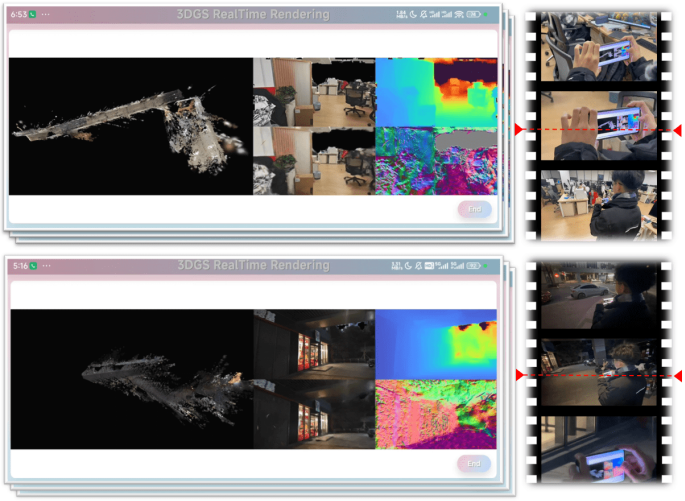


Fig. 14: Mobile App of VINGS-Mono.

in Fig. 14, our method demonstrates strong performance in reconstructing low-texture regions such as white walls. Additionally, in outdoor scenes under low-light conditions, our method achieves high-quality reconstruction of highly exposed areas, such as illuminated signboards.

IX. Conclusions and Future Work

A. Conclusions

In this paper, we propose VINGS-Mono, a monocular inertial Gaussian Splatting SLAM framework for large-scale environments. It introduces a score manager for efficient Gaussian pruning, a single-to-multi pose refinement module to improve tracking, a loop closure with novel view synthesis for global consistency, and dynamic object masking to handle transient objects, achieving efficient and accurate SLAM performance.

We evaluate VINGS-Mono through experiments on indoor and outdoor datasets, real-world large-scale tests, and ablation studies. Results show superior localization and mapping compared to state-of-the-art NeRF/GS and visual SLAM methods. A mobile app demonstrates real-time mapping in both indoor and outdoor scenarios. By leveraging 2DGS, VINGS-Mono reconstructs dense geometry and color without LiDAR or depth cameras, preserving critical scene details for navigation and enabling tasks like image-goal navigation. Its Gaussian maps with novel view rendering also support VR/AR and digital twin applications, enhancing scalability and adaptability.

B. Limitations and Future Works

A key limitation of our work lies in its inability to effectively handle extremely high-speed motion. Specifically, DBA faces challenges in recovering dense geometric information when frame intervals are large, while the multiple training iterations required by GS limit the reconstruction speed of the 2D Gaussian map. To address this, our future work will focus on integrating additional priors into DBA and incorporating networks such as [81], [82] to directly output Gaussian attributes, thereby reducing the number of training iterations. For loop detection, judgments based on rendering loss are sensitive to large illumination changes. We find that using DINO [39] features for further matching may help address this issue. For loop correction, we can only retrain the Gaussians on the GPU and adjust the position and rotation attributes of the Gaussians on the CPU, which limits the rendering quality of the historical map. Another limitation is that our system has not yet been deployed for on-device computation. In future work, we will explore deploying VINGS-Mono directly onto edge computing devices [83] to further enhance the practical value and applicability of our algorithm. [84]

References

[1] B. Kerbl, G. Kopanas, T. Leimkühler, G. Drettakis, 3d gaussian splatting for real-time radiance field rendering, ACM Transactions on Graphics 42 (4) (2023).

[2] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, J. Luiten, Splatam: Splat, track map 3d gaussians for dense rgb-d slam, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.

[3] Z. Peng, T. Shao, L. Yong, J. Zhou, Y. Yang, J. Wang, K. Zhou, Rtg-slam: Real-time 3d reconstruction at scale using gaussian splatting (2024).

[4] S. Hong, J. He, X. Zheng, C. Zheng, S. Shen, Liv-gaussmap: Lidar-inertial-visual fusion for real-time 3d radiance field map rendering, IEEE Robotics and Automation Letters (2024).

[5] H. Matsuki, R. Murai, P. H. J. Kelly, A. J. Davison, Gaussian Splatting SLAM, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.

[6] H. Huang, L. Li, C. Hui, S.-K. Yeung, Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.

[7] Y. Zhang, F. Tosi, S. Mattoccia, M. Poggi, Go-slam: Global optimization for consistent 3d instant reconstruction, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2023.

[8] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng, Nerf: Representing scenes as neural radiance fields for view synthesis, Communications of the ACM 65 (1) (2021) 99–106.

