CIMAT Centro de Investigación en Matemáticas, A.C.

2D Gaussian Splatting-based camera localization

T E S I S

Que para obtener el grado de

Maestro en Ciencias Computacionales y Matemáticas Industriales

Presenta

Sergio Alberto De León Martínez

Director de Tesis:

Dr. Jean-Bernard Hayet

Autorización de la versión final

Guanajuato, Guanajuato, Julio 2025

Dedico este trabajo a mis pilares, mi familia

Summary

Simultaneous Localization and Mapping (SLAM) is a core problem in computer vision and robotics: an agent must build a map of an environment while estimating its own pose within that map. Visual SLAM is the special case where the only sensor is a camera. Recent progress in hardware and learning has renewed interest in map representations that are compact, perform in real-time, unify tracking and mapping, and—crucially for applications such as augmented reality—enable photorealistic rendering. This thesis investigates Gaussian Splatting (GS), a recent view-synthesis technique, as the map representation for a visual SLAM system. We adopt a modified variant of GS that addresses several limitations of the original formulation and integrate it into our pipeline. We conduct extensive experiments on the TUM-RGBD and Replica datasets, report quantitative and qualitative results, and analyze strengths and failure modes. While the approach provides high-fidelity maps and supports real-time operation, it remains challenged by dynamic scenes, which we discuss as a key avenue for future work.

keywords Visual SLAM, Gaussian Splatting, real-time, photorealistic rendering.

Spanish version: La Localización y Mapeo Simultáneos (SLAM) es un problema central en visión por computadora y robótica: un agente debe construir un mapa del entorno mientras estima su propia pose (posición y orientación) dentro de ese mapa. El SLAM visual es el caso particular en el que la cámara es la única fuente de información. Los avances recientes en hardware y en aprendizaje han renovado el interés por representaciones de mapa que sean compactas, operen en tiempo real, unifiquen el seguimiento y el mapeo y—crucialmente para aplicaciones como la realidad aumentada—permitan un renderizado fotorrealista. Esta tesis investiga Gaussian Splatting (GS), una técnica reciente de síntesis de vistas, como representación de mapa para un sistema de SLAM visual. Adoptamos una variante modificada de GS que aborda varias limitaciones de la formulación original y la integramos en nuestro pipeline. Realizamos experimentos extensivos en los conjuntos de datos TUM-RGBD y Replica, reportamos resultados cuantitativos y cualitativos, y analizamos fortalezas y modos de fallo. Si bien el enfoque produce mapas de alta fidelidad y admite operación en tiempo real, sigue enfrentando desafíos en escenas dinámicas, los cuales discutimos como una vía clave para trabajos futuros.

Acknowledgements

I would first like to express my deepest gratitude to my parents Sergio and Ma. Guadalupe; their unwavering love and support made all of this possible. I am profoundly grateful to my advisor Jean-Bernard Hayet, whose patience, guidance, and trust in my work were invaluable. My advisor's mentorship helped me develop a rigorous methodology and grow toward independent research. I also thank my family—especially my sisters Brenda and Elena—for always being there when I needed them, and all my friends, in particular to Jesús, Juan and Kapioma for making this journey lighter and for being an essential part of my life over the past two years.

I am indebted to the members of the review committee for their thoughtful feedback and constructive comments, which significantly improved this thesis. Finally, I gratefully acknowledge the support of *SECIHTI* and CIMAT for providing the material and intellectual resources that made this work possible.

Contents

List of Figures

Chapter 1

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a classic problem in robotics and computer vision, taking its roots in the early research on autonomous robots in the 1980's; it aims to recover the positions of $3D$ landmarks (i.e., the scene structure, or scene map) of a scene and the robot positions and orientations (motion/trajectory) within this map, all of this from sensor data; an important characteristic is that it needs to operate as close to real-time as possible, because the idea is to do all of this *on the fly*, typically for an autonomous agent to be able to take decisions about its motion in the world. It is important to mention that we need to solve both problems (localization and mapping) synchronously because the two are highly coupled: Being badly localized implies that introducing new features in the map will bring high uncertainty in this map; and errors on the map lead to errors in the localization. In computer vision, the *off-line* version of this problem is called Structure-from-Motion (SfM), and the main difference with this last technique is that SLAM operates in an *incremental* way, updating its estimates about the map and the sensor pose with each new sensor measurement. There exist two types of sensors used in the SLAM context, the proprioceptive sensors like inertial measurement units (IMU's) and the exteroceptive sensors like cameras, light detection and ranging (LiDAR's) and radars. In this thesis, we focus on using the second kind. Probably the most challenging

scenario is when the sensor measurement comes from a single camera, since in this case we do not have information about the *depth* of the elements within the scene, hence we cannot associate directly observations (in the images) to map elements. We refer to this case as visual SLAM and it is the main topic of this thesis work.

1.1 Document organization

This thesis is organized as follows. In Chapter 1, we present the motivation behind the problem, provide an overview of modern SLAM architectures, and review relevant previous work. Chapter 2 introduces a recent novel view-synthesis technique known as Gaussian Splatting. We describe the method in detail, highlight its limitations, and discuss how to address these shortcomings. An improved version (known as 2D Gaussian Splatting) then serves as the foundation for building our visual SLAM system. Chapter 3 focuses on the proposed methodology, detailing the two main modules—tracking and mapping—and explaining how they interact to form a unified SLAM system that employs a single representation for both tasks. In Chapter 4, we evaluate the method on the Replica Straub et al. (2019) and TUM RGB-D Sturm, Engelhard, Endres, Burgard, and Cremers (2012) datasets and provide a discussion about the results. Finally, Chapter 5 summarizes the contributions of this work, presents the main conclusions, and outlines promising directions for future research.

1.2 Motivation

For a robot to accurately execute an assigned task, it is convenient (although *not necessary*) to have a representation of the physical space where it is inserted, i.e. a *map*; at the same time, the robot estimates its location within this representation and can use it for *planning*. If the world map is known, then SLAM reduces to the task of *localization*. On the other hand, if the camera trajectory is known, for instance, when an absolute positioning system is used, like differential GPS or motion capture, then SLAM only requires to perform the task of *mapping*. Most of the time, this

last case is not applicable because it is expensive and limited; hence arises the need of SLAM algorithms to perform accurate camera trajectory estimates and mapping of the scene. Both tasks of *localization* and *mapping* taken individually are rather "simple" to solve, but the combination of the two is more difficult.

There are many applications of SLAM in real-world problems, and let us mention some of them. We can find one of the most popular applications in autonomous driving LLC (2020), where it is crucial to have a world map to make decisions in a very dynamic and complex environment. In industry, and in particular in warehouse contexts enVista Corporation (2025), the use of Autonomous Mobile Robots allows to handle mechanical and repetitive tasks, and in order to operate safely and efficiently they need to have integrated SLAM algorithms. Lastly, perhaps the application most related to this work is augmented reality Meta (2019), where a portable device must be endowed with *mapping* and *localization* capabilities in order to perform high-level tasks like place recognition or $3D$ perception and *superimpose virtual elements* in a consistent way. All in all, SLAM systems are an essential component to achieve autonomous agents that can perceive the world as humans do.

1.3 SLAM modern architecture

As mentioned above, mapping and tracking are highly coupled. If we have the camera poses, then we can retrieve at least a sparse set of $3D$ landmarks represented, for example, as a point cloud (through a process called *triangulation*); the other way around is true, if we have the $3D$ landmarks, then we can set $2D-2D$, $3D-2D$ or $3D-3D$ associations and recover camera poses (a problem known as camera *pose estimation*). The success of a SLAM system lies in achieving the mapping and tracking processes *incrementally and simultaneously*. The modern approach to solve this problem has been introduced in the seminal work of Klein and Murray (2007). It consists in splitting the chicken-and-egg SLAM problem in two *threads* that run in parallel:

- the *frontend* is responsible for receiving the raw sensor measurements and for extracting initial guesses for the camera trajectory and the map; it is a *fast* process, focused on the localization task, that provides the last available camera pose;

- the *backend* is responsible for refining those initial guesses and for ensuring global or local consistency; it is a *slow* process (with a heavy optimization problem involving a lot of variables), essentially focused on the hardest task, mapping.

Since then, many approaches have been proposed in this field, but we can mainly distinguish between two research directions, which are *indirect* and *direct* methods.

For *indirect* methods, the frontend extracts intermediate representations of the input frames, in the form of key-points and image descriptors which are then tracked along different frames to estimate an initial guess of the camera trajectory and the map. In this case, the raw information for localizing and mapping has a fundamentally *geometric* nature.

On the other hand, *direct* methods make use of the full raw input frame, virtually using all the pixel information contained within the image (including their color). Hence, the raw information for localizing and mapping has a *photometric* nature.

Both families of methods have advantages and disadvantages. Indirect methods can cope with large frame-to-frame motions, but they are slow, due to the costly steps of feature extraction, matching, and outlier removal. Indirect methods can potentially exploit all the information in the image, they have higher robustness to motion blur and weak texture, and increasing the camera frame-rate reduces the computational cost per frame. However, they are highly sensitive to initial values, which limit the range of possible frame-to-frame motion. Now, let us talk about some previous work.

1.4 Previous work

Below is a concise survey of seven milestone systems that have shaped the modern landscape of visual SLAM. The papers are ordered to mirror the historical progress, emphasizing the core technical contributions that make these works still highly influential today.

PTAM — Parallel Tracking and Mapping for Small AR Workspaces Klein and Murray (2007)

Split architecture. It introduces the now-standard dual-thread design that decouples rapid pose tracking from slower key-frame bundle adjustment.

Key-frame bundle adjustment. It demonstrates that global bundle adjustment (BA) can run in real-time on commodity CPUs when restricting the optimization to a sliding window of key-frames.

AR focus. It targets hand-held, small-scale workspaces, but proves that SLAM could achieve millimeter-level accuracy suitable for augmented-reality overlay.

MonoSLAM — Real-Time Single Camera SLAM Davison, Reid, Molton, and Stasse (2007)

First fully monocular SLAM. It pioneers Extended Kalman Filter (EKF)-based state estimation that jointly filters camera pose and a sparse set of point landmarks at 30Hz.

Active feature management. It introduces uncertainty-aware feature initialization and an active strategy for selecting image measurements that maximize information gain.

Proof of real-time viability. It shows drift-free localization with nothing but a single, freely-moving camera—laying the foundation for later vision-only systems.

ORB-SLAM — A Versatile and Accurate Monocular SLAM System Mur-Artal, Montiel, and Tardós (2015)

Unified ORB pipeline. It uses the same fast, rotation-invariant ORB key-points for tracking, mapping, re-localization, and loop closing, simplifying the system.

Survival-of-the-fittest map. An aggressive culling strategy allows to keep only high-quality key-frames and map points, enabling life-long operation without unbounded growth.

Robust loop closing. It combines a co-visibility graph with pose-graph optimization (Essential Graph) to achieve state-of-the-art accuracy on large datasets.

LSD-SLAM — Large-Scale Direct Monocular SLAM Engel, Schöps, and Cremers (2014)

Direct, semi-dense formulation. It tracks the camera via photometric error on image intensities, avoiding discrete feature detection.

Per-key-frame depth maps. It estimates semi-dense inverse-depth maps online, which act as both geometry and appearance for subsequent tracking.

Scale-drift correction. It maintains a Sim(3) pose graph to handle unknown scale in monocular sequences, enabling drift-corrected, kilometer-scale trajectories.

DROID-SLAM — Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras Teed and Deng (2021)

Learned dense bundle adjustment. It introduces a differentiable, recurrent BA module that iteratively refines depths and poses within a neural network.

Cross-modal generalization. It is trained only on monocular videos, yet it is capable of exploiting stereo baselines or depth sensors at test-time for further gains.

Robustness leap. It achieves significant accuracy improvements and far fewer catastrophic failures than classical pipelines across diverse datasets.

MonoGS — Gaussian Splatting SLAM Davison et al. (2024)

All-Gaussian representation. It leverages 3-D Gaussian splats as the sole geometric and photometric primitive for simultaneous tracking, mapping, and rendering, *combining the strengths of direct and indirect methods.*

Direct pose optimization. It tracks the camera by minimizing the reprojection error against the splatted scene, bypassing external SfM initialization.

High-fidelity reconstruction. It achieves state-of-the-art novel-view synthesis and accurate trajectories, while running live (∼3 FPS) on monocular video.

MASt3R-SLAM — Real-Time Dense SLAM with 3-D Reconstruction Priors Murai, Dexheimer, and Davison (2024)

Strong two-view prior. It builds the SLAM pipeline around MASt3R Vincent Leroy (2024), a foundational model that can be trained as a two-view matcher that supplies geometry-aware correspondences from the outset.

Globally-consistent dense maps. It delivers scale-aware dense reconstructions and globally-optimized trajectories at 15 FPS with only monocular input.

Plug-and-play design. It requires minimal camera assumptions and demonstrates robustness on in-the-wild videos, setting a new bar for real-time monocular SLAM.

In a retrospective way, we can say that the field has progressed from probabilistic filtering of sparse points (MonoSLAM) through key-frame BA (PTAM, ORB-SLAM), semi-dense direct methods (LSD-SLAM), and learned dense networks (DROID-SLAM), to recent representations that unify high-fidelity rendering with SLAM (MonoGS) and data-driven 3D priors that enable robust dense mapping in the wild (MASt3R-SLAM).

Each breakthrough has introduced a new ingredient—architectural decoupling, direct photometric alignment, learned optimization, or novel scene primitives—that collectively informs contemporary research directions.

Chapter 2

Gaussian Splatting: Key concepts

Novel view synthesis has been a long-standing problem in computer vision and computer graphics. With the advent of deep learning and hardware-accelerated solutions, new rendering techniques have emerged and dominated the current state-of-the-art. In 2020, a landmark paper called "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis" Mildenhall et al. (2020) has introduced a novel neural network-based ray-tracing technique that achieves high rendering quality on several benchmarks. At the core of this technique, the authors propose to use an *implicit* model of the scene (no triangle, no surface, no $3D$ point) through a couple of continuous functions, approximated as neural networks, for representing the radiance and opacity properties all over the $3D$ space. With these functions, one can query the colors or the depths corresponding to the pixels of an image, by integration of the radiance function along the queried rays. On the other hand, NeRF has to settle many technical related issues to perform rendering in real time.

Three years later, in 2023, a new, powerful alternative appeared in the paper "$3D$ Gaussian Splatting for Real-Time Radiance Field Rendering" Kerbl et al. (2023). $3D$ Gaussian Splatting offers high quality rendering as well as real-time operation. In the following, we will refer to this method as $3DGS$ and, in the next sections, we describe it in more detail.

2.1 3D Gaussian splatting: The method

The general setting goes as follows. Let $\mathcal{I}^* = \{\mathbf{I}_j^*\}_{j=1}^N$ be a set of N images (taken with a camera) of the same scene, and let $\mathcal{T}^* = \{\mathbf{T}_{WC_j}^*\}_{j=1}^N$ be the corresponding camera poses (positions and rotations where the images have been taken). Note that all the poses we will deal with are assumed to be expressed in the *same* 3D reference frame (coined as "world frame"). We will use the super-index * to indicate that the data are real/ground-truth, not synthesized or estimated ones. Also note that the images should have some overlap for $3D$ reconstruction to be possible, which means that their field of view should intersect. A camera pose $\mathbf{T}_{WC}$ is a rigid body transformation (rotation and translation) that transforms points from a predefined world frame to the camera viewpoint, that is, $\mathbf{T}_{WC} \in \mathrm{SE}(3)$. The task of novel view synthesis is to generate a *new*, synthetic image $\hat{\mathbf{I}}$ for this same scene, given an arbitrary pose $\mathbf{T}_{WC}$. As we will see in the next section, having $3D$ Gaussians as a representation of the world allows to develop a *fully differentiable* rasterization method to produce $\hat{\mathbf{I}}$, which makes it suitable for gradient-based optimization algorithms as we will see in section 2.1.4. Finally, in order to keep account of the most informative Gaussians and to accurately represent high frequencies from the world, the method uses a heuristic called Adaptive Density Control (ADC) explained in section 2.1.5. An overview of the complete pipeline is shown in figure 2.1.

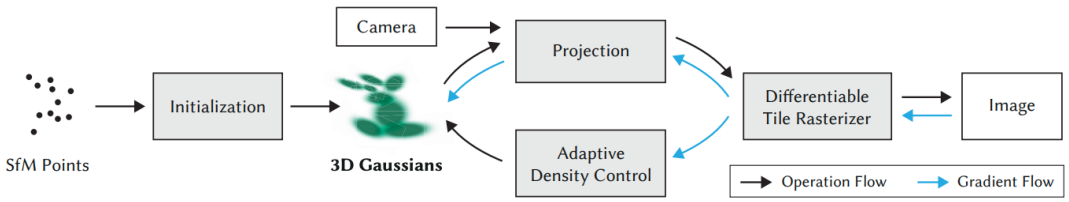

Figure 2.1: 3D Gaussian Splatting pipeline, source: Kerbl et al. (2023).