[9] F. Tosi, Y. Zhang, Z. Gong, E. Sandström, S. Mattoccia, M. R. Oswald, M. Poggi, How nerfs and 3d gaussian splatting are reshaping slam: a survey, arXiv preprint arXiv:2402.13255 4 (2024) 1.

[10] E. Sucar, S. Liu, J. Ortiz, A. J. Davison, imap: Implicit mapping and positioning in real-time, in: 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 6209–6218. doi:10.1109/ICCV48922.2021.00617.

[11] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, M. Pollefeys, Nice-slam: Neural implicit scalable encoding for slam, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

[12] E. Sandström, Y. Li, L. Van Gool, M. R. Oswald, Point-slam: Dense neural point cloud-based slam, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2023.

[13] Y. Mao, X. Yu, Z. Zhang, K. Wang, Y. Wang, R. Xiong, Y. Liao, Ngel-slam: Neural implicit representation-based global consistent low-latency slam system, in: 2024 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2024, pp. 6952–6958.

[14] C. Campos, R. Elvira, J. J. Gomez, J. M. M. Montiel, J. D. Tardós, ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM, IEEE Transactions on Robotics 37 (6) (2021) 1874–1890.

[15] T. Deng, G. Shen, T. Qin, J. Wang, W. Zhao, J. Wang, D. Wang, W. Chen, Plgslam: Progressive neural scene represenation with local to global bundle adjustment, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 19657–19666.

[16] Z. Zhu, S. Peng, V. Larsson, Z. Cui, M. R. Oswald, A. Geiger, M. Pollefeys, Nicer-slam: Neural implicit scene encoding for rgb slam, in: 2024 International Conference on 3D Vision (3DV), IEEE, 2024, pp. 42–52.

[17] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, A. Fitzgibbon, Kinectfusion: Real-time dense surface mapping and tracking, in: 2011 10th IEEE International Symposium on Mixed and Augmented Reality, 2011, pp. 127–136. doi:10.1109/ISMAR.2011.6092378.

[18] T. Whelan, R. Salas-Moreno, B. Glocker, A. Davison, S. Leutenegger, Elasticfusion: Real-time dense slam and light source estimation, The International Journal of Robotics Research 35 (09 2016). doi:10.1177/0278364916669237.

[19] T. Qin, P. Li, S. Shen, Vins-mono: A robust and versatile monocular visual-inertial state estimator, IEEE Transactions on Robotics 34 (4) (2018) 1004–1020. doi:10.1109/TRO.2018.2853729.

[20] V. Yugay, Y. Li, T. Gevers, M. R. Oswald, Gaussian-slam: Photo-realistic dense slam with gaussian splatting, arXiv preprint arXiv:2312.10070 (2023).

[21] S. Ha, J. Yeon, H. Yu, Rgbd gs-icp slam (2024). arXiv:2403.12550. URL https://arxiv.org/abs/2403.12550

[22] P. Zhu, Y. Zhuang, B. Chen, L. Li, C. Wu, Z. Liu, Mgs-slam: Monocular sparse tracking and gaussian mapping with depth smooth regularization, arXiv preprint arXiv:2405.06241 (2024).

[23] Z. Teed, L. Lipson, J. Deng, Deep patch visual odometry, Advances in Neural Information Processing Systems 36 (2024).

[24] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of rgb-d slam systems, in: Proc. of the International Conference on Intelligent Robot Systems (IROS), 2012.

[25] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, et al., The replica dataset: A digital replica of indoor spaces, arXiv preprint arXiv:1906.05797 (2019).

[26] A. Oliva, A. Torralba, Modeling the shape of the scene: A holistic representation of the spatial envelope, International journal of computer vision 42 (2001) 145–175.

[27] M. Calonder, V. Lepetit, C. Strecha, P. Fua, Brief: Binary robust independent elementary features, in: Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11, Springer, 2010, pp. 778–792.

[28] C. McManus, B. Upcroft, P. Newman, Scene signatures: Localised and point-less features for localisation, Robotics: Science and Systems X (2014) 1–9.

[29] D. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision 60 (2004) 91–110.

[30] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, Orb: An efficient alternative to sift or surf, in: 2011 International conference on computer vision, Ieee, 2011, pp. 2564–2571.

[31] M. Agrawal, K. Konolige, M. R. Blas, Censure: Center surround extremas for realtime feature detection and matching, in: European conference on computer vision, Springer, 2008, pp. 102–115.