2.1.1 Differentiable rasterization

To parametrize the shape and geometry of each $3D$ Gaussian $\mathcal{N}(\mu_W, \mathbf{\Sigma}_W)$, the method takes its mean $\mu_W \in \mathbb{R}^3$, that defines where the Gaussian is located in the world, and

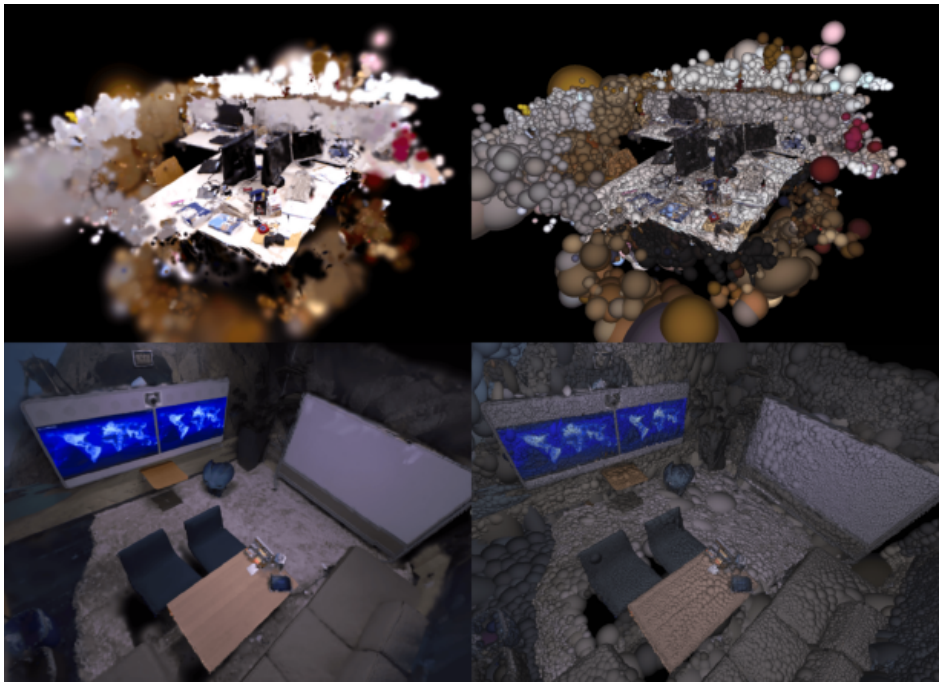

Figure 2.2: Raw 3D Gaussians in the world. Source: "MonoGS" Davison et al. (2024). The left images are rendered images from Gaussian splatting; the right images give a visualization of the underlying representation, i.e. the 3D Gaussians.

the covariance matrix $\mathbf{\Sigma}_W \in \mathbb{R}^{3\times3}$, that defines the scale and the direction of the orthogonal axis of the Gaussian. Since we are interested in having a photometric representation of the world, the method additionally attaches some appearance parameters to each Gaussian; these are: an opacity factor α and color data $\mathbf{c}$, parametrized by spherical harmonic coefficients $\mathbf{f}$, in order to capture the view-dependent features (i.e. the color is in fact a *function* that gives the color as R,G,B components as functions of the direction you are seeing); more details about the spherical harmonics are given in 2.1.2. The important point here is that these parameters are *learned* through an optimization process. Figure 2.2 illustrates the raw Gaussians in 3D world, and we can visualize their shapes and appearance parameters.

As in many previous works such as NeRF Mildenhall et al. (2020), in order to produce a color $\mathbf{C_x}$ at a pixel $\mathbf{x}$ of the synthesized image $\hat{\mathbf{I}}$, the $3DGS$ image formation model follows the *alpha blending strategy* specified by the following equation:

$$\mathbf{C}_{\mathbf{x}} = \sum_{i \in \mathcal{N}} \mathbf{c}_i \alpha_i(\mathbf{x}) \prod_{j=1}^{i-1} (1 - \alpha_j(\mathbf{x})), \tag{2.1}$$

where the sum is taken over the set $\mathcal{N}$ of Gaussians and the sub-indices i, j refer to some of these Gaussians. This can be seen as a weighted sum of the colors $\mathbf{c}_i$ associated to the $3D$ Gaussians i traversed along the ray that starts at the center of the camera and intersects the corresponding pixel. The crucial term here is $\alpha_i(\mathbf{x})$, which is the opacity of Gaussian i, decayed by the value (at $\mathbf{x}$) of the projection of this $3D$ Gaussian to the image plane: The final color at a given pixel $\mathbf{x}$ will result from a mixture of colors from the different Gaussians, weighted by these $\alpha_i(\mathbf{x})$, which fade out when getting farther from a Gaussian center. The terms $1 - \alpha_j(\mathbf{x})$ are contributions from Gaussians located *in front* of Gaussian i: They allow to model the occlusions between Gaussians. Note that this implies that the indices i, j follow an *order* of the Gaussians along the z axis: From the camera to the world.

To maintain the Gaussian form and the differentiability of the whole process, this projection is approximated by the Jacobian of the projection transformation, so that the mean μ_I and covariance matrix $\mathbf{\Sigma}_I$ of the "splatted" Gaussian $\mathcal{N}(\mu_I, \mathbf{\Sigma}_I)$ are

$$\mu_I = \pi(\mathbf{T}_{CW} \cdot \mu_W), \quad \mathbf{\Sigma}_I = \mathbf{J}\mathbf{W}\mathbf{\Sigma}_W\mathbf{W}^T\mathbf{J}^T, \tag{2.2}$$

where π is the perspective projection operation, $\mathbf{J}$ is the Jacobian of the projection transformation, and $\mathbf{W}$ is the rotational component of $\mathbf{T}_{WC}$. We insist that this is an approximation: The projection of a $3D$ Gaussian distribution is in general *not* a $2D$ Gaussian distribution, because said projection is *not linear*. However, this process allows to have a fully differentiable pipeline, which is crucial for the optimization process.

A critical aspect is that the sum of Eq. 2.1 is evaluated *directly in the image plane* (whereas for example, in Nerf, it is evaluated in 3D, along rays) and taken over $\mathcal{N}$, the set of Gaussians sorted along the corresponding ray *according to their depth*. The

Figure 2.3: Comparison of original image (left) vs rasterized image (right) on TUM-RGBD dataset (top) and Replica dataset (bottom).

sorting algorithm employed in $3DGS$ is one the of the main contributions of the original paper; additionally, it groups the Gaussians into tiles; this allows to weight just the relevant $3D$ points in the alpha blending process and alleviate the while rendering process. As well as rasterizing an RGB image by putting the color $\mathbf{c}_i$ in equation 2.1, we can also replace the color with the depth component z_i of each Gaussian and rasterize a depth map, that we can use to assess the geometry of the reconstructed scene. The figure 2.3 shows a comparison of original images against final rasterization with Gaussian Splatting.

2.1.2 Modeling choices

Since we are interested in geometrically meaningful Gaussians, $3DGS$ make some strong modeling choices to impose these restrictions and to keep the process computationally feasible.

Parameterization

The Gaussians are parameterized by their mean (which is a free variable in $\mathbb{R}^3$) and their covariance matrix. Now, the covariance matrix $\mathbf{\Sigma}_W$ only has physical meaning if it is a positive semi-definite matrix. Since it may be messy to impose this restriction directly during the optimization process, it is preferable to factorize it into a positive diagonal scale matrix $\mathbf{S}$ and a rotation matrix $\mathbf{R}$ as

$$\mathbf{\Sigma} = \mathbf{RSS}^T\mathbf{R}^T. \tag{2.3}$$

To allow independent optimization of both factors, these parameters are stored separately: a $3D$ vector for scaling and a quaternion to represent rotation.

Now, a constraint is that the scales in $\mathbf{S}$ should be kept positive. To enforce this constraint, an exponential is applied as activation function to the optimized variable. Said in another way, we optimize the logarithms of the scales.

Second, the opacities α_i should be in the $[0, 1]$ interval. For this constraint, the method uses a logit variable instead of α_i, i.e. $\log \frac{\alpha_i}{1-\alpha_i}$, and applies a sigmoid as the activation function.

Spherical harmonics

The method captures view-dependent color appearances by modeling them with spherical harmonics functions as:

$$\mathbf{c} = \sum_{l=0}^{L} \sum_{m=-l}^{l} \mathbf{f}_{lm} Y_{lm}(\mathbf{d}(\mu)), \tag{2.4}$$

where $\mathbf{c}$ is the color of the Gaussian primitive, L is the degree of spherical harmonics, $\mathbf{f}_{lm}$ are the spherical harmonics coefficients, $\mathbf{d}(\mu)$ is the view direction at position μ and $Y_{lm}(\mathbf{d}(\mu))$ is the spherical harmonics basis function with direction $\mathbf{d}(\mu)$. This corresponds to a decomposition of a function of interest with support on the unit sphere on a basis of functions on this unit sphere.

A foundational reference on Spherical Harmonic (SH) lighting is provided by Green (2003), who introduces efficient real-time rendering techniques using SH projection and dot-product evaluation of light integrals Green (2003). SH are widely adopted in computer graphics for low-frequency lighting applications such as global illumination and precomputed radiance.