[32] J. Sivic, A. Zisserman, Video google: a text retrieval approach to object matching in videos, Proceedings Ninth IEEE International Conference on Computer Vision (2003) 1470–1477 vol.2. URL https://api.semanticscholar.org/CorpusID:14457153

[33] D. Nistér, H. Stewénius, Scalable recognition with a vocabulary tree, 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) 2 (2006) 2161–2168. URL https://api.semanticscholar.org/CorpusID:1654266

[34] Z. Chen, O. Lam, A. Jacobson, M. Milford, Convolutional neural network-based place recognition, arXiv preprint arXiv:1411.1509 (2014).

[35] R. Arandjelovic, P. Gronát, A. Torii, T. Pajdla, J. Sivic, Netvlad: CNN architecture for weakly supervised place recognition, CoRR abs/1511.07247 (2015). arXiv:1511.07247. URL http://arxiv.org/abs/1511.07247

[36] L. Zhu, Y. Li, E. Sandström, S. Huang, K. Schindler, I. Armeni, Loopsplat: Loop closure by registering 3d gaussian splats, ArXiv abs/2408.10154 (2024). URL https://api.semanticscholar.org/CorpusID:271903921

[37] P.-E. Sarlin, C. Cadena, R. Siegwart, M. Dymczyk, From coarse to fine: Robust hierarchical localization at large scale, in: CVPR, 2019.

[38] S. Izquierdo, J. Civera, Optimal transport aggregation for visual place recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 17658–17668.

[39] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al., Dinov2: Learning robust visual features without supervision, arXiv preprint arXiv:2304.07193 (2023).

[40] L. Liso, E. Sandström, V. Yugay, L. Van Gool, M. R. Oswald, Loopy-slam: Dense neural slam with loop closures, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 20363–20373.

[41] J. Engel, T. Schöps, D. Cremers, Lsd-slam: Large-scale direct monocular slam, in: European conference on computer vision, Springer, 2014, pp. 834–849.

[42] M. Yang, Y. Chen, H.-S. Kim, Efficient deep visual and inertial odometry with adaptive visual modality selection, in: European Conference on Computer Vision, Springer, 2022, pp. 233–250.

[43] T. Fu, S. Su, Y. Lu, C. Wang, islam: Imperative slam, IEEE Robotics and Automation Letters (2024).

[44] H. Matsuki, K. Tateno, M. Niemeyer, F. Tombari, Newton: Neural view-centric mapping for on-the-fly large-scale slam, IEEE Robotics and Automation Letters (2024).

[45] C. Wu, Y. Duan, X. Zhang, Y. Sheng, J. Ji, Y. Zhang, Mm-gaussian: 3d gaussian-based multi-modal fusion for localization and reconstruction in unbounded scenes, arXiv preprint arXiv:2404.04026 (2024).

[46] Z. Teed, J. Deng, Raft: Recurrent all-pairs field transforms for optical flow, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, Springer, 2020, pp. 402–419.

[47] Z. Teed, J. Deng, Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras, in: M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, J. W. Vaughan (Eds.), Advances in Neural Information Processing Systems, Vol. 34, Curran Associates, Inc., 2021, pp. 16558–16569.
URL https://proceedings.neurips.cc/paper_files/paper/2021/file/89fcd07f20b6785b92134bd6c1d0fa42-Paper.pdf

[48] Y. Zhou, X. Li, S. Li, X. Wang, S. Feng, Y. Tan, Dba-fusion: Tightly integrating deep dense visual bundle adjustment with multiple sensors for large-scale localization and mapping, IEEE Robotics and Automation Letters (2024).

[49] F. Dellaert, Factor graphs and gtsam: A hands-on introduction, Georgia Institute of Technology, Tech. Rep 2 (2012) 4.

[50] A. Rosinol, J. J. Leonard, L. Carlone, Probabilistic volumetric fusion for dense monocular slam, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023, pp. 3097–3105.

[51] C. Forster, L. Carlone, F. Dellaert, D. Scaramuzza, On-manifold preintegration for real-time visual–inertial odometry, IEEE Transactions on Robotics 33 (1) (2016) 1–21.

[52] B. Huang, Z. Yu, A. Chen, A. Geiger, S. Gao, 2d gaussian splatting for geometrically accurate radiance fields, in: ACM SIGGRAPH 2024 conference papers, 2024, pp. 1–11.

[53] V. Ye, A. Kanazawa, Mathematical supplement for the gsplat library, arXiv preprint arXiv:2312.02121 7 (2023).

[54] D. B. Kirk, W. H. Wen-Mei, Programming massively parallel processors: a hands-on approach, Morgan kaufmann, 2016.

[55] S. S. Mallick, R. Goel, B. Kerbl, M. Steinberger, F. V. Carrasco, F. De La Torre, Taming 3dgs: High-quality radiance fields with limited resources, in: SIGGRAPH Asia 2024 Conference Papers, 2024, pp. 1–11.