2.1.3 Initialization

The original method assumes that we start with precomputed camera poses $\{\mathbf{T}^*_{WC_i}\}_{i=1}^N$, corresponding to our training images $\mathbf{I}^*_i$ and a sparse point cloud reconstruction of the scene. This setting can be obtained with established Structure from Motion (SfM) pipelines, and the most popular software that condense state of the art methods is Colmap Schönberger and Frahm (2016). Then, $3DGS$ initializes the mean or position of each Gaussian at the point from the sparse point cloud obtained from SfM. Hereafter we explain the training process.

2.1.4 Training

Until now, we have seen that $3DGS$ gives us a way to rasterize an image $\hat{\mathbf{I}}(\mathcal{G}, \mathbf{T}_{WC})$ from a set of Gaussian primitives $\mathcal{G}$ and a camera pose $\mathbf{T}_{WC}$. Now, we need to make some sense out of this bunch of blobs in space and try to fit it to the images we have. In this regard, it is necessary to compare the rendered images against the real pictures, make any modification to the Gaussians to improve the fit between the rendered and real images and repeat until we have an acceptable reconstruction. So, the problem of building an accurate photometric representation of the world reduces to solving the following optimization problem:

$$\min_{\mathcal{G}} \sum_{j=1}^{N} ||\hat{\mathbf{I}}(\mathcal{G}, \mathbf{T}^*_{WC_j}) - \mathbf{I}^*_j||_1, \tag{2.5}$$

where $\mathbf{I}^*_j$ is one of the N original images, $\mathbf{T}^*_{CW_j}$ are the known camera pose parameters corresponding to this image j, and $\hat{\mathbf{I}}(\mathcal{G}, \mathbf{T}^*_{CW_j})$ the rasterized image obtained from

the Gaussians $\mathcal{G}$ we have described in the previous sections, along the rasterization process. Note that, since this rasterization process is fully differentiable, a gradient descent-like method can be applied to solve the problem 2.5. Also note that, what will change in the context of SLAM (next chapter) is that the camera poses $\mathbf{T}_{WC}$ *will not be known*, making the problem a bit more difficult to solve.

In the original article, the authors additionally include to the loss function above another term using the Structural Similarity Index Measure (SSIM), which is a perception measure that takes in to account the luminance, contrast and structure. The contribution of this term to the loss is important because, in contrast to the $\mathcal{L}_1$ loss, the SSIM assumes correlation between pixels, mainly between neighbors.

2.1.5 Adaptive Density Control

The whole method starts from an initial sparse point cloud obtained from Structure from Motion (SfM), as described in 2.1.3, such that the means and colors are initialized from the position and colors of the sparse points present in the initial point cloud, typically hundreds of points. Now, one can guess that this initialization may give Gaussians that do not come in sufficient numbers in some places, or that are not necessary in others. Hence, with the original $3DGS$ method, the authors have developed a strategy to control the number of Gaussians and the density per unit volume.

To take into account informative primitives, the method periodically *prunes* those Gaussians with an opacity α_i below a threshold, that are considered almost "transparent". Similarly, the method prunes those with a large footprint in view-space. On the other hand, the method populates regions of missing geometric features, the so-called "under-reconstructed regions" but also those where Gaussians cover a large area, corresponding to "over-reconstruction" areas. Both cases induce a large view-space positional gradient (i.e. the gradient of the loss with respect to the x, y, z parameters of the Gaussian), which can be interpreted as a lack of geometric representation. For Gaussians in the case of "under-reconstruction", the method creates a copy and moves

it in the direction of the positional gradient. For the Gaussians in the case of "over-reconstruction", it is preferable to split those primitives and initialize their positions using the original Gaussians as a probability density function.

2.2 2D Gaussian Splatting

There are some potential problems with $3DGS$ that we discuss in the following. Due to its $3D$ nature, during rendering, $3DGS$ evaluates a Gaussian value at the intersection between pixel ray and $3D$ Gaussian. Depending on the point of view, this intersection plane will change, leading to a possible inconsistency on the depth values, figure 2.4 illustrates this discrepancy. As we have seen, in order to maintain the Gaussian form, the $3D$ Gaussians are splatted linearly in 2.2, but, as we have seen before, this projection is an approximation that is only accurate at the center of the Gaussian and has increasing error as we move far away. Moreover, the 3D Gaussians in the world have no geometric correlation, making it difficult to extract the associated surface. To solve these issues, a new method is proposed in "2D Gaussian Splatting for Geometrically Accurate Radiance Fields" B. Huang, Zehao Yu, Geiger, and Gao (2024). Using $2D$ ellipses as primitives, instead of $3D$ ellipsoids, this modification allows us to correct the perspective inconsistencies, project *analytically* to the image plane, and naturally build a normal vector to the actual surface. In the following, we will refer to the method as $2DGS$.

2.2.1 Surfels perspective correction

Just as in $3DGS$, each ellipse is specified by its central point $\mathbf{p}_c$, two principal tangential vectors $\mathbf{t}_u, \mathbf{t}_v$, and two scalars (s_u, s_v) that control the variances of the $2D$ Gaussian. Then each disk is parameterized as the two-coordinate system:

$$\mathbf{p}(u, v) = \mathbf{p}_c + s_u \mathbf{t}_u u + s_v \mathbf{t}_v v. \tag{2.6}$$

Observe that, based on this parameterization, we can define the normal vector for each

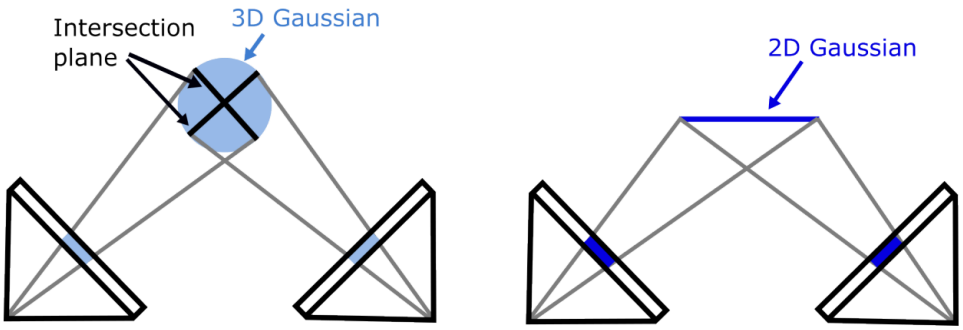

Figure 2.4: Comparison of 3DGS and 2DGS. 3DGS utilizes different intersection planes for value evaluation when viewing from different viewpoints, resulting in inconsistency.

disk as $\mathbf{t}_w = \mathbf{t}_u \times \mathbf{t}_v$. We can arrange the orientation in the matrix $\mathbf{R} = [\mathbf{t}_u, \mathbf{t}_v, \mathbf{t}_w]$ and the scalars in the diagonal matrix as $\mathbf{S} = \text{diag}(s_u, s_v, 0)$. Then, we can rewrite the $3D$ points from 2.6 in homogeneous coordinates, as vectors in $\mathbb{R}^4$, as:

$$\mathbf{p}(u, v) = \mathbf{H}(u, v, 1, 1)^T, \tag{2.7}$$

where

$$\mathbf{H} \triangleq \begin{bmatrix} s_u\mathbf{t}_u & s_v\mathbf{t}_v & \mathbf{0} & \mathbf{p}_c \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{RS} & \mathbf{p}_c \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.8}$$

is the 4×4 transformation matrix that describes the geometry of the Gaussian. Exactly as in $3DGS$, the center $\mathbf{p}_k$, the scales (s_u, s_v) and the rotation $(\mathbf{t}_u, \mathbf{t}_v)$ are learnable parameters. For a point $\mathbf{u} = (u, v)$ in the local coordinates of the uv space, its $2D$ Gaussian value can then be evaluated by its standard Gaussian:

$$\hat{\mathcal{G}}(\mathbf{u}) = \exp\Big(-\frac{u^2 + v^2}{2}\Big). \tag{2.9}$$

Let $\mathbf{T}_{CW}$ be the transformation from the world coordinates to the camera coordinates, then the points as seen in the screen space can be expressed as:

$$\mathbf{x}_s = (xz, yz, z, 1) = \mathbf{T}_{CW}\mathbf{p}(u, v) = \mathbf{T}_{CW}\mathbf{H}(u, v, 1, 1). \tag{2.10}$$

Based on the previous expression, we could be tempted to use the implicit transformation $(\mathbf{T}_{CW}\mathbf{H})^{-1}$ to go back from the image coordinates to the local coordinates of the ellipses, but as stated in B. Huang et al. (2024), this leads to numerical instability. On the other hand, if we opt to use the affine approximation of the perspective projection as in $3DGS$, this has an increasing error as we move away from the center. The crucial observation made in B. Huang et al. (2024), is that projecting the $2D$ splat onto an image plane can be described by general $2D$-to-$2D$ mapping in homogeneous coordinates.

An image coordinate $\mathbf{x} = (x, y)$ is characterized by being orthogonal (in homogeneous coordinates) to the two vectors $\mathbf{h}_x = (-1, 0, 0, x)$ and $\mathbf{h}_y = (0, -1, 0, y)$. In other words, we have:

$$(x, y, 1, 1) \cdot \mathbf{h}_x = (x, y, 1, 1) \cdot \mathbf{h}_y = 0, \tag{2.11}$$

which are in fact the normal vectors to the $x-$plane and $y-$plane. It is well known that to transform the normals vectors from one coordinate system to another, we need to apply the inverse transpose of the transformation, in our case, of $(\mathbf{T}_{CW}\mathbf{H})^{-1}$, so the normal vectors, $\mathbf{h}_u, \mathbf{h}_v$ in the local coordinates of the uv space are specified as:

$$\mathbf{h}_u = (\mathbf{T}_{CW}\mathbf{H})^T\mathbf{h}_x \quad \mathbf{h}_v = (\mathbf{T}_{CW}\mathbf{H})^T\mathbf{h}_y. \tag{2.12}$$

Hence, the associated point to the image coordinate $\mathbf{x}$ on the $2D$ Gaussian plane, i.e. the intersection of the ray that goes through $\mathbf{x}$ with the disk, should be orthogonal to the transformed normal vectors:

$$\mathbf{h}_u \cdot (u, v, 1, 1)^T = \mathbf{h}_v \cdot (u, v, 1, 1)^T = 0. \tag{2.13}$$

Solving this linear system of equations, we finally obtain:

Figure 2.5: Perspective correction. 2DGS to the left, 3DGS to the right

$$u(\mathbf{x}) = \frac{\mathbf{h}_u^2\mathbf{h}_v^4 - \mathbf{h}_u^4\mathbf{h}_v^2}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \quad v(\mathbf{x}) = \frac{\mathbf{h}_u^4\mathbf{h}_v^1 - \mathbf{h}_u^1\mathbf{h}_v^4}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \tag{2.14}$$

where $\mathbf{h}_u^i$, $\mathbf{h}_v^i$ are the i-th parameter of the $3D$ plane (expressed in $4D$ homogeneous coordinates). Note that $\mathbf{h}_u^3$ and $\mathbf{h}_v^3$ are always zero according to 2.8.

The equations 2.14 are crucial for $2DGS$, as they give us an *explicit* mapping from the image coordinates to the local coordinates on the plane where the $2D$ Gaussian is located. This gives the key in allowing to evaluate efficiently the terms $\alpha_i(\mathbf{x})$ in equation 2.1: For each pixel, and for a given Gaussian, we will have access to the local coordinates u, v and we can evaluate the Gaussian through Eq. 2.9. Here, we emphasize that the constructed mapping is *exact*, contrary to what we have seen in $3DGS$. Figure 2.5 illustrates the perspective correction of $2DGS$, additionally we can observe that, contrary to $3DGS$, the depth rasterized with $2DGS$ varies within the same ellipses.

2.2.2 Distortion and Normal Regularization

One of the main contributions of $2DGS$ is the capability of the proposed system to accurately reconstruct the geometry of the scene. The method achieves this by adding two regularization terms to the photometric loss (similar to Eq. 2.5), which we discuss

in the following.

Depth distortion term.

In $3DGS$, the splats along a ray could potentially be very sparse and the photometric result would be the same, but in surface rendering the rays intersect the first visible surface exactly once. To mitigate this issue, $2DGS$ introduces a regularization term that essentially *concentrates* the splats along the ray. This term is specified as:

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j|, \tag{2.15}$$

where $\omega_i = \alpha_i \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1}(1 - \alpha_j \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$ is the blending weight of the $i-$th intersection and z_i is the depth of the intersection points. The sum is taken along rays and penalizes Gaussians located far away one from the other.

Normal consistency term.

Now that we can easily define the normal vector to each Gaussian, we need to ensure that these normals are indeed aligned with the actual surface; the way the method proposes to do so is to use the gradient of the depth map as a prior. First, $2DGS$ consider the actual surface at the median intersection point $\mathbf{p}_s$, which is the point where the accumulated opacity (starting from the camera) reaches 0.5. On the other hand, with the depth map, it estimates the normal to the surface as the cross product of the gradient at that point, using finite differences from nearby depth points as follows:

$$\mathbf{N}(x, y) = \frac{\nabla_x d_{\mathbf{p}_s} \times \nabla_y d_{\mathbf{p}_s}}{|\nabla_x d_{\mathbf{p}_s} \times \nabla_y d_{\mathbf{p}_s}|}. \tag{2.16}$$

Finally, all the splats' normals are aligned with $\mathbf{N}$, by penalizing non-parallel normals.

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N}), \tag{2.17}$$

where i indexes over intersected splats along the ray, ω_i denotes the blending weight

Figure 2.6: Visualization of the of RGB original image and the normals map estimated from the Gaussians.

of the intersection point (see above) and $\mathbf{n}_i$ represents the normal of the splat that is oriented towards the camera. Figure 2.6 illustrates the render normal map encoded in the RGB format. The final loss is:

$$\mathcal{L} = \mathcal{L}_{pho} + \alpha\mathcal{L}_d + \beta\mathcal{L}_n, \tag{2.18}$$

where $\mathcal{L}_{pho}$ is the photometric loss stated as before in Eq. 2.5, and α, β are two parameters to weight the contribution of each regularization term. The rest of the pipeline is the same as with $3DGS$.

Chapter 3

Proposed methodology

In this chapter, we detail the methodology we have developed to construct a robust and efficient Simultaneous Localization and Mapping (SLAM) system leveraging 2D Gaussian Splatting ($2DGS$), that we have described in the previous Chapter. We begin by describing the general pipeline, which integrates distinct tracking and mapping modules to achieve accurate real-time camera pose estimation and dense scene reconstruction. The tracking module specifically focuses on optimizing the camera poses via differentiable rendering techniques that minimize photometric discrepancies between rendered and observed frames. Subsequently, we provide a comprehensive derivation of the analytic camera pose Jacobian, which significantly enhances computational efficiency during pose optimization. Furthermore, we outline essential image pre-processing strategies to ensure the effective utilization of valuable pixel information. Finally, we discuss our approach to keyframe selection and management, as well as the strategies employed for Gaussian insertion and adaptive density control to maintain computational efficiency without sacrificing reconstruction accuracy. The subsequent sections describe each component of our system in greater detail.

3.1 General pipeline

The proposed SLAM system employs a structured pipeline that integrates a tracking module and a mapping module, both built around the principle of 2D Gaussian

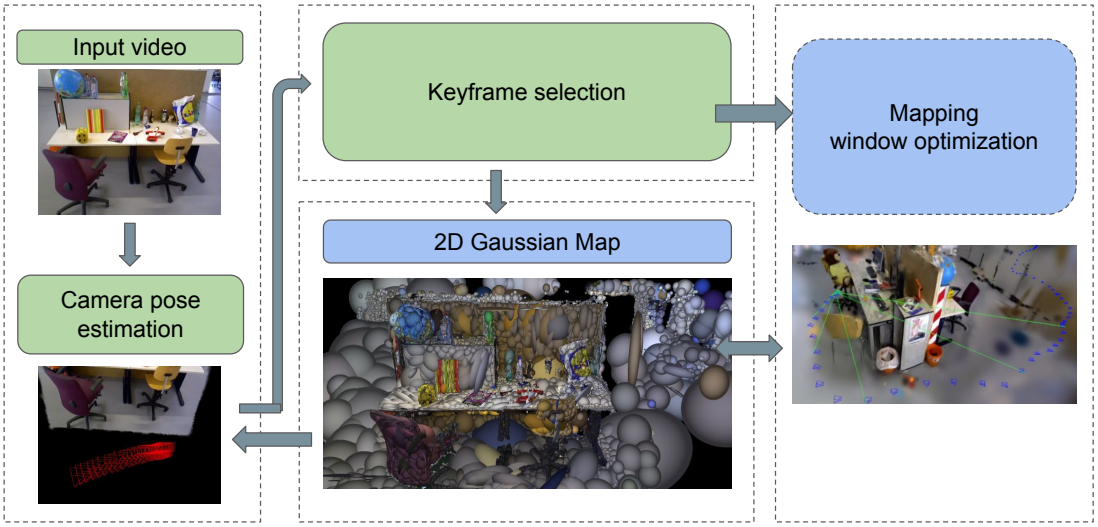

Figure 3.1: General pipeline of the proposed approach.

Splatting ($2DGS$). Figure 3.1 illustrates the full pipeline. Initially, the tracking module estimates the camera pose by minimizing a photometric loss function through differentiable rendering, matching the current observed frame with a rendered representation of the scene. This process leverages analytic derivatives of the camera poses, significantly improving computational efficiency.

Concurrently, a dedicated image pre-processing step ensures the optimal use of input frame information. This involves applying masks based on color brightness, gradient intensities, and opacity distributions to selectively emphasize pixels of high photometric value.

Subsequent to tracking, the system implements a robust keyframe selection strategy based on Gaussian covisibility and relative camera motion, ensuring keyframes contribute meaningful scene information while maintaining computational tractability. Once keyframes are established, the mapping module performs dense scene reconstruction by inserting new Gaussians derived from the depth maps, refining them iteratively through adaptive density control (see Section 2.1.5) and full-window bun-

dle adjustment optimizations. Throughout this process, camera poses are periodically revisited and adjusted to ensure global consistency and accuracy.

Each component of this pipeline synergistically contributes to the SLAM system's ability to achieve real-time, accurate pose estimation and detailed, dense scene reconstruction, laying the groundwork for robust localization and mapping applications.

3.2 Tracking module

The tracking module is in charge of estimating the camera pose of each new frame with respect to the last keyframe, if possible in real time. Using the differentiability property of the rasterization process and the fast rendering method of $2DGS$, we minimize the photometric loss to find the camera pose parameters $\hat{\mathbf{T}}_{CW}$ that best adjust the rendering with the input frame, that is:

$$\hat{\mathbf{T}}_{CW} = \arg\min_{\mathbf{T}_{CW}} \{\mathcal{L}_{pho} \triangleq ||\hat{\mathbf{I}}(\mathcal{G}, \mathbf{T}_{CW}) - \mathbf{I}^*||_1\}, \tag{3.1}$$

where $\mathbf{I}^*$ is the last acquired image. Note that this is the same loss that we have seen in the previous chapter, except that we optimize it only with respect to the pose parameters $\mathbf{T}_{CW}$, not with respect to the Gaussian parameters $\mathcal{G}$. Additionally, if we have access to depth information (through an acquired depth image $\mathbf{D}^*$), we may also include the geometric depth loss, specified as:

$$\mathcal{L}_{geo} = ||\hat{\mathbf{D}}(\mathcal{G}, \mathbf{T}_{CW}) - \mathbf{D}^*||_1, \tag{3.2}$$

where $\hat{\mathbf{D}}(\mathcal{G}, \mathbf{T}_{CW})$ is the rendered depth image.

During tracking, the map $\mathcal{G}$ is assumed to be fixed, and the optimization is performed just over the camera pose parameters $\mathbf{T}_{CW}$, i.e. very few parameters (six, in principle). As the optimization is done with gradient descent techniques, an important point is the computation of the gradient of the loss function with respect to these pose parameters. In the following, we give a few elements about it.

3.2.1 Analytic camera pose Jacobian

The original implementation of $2DGS$ assumes that the camera poses are known or have been pre-computed with algorithms like Structure from Motion (SfM), in an offline fashion. To avoid the overhead of the derivative computation and to improve the computational speed, the authors of Hidenobu Matsuki (2025) derive the *analytic* Jacobian of the camera pose for $2DGS$ and implement it with a custom CUDA kernel. Let us rewrite it here in more detail. Ultimately, we wish to compute the derivative of the loss function $\mathcal{L}$ with respect to the camera pose $\mathbf{T}_{CW}$. To do so, we use the minimal parameterization of SE(3) given by the Lie algebra $\mathfrak{se}(3)$ Joan Solà (2018). Hence, let us consider a vector τ in this space. To simplify the notation, let us denote the rendered image $\hat{\mathbf{I}}(\mathcal{G}, \mathbf{T}_{CW})$ as simply $\hat{\mathbf{I}}$. By applying the chain rule, we have:

$$\frac{\partial \mathcal{L}}{\partial \tau} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}} \frac{\partial \hat{\mathbf{I}}}{\partial \tau} \tag{3.3}$$

$$= \sum_{i \in \mathcal{N}} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}} \frac{\partial \hat{\mathbf{I}}}{\partial \alpha_i} \frac{\partial \alpha_i}{\partial \tau} \tag{3.4}$$

$$= \sum_{i \in \mathcal{N}} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}} \frac{\partial \hat{\mathbf{I}}}{\partial \alpha_i} \frac{\partial (\tilde{\alpha}_i \hat{\mathcal{G}}_i(\mathbf{u}))}{\partial \tau} \tag{3.5}$$

$$= \sum_{i \in \mathcal{N}} \tilde{\alpha}_i \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}} \frac{\partial \hat{\mathbf{I}}}{\partial \alpha_i} \frac{\partial \hat{\mathcal{G}}_i(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \tau} \tag{3.6}$$

$$= \sum_{i \in \mathcal{N}} \tilde{\alpha}_i \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}} \frac{\partial \hat{\mathbf{I}}}{\partial \alpha_i} \frac{\partial \hat{\mathcal{G}}_i(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{T_{CW}}} \frac{\partial \mathbf{T_{CW}}}{\partial \tau}, \tag{3.7}$$

where the sum is taken as in 2.1, $\tilde{\alpha}_i$ is the opacity scalar value associated to each Gaussian, $\mathbf{u}$ is the solution to the intersection point of the pixel ray with the Gaussian disk, specified in equation 2.14, and $\alpha_i \triangleq \tilde{\alpha}_i \hat{\mathcal{G}}_i(\mathbf{u})$ is the opacity contribution ponderated by the Gaussian evaluation at $\mathbf{u}$. Observe that $2DGS$ backpropagates through this term to the matrix $\mathbf{T}_{CW}$. Then we need to compute the derivative, $\frac{\partial \mathbf{T}_{CW}}{\partial \tau}$. We can write:

$$\frac{\partial \mathbf{T}_{CW}}{\partial \tau} = \lim_{\tau \to 0} \frac{\exp(\tau^{\wedge})\mathbf{T}_{CW} - \mathbf{T}_{CW}}{\tau} \tag{3.8}$$

$$= \lim_{\tau \to 0} \frac{(\mathbf{I} + \tau^{\wedge})\mathbf{T}_{CW} - \mathbf{T}_{CW}}{\tau} \tag{3.9}$$

$$= \lim_{\tau \to 0} \frac{\tau^{\wedge}\mathbf{T}_{CW}}{\tau}, \tag{3.10}$$

where the notation $\exp(\tau^{\wedge})$ refers to the exponential map Joan Solà (2018). Since $\tau^{\wedge}$ lives in the Lie algebra $\mathfrak{se}(3)$, it has the form:

$$\tau^{\wedge} = \begin{pmatrix} \theta^{\times} & \rho \\ \mathbf{0} & 0 \end{pmatrix}, \tag{3.11}$$

where $\tau = (\rho, \theta) \in \mathbb{R}^6$ are the six degrees of freedom camera extrinsic parameters and $\mathbf{v}^{\times}$ refers to the skew-symmetric matrix associated to a vector $\mathbf{v}$ (i.e., the unique matrix such that for any $\mathbf{w} \in \mathbb{R}^3$, $\mathbf{v}^{\times}\mathbf{w} = \mathbf{v} \times \mathbf{w}$). Then, by expanding the terms in the numerator in 3.10, we have:

$$\lim_{\tau \to 0} \frac{\begin{pmatrix} \theta^{\times}\mathbf{R} & \theta^{\times}\mathbf{t} + \rho \\ \mathbf{0} & 0 \end{pmatrix}}{\tau}, \tag{3.12}$$

where $\mathbf{R}$ and $\mathbf{t}$ are the rotational and translational components of $\mathbf{T}_{CW}$. If we flatten the matrix in the numerator to consider just the relevant terms we have:

$$\lim_{\tau \to 0} \frac{(\theta^{\times}\mathbf{R}_{:,1}, \theta^{\times}\mathbf{R}_{:,2}, \theta^{\times}\mathbf{R}_{:,3}, \theta^{\times}\mathbf{t} + \rho)}{\tau}. \tag{3.13}$$

Observe now that 3.13 is an "ordinary" Jacobian in $\mathbb{R}^{12 \times 6}$. Before taking the explicit derivative, let us rearrange the terms in the numerator:

$$\lim_{\tau \to 0} \frac{(-\mathbf{R}^{\times}_{:,1}\theta, -\mathbf{R}^{\times}_{:,2}\theta, -\mathbf{R}^{\times}_{:,3}\theta, -\mathbf{t}^{\times}\theta + \rho)}{\tau}, \tag{3.14}$$

where $\mathbf{R}_{:,i}$ denotes the $i-$th column of matrix $\mathbf{R}$. Observe that we have used the

property $\mathbf{v}^{\times}\mathbf{w} = -\mathbf{w}^{\times}\mathbf{v}$ that satisfies any pair of vectors $\mathbf{v}$ and $\mathbf{w}$. Finally, the Jacobian is left to us as follows:

$$\frac{\partial \mathbf{T}_{CW}}{\partial \tau} = \begin{pmatrix} \mathbf{0} & -\mathbf{R}^{\times}_{:,1} \\ \mathbf{0} & -\mathbf{R}^{\times}_{:,2} \\ \mathbf{0} & -\mathbf{R}^{\times}_{:,3} \\ \mathbf{I} & -\mathbf{t}^{\times} \end{pmatrix}. \tag{3.15}$$

The backward pass with the derivatives computed explicitly as before is implemented in custom CUDA kernels by Hidenobu Matsuki (2025).

3.2.2 Image pre-processing

To better extract the information within the RGB image, the method applies some preliminary processing to each input frame, which consists of a series of mask applications to promote those pixels with high quality, which are then used during tracking and mapping. In tracking, only the pixels that are selected through these masks are used for the optimization of Eq. 3.1, which is also a way to alleviate the computational burden related to this optimization.

Color mask

To avoid considering a pixel with low brightness, we apply a color mask and only consider those pixels in which the total brightness along the R, G, and B channels is above a predefined threshold. This masking process is illustrated in Fig. 3.2.

Gradient mask

In order to accurately track the camera pose, we attach more importance to those pixels with high bright changes, because there is a lot of ambiguity in which pixel is which in uniform zones. To do this, we first compute the gradient norm of the input frame and then take just the pixels above a threshold aware of the scale of the gradient norm intensities. This masking process is illustrated in Fig. 3.3.

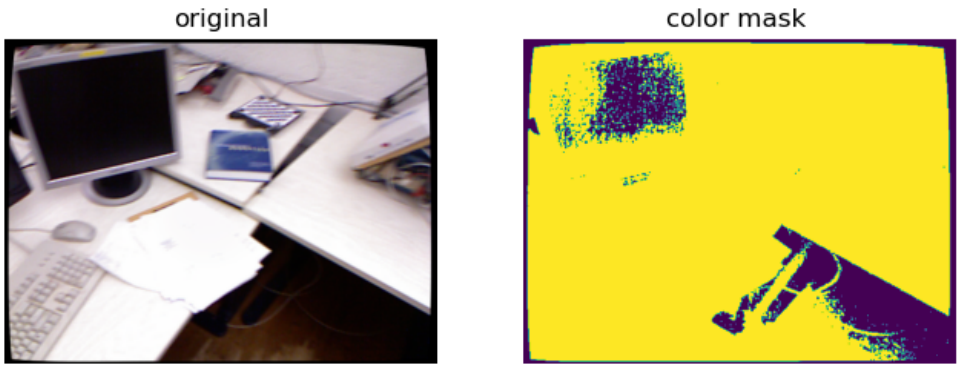

Figure 3.2: Original image compared with the corresponding color mask. Dark regions correspond to the pixels with low level of brightness.

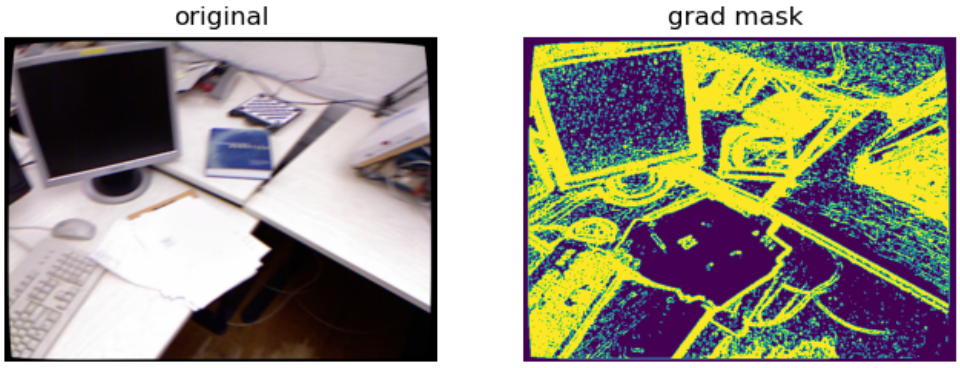

Figure 3.3: Original image compared with the corresponding gradient mask. Dark regions correspond to low levels of gradient intensity.

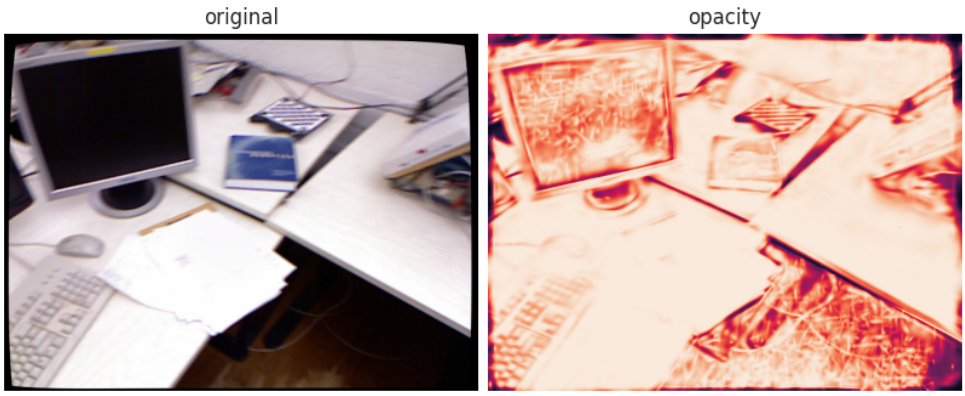

Figure 3.4: Original image compared with opacity map. Black regions correspond to low population of Gaussians.

Opacity mask

Since we are using a Gaussian Splatting-based camera localization pipeline, it is important to endorse regions with a high population of Gaussian splats, i.e. pixels with relevant enough information coming from the Gaussians along the corresponding ray. We extract that information by rendering an opacity map at each pixel $\mathbf{x}$ as:

$$\mathbf{O}_{\mathbf{x}} = \sum_{i \in \mathcal{N}} \alpha_i(\mathbf{x}) \prod_{j=1}^{i-1} (1 - \alpha_j(\mathbf{x})). \tag{3.16}$$

This formula is very similar to the one we have already described in 2.1, but without considering any color channel. It just accumulates the opacity along the ray. The idea is that we should not use the zones with low opacity to optimize the camera pose, since the density of Gaussians is lower in those areas. Then, we multiply this opacity map by the input frame in a pixel-wise setting to give higher contributions to the interest regions (high opacities).

This masking process is illustrated in Fig. 3.4.

3.2.3 Keyframe selection

The mapping module is the most computationally intensive part of the system, so it is necessary to develop some strategies to achieve efficiency; one of the most common heuristics is to select some input frames, which we call *keyframes*, such that these

frames contribute with novel information about the scene. The mapping process performs its optimization tasks only based on these selected frames.

Then we keep a sliding window of the most recent keyframes plus some random past keyframes to avoid the possible forgetting of previous scene details. In this section, we will review a strategy that considers the underneath Gaussian splats structure and that is robust to occlusions, as well as the window and Gaussian insertion heuristics.

Gaussian covisibility

An important concept for deciding about the keyframe selection is the feature covisibility, which is whether a feature is seen or not in a pair of frames. We follow the same strategy as Davison et al. (2024). We say that a Gaussian splat is *visible* from a view (1) if it is used during rasterization (i.e. for some pixel, it appears in the alpha blending formula) and (2) if the alpha blending weight $\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$ up to this Gaussian has not yet reached 0.5 (which means that there is no occlusion from Gaussians present in front of this one). Note that the second criterion uses the unweighted Gaussians opacity parameters α_i and the depth order obtained at the tile corresponding to the Gaussian centroid projection to perform the evaluation. Let us denote $\mathcal{V}_i$ and $\mathcal{V}_{kf}$ the set of visible Gaussians from the $i-$view and from the last keyframe, respectively; then if the intersection over the union ratio drops below a threshold γ, that is:

$$\frac{|\mathcal{V}_i \cap \mathcal{V}_{kf}|}{|\mathcal{V}_i \cup \mathcal{V}_{kf}|} < \gamma, \tag{3.17}$$

or if the relative translation is large with respect to the median depth, then we register frame i as a new keyframe. The idea is that if the new frame "different enough" from the last keyframe, then it is worth considering it as a new keyframe.

Window management

To maintain efficiency, we keep a relatively small window size. If the Gaussian covisibility ratio between the current input frame and any of the keyframes within the window drops below a threshold, then it is removed from the window. Similarly, if the window is full, we use the following heuristic to remove that one keyframe relatively distant from the current input frame. Let us denote the translation from keyframe i to keyframe j as $\mathbf{t}_{i,j}$, then for each keyframe within the window, we compute the following index:

$$W_i = \frac{\|\mathbf{t}_{i,curr}\|}{\sum_{j \neq i} \|\mathbf{t}_{i,j}\|}, \tag{3.18}$$

where $\mathbf{t}_{i,curr}$ denotes the translation distance between the keyframe i and the current input frame, and the sum is taken over the keyframes within the window different from i. Then we remove the keyframe i with the highest index W_i. In this way, we manage to remove keyframes relatively away from the current input frame and too close from the other keyframes within the window, avoiding unnecessary overlapping.

Gaussian insertion

Each time we select a new keyframe, we insert new Gaussians by using the depth map signal. Namely, we backproject the depth values and initiate the centers of each Gaussian at these points. In a similar way, we initialize the normals of each Gaussian disk using the depth signal by taking the cross product of the finite horizontal and vertical differences of neighboring points. To keep memory efficiency, we randomly select pixels to insert new Gaussians. Then, during the mapping stage, the population is refined by the adaptive density control mechanism (see Section 2.1.5).

3.3 Mapping

The mapping module is in charge of producing the dense photometric representation of the scene. We follow the same training pipeline as in Chapter 2, tailored to the

sliding window strategy. Each time we select a keyframe, we perform a relatively small amount of optimization iterations, and when we have a full window of keyframes, we perform a full bundle adjustment with a larger number of training iterations. In addition, we revisit the camera poses to ensure global consistency. Let us describe this module in more detail.

3.3.1 Cost function

During the execution of the mapping module, we use the photometric loss $\mathcal{L}_{pho}$ (see Eq. 3.1) plus the geometric loss $\mathcal{L}_{geo}$ (see Eq. 3.2) as the loss objective, similarly to the tracking module. In addition, to maintain the normals to each Gaussian disk aligned with the real geometric signal given by the depth map, we add to the loss function the normal consistency loss $\mathcal{L}_n$ proposed in B. Huang et al. (2024) and described before in section 2.2.2. Since, in the original Gaussian Splatting training pipeline, there is no restriction to the elongation of each Gaussian along the camera view, we add to the cost function the following scaling regularization term:

$$\mathcal{L}_{iso} = \sum_{i=1}^{|\mathcal{G}|} ||\mathbf{s}_i - \bar{\mathbf{s}}_i \cdot \mathbf{1}||_1, \tag{3.19}$$

which penalizes the scaling parameters by their difference from the mean $\bar{\mathbf{s}}_i$. It has been shown in Davison et al. (2024) that this improves the performance of the tracking module. As mentioned previously, during the execution of the mapping module, the camera poses are also refined, mainly to alleviate possible trajectory drifts during the tracking module. The final optimization objective has the following form:

$$\min_{\mathbf{T}_{CW},\mathcal{G}} \{\mathcal{L}_{pho} + \lambda_1 \mathcal{L}_{geo} + \lambda_2 \mathcal{L}_n + \lambda_3 \mathcal{L}_{iso}\}, \tag{3.20}$$

where the coefficients λ_i weight the contribution of each cost function term. We follow the same training pipeline described in section 2.2.

3.3.2 Analytic camera pose Jacobian

Since we are optimizing with respect to the camera poses, it is necessary to compute the derivative of the normal consistency, which we recall as:

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N}), \tag{3.21}$$

where ω_i is the alpha blending weight as stated in section 2.2.2. Similarly as in the tracking module we have to compute the Jacobian of the render normal map in the camera coordinates $\mathbf{n}_c$ with respect to the camera extrinsic parameters $\frac{\partial \mathbf{n}_c}{\partial \tau}$:

$$\begin{aligned}
\frac{\partial \mathbf{n}_c}{\partial \tau} &= \lim_{\tau \to 0} \frac{\exp(\tau^\wedge)\mathbf{n}_c - \mathbf{n}_c}{\tau} && (3.22)\\
&= \lim_{\tau \to 0} \frac{(\mathbf{I} + \tau^\wedge)\mathbf{n}_c - \mathbf{n}_c}{\tau} && (3.23)\\
&= \lim_{\tau \to 0} \frac{\tau^\wedge \mathbf{n}_c}{\tau} && (3.24)\\
&= \lim_{\tau \to 0} \frac{\theta^\times \mathbf{n}_c + \rho}{\tau} && (3.25)\\
&= \lim_{\tau \to 0} \frac{-\mathbf{n}_c^\times \theta + \rho}{\tau} && (3.26)\\
&= \begin{bmatrix} \mathbf{I} & -\mathbf{n}_c^\times \end{bmatrix}, && (3.27)
\end{aligned}$$

where the notations are the same as what we have seen in 3.2.1.

Chapter 4

Results and discussion

In this chapter, we will describe in detail the set of experiments that we have conducted with the methodology explained in the previous Chapter. First, in Section 4.1 we will make a brief description of the two kinds of metrics (spatial and appearance-based) we have used to evaluate the performance of our algorithm, in comparison with state-of-the-art methods. Then, in Section 4.2, we introduce the datasets we have chosen to evaluate the methodology. Then, in Section 4.3, we will provide some of the technical details related with the implementation. Finally, in Section 4.4, we discuss and analyze the results obtained in the experiments.

4.1 Metrics description

Let us briefly review some of the standard metrics for evaluating the quality of the estimated trajectories and of the novel view synthesis. These are two very different aspects, the first one being about spatial closeness (between an estimated trajectory and a ground-truth trajectory) and the second one being about appearance closeness of the rendered images with respect to the ground-truth ones.

4.1.1 Evaluation of the quality of the trajectory

First, we describe how to evaluate the performance of trajectory estimation. It is important to remember that, for this kind of SLAM problems, there is a fundamental translation and orientation ambiguity originated by the freedom in choosing the global coordinate framework definition. In the monocular case, there is also an ambiguity about the *scale* of the scene and of the trajectory, because any 3D reconstruction done with a monocular camera can only be correct *up to a scale*. Hence, to perform the evaluation in a way that these ambiguous factors do not interfere, it is important to *align* the reference (ground-truth) trajectories and the trajectories estimated by our system before performing any comparison, to handle this unknown transformation between the two trajectories. We can formulate the problem as follows: Assume that there exist two corresponding point sets $\{\mathbf{p}_i\}$ and $\{\mathbf{p}'_i\}$, $i = 1...N$ in $\mathbb{R}^3$, such that they are related by an unknown Euclidean transform:

$$\mathbf{p}'_i = s\mathbf{R}\mathbf{p}_i + \mathbf{t} + \boldsymbol{\nu}_i, \tag{4.1}$$

where $\mathbf{R}$ is a 3D rotation matrix represented as an orthogonal matrix, $\mathbf{t}$ is a 3D translation vector, $\boldsymbol{\nu}_i$ is a noise vector and s is a scale correction factor. Since it is possible to solve independently for $s, \mathbf{R}$ and $\mathbf{t}$, we will assume $s = 1$ for simplicity as in D.W. Eggert (1997). Solving this problem typically requires minimizing the least squares error:

$$\min_{s,\mathbf{R},\mathbf{t}} \sum_{i=1}^{N} ||\mathbf{p}'_i - s\mathbf{R}\mathbf{p}_i - \mathbf{t}||^2. \tag{4.2}$$

To compute the solution, we use the algorithm developed by Horn and described in D.W. Eggert (1997).

To come back to our problem, we use:

- the translations components along the reference trajectory $\{\mathbf{T}^*_{WC_j}\}_{j=1}^N$
- the translations components of the estimated trajectory $\{\hat{\mathbf{T}}_{WC_j}\}_{j=1}^N$

to compute the unknown Euclidean transformations through Eq. 4.2 and align the

two trajectories, producing ground truth positions $\mathbf{T}^*_{WC_j}$ at timestep j and aligned estimated positions $\bar{\mathbf{T}}_{WC_j}$:

$$\bar{\mathbf{T}}_{WC_j} \triangleq e(\hat{\mathbf{T}}_{WC_j}; s, \mathbf{R}, \mathbf{t}),$$

where the function e applies the Euclidean transformation parameterized by $s, \mathbf{R}, \mathbf{t}$ to its arguments. From these, we can compute the Absolute Pose Error (APE) and the Relative Pose Error (RPE) defined as follows.

Absolute Pose Error

The Absolute Pose Error is a metric for assessing the *global* consistency of a SLAM trajectory. For each timestamp j, consider the ground truth camera pose $\mathbf{T}^*_{CW_j}$ and the aligned and estimated camera pose $\bar{\mathbf{T}}_{CW_j}$, then the absolute relative pose at timestamp i is defined as the product:

$$\mathbf{E}_j = (\bar{\mathbf{T}}_{CW_j})^{-1}\mathbf{T}^*_{CW_j}. \tag{4.3}$$

Note that $\mathbf{E}_j$ is a $3D$ rigid transform. We consider only the translation part $\text{trans}(\mathbf{E}_j)$ and compute the corresponding Root Mean Square Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{j=1}^{N} ||\text{trans}(\mathbf{E}_j)||^2}, \tag{4.4}$$

where the sum is taken over all the N frames of the sequence