[56] M. Fang, J. Fang, W. Zhang, H. Zhou, J. Liao, Y. Wang, Benchmarking the gpu memory at the warp level, Parallel Computing 71 (2018) 23–41.

[57] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, X. Li, Gs-slam: Dense visual slam with 3d gaussian splatting, in: CVPR, 2024.

[58] P. Lindenberger, P.-E. Sarlin, M. Pollefeys, Lightglue: Local feature matching at light speed, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 17627–17638.

[59] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian, X. Wang, 4d gaussian splatting for real-time dynamic scene rendering, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 20310–20320.

[60] J. Luiten, G. Kopanas, B. Leibe, D. Ramanan, Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis, in: 2024 International Conference on 3D Vision (3DV), IEEE, 2024, pp. 800–809.

[61] Z. Yang, H. Yang, Z. Pan, L. Zhang, Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting, arXiv preprint arXiv:2310.10642 (2023).

[62] Y. Chen, C. Gu, J. Jiang, X. Zhu, L. Zhang, Periodic vibration gaussian: Dynamic urban scene reconstruction and real-time rendering, arXiv preprint arXiv:2311.18561 (2023).

[63] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, J. Wang, Fast segment anything, arXiv preprint arXiv:2306.12156 (2023).

[64] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, M. Nießner, Scannet: Richly-annotated 3d reconstructions of indoor scenes, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5828–5839.

[65] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, C. Theobalt, Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration, ACM Transactions on Graphics (ToG) 36 (4) (2017) 1.

[66] A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite, 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012) 3354–3361.
URL https://api.semanticscholar.org/CorpusID:6724907

[67] Y. Liao, J. Xie, A. Geiger, Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d, IEEE Transactions on Pattern Analysis and Machine Intelligence 45 (3) (2022) 3292–3310.

[68] Waymo, Waymo autonomous driving system overview (2021).
URL https://waymo.com/technology/

[69] H. Turki, D. Ramanan, M. Satyanarayanan, Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12922–12931.

[70] B. Kerbl, A. Meuleman, G. Kopanas, M. Wimmer, A. Lanvin, G. Drettakis, A hierarchical 3d gaussian representation for real-time rendering of very large datasets, ACM Transactions on Graphics (TOG) 43 (4) (2024) 1–15.

[71] J. L. Schönberger, E. Zheng, J.-M. Frahm, M. Pollefeys, Pixelwise view selection for unstructured multi-view stereo, in: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14, Springer, 2016, pp. 501–518.

[72] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of rgb-d slam systems, in: 2012 IEEE/RSJ international conference on intelligent robots and systems, IEEE, 2012, pp. 573–580.

[73] M. Grupp, evo: Python package for the evaluation of odometry and slam., https://github.com/MichaelGrupp/evo (2017).

[74] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, C. Shen, Metric3d: Towards zero-shot metric 3d prediction from a single image, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 9043–9053.

[75] A. Rosinol, J. J. Leonard, L. Carlone, Nerf-slam: Real-time dense monocular slam with neural radiance fields, in: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2023, pp. 3437–3444.

[76] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, H. Zhao, Depth anything: Unleashing the power of large-scale unlabeled data, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 10371–10381.

[77] E. Palazzolo, J. Behley, P. Lottes, P. Giguère, C. Stachniss, ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals, in: iros, 2019.
URL https://www.ipb.uni-bonn.de/pdfs/palazzolo2019iros.pdf

[78] B. Curless, M. Levoy, A volumetric method for building complex models from range images, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996, pp. 303–312.

[79] Z. Luo, F. K. Gustafsson, Z. Zhao, J. Sjölund, T. B. Schön, Refusion: Enabling large-size realistic image restoration with latent-space diffusion models, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2023, pp. 1680–1691.

[80] H. Jiang, Y. Xu, K. Li, J. Feng, L. Zhang, Rodyn-slam: Robust dynamic dense rgb-d slam with neural radiance fields, IEEE Robotics and Automation Letters (2024).

[81] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, H. Zhao, Point transformer v3: Simpler faster stronger, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 4840–4851.

[82] Z. Fan, J. Zhang, W. Cong, P. Wang, R. Li, K. Wen, S. Zhou, A. Kadambi, Z. Wang, D. Xu, et al., Large spatial model: End-to-end unposed images to semantic 3d, arXiv preprint arXiv:2410.18956 (2024).

[83] Q. Hou, R. Rauwendaal, Z. Li, H. Le, F. Farhadzadeh, F. Porikli, A. Bourd, A. Said, Sort-free gaussian splatting via weighted sum rendering, arXiv preprint arXiv:2410.18931 (2024).

[84] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, C. Theobalt, Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration, ACM Transactions on Graphics (ToG) 36 (4) (2017) 1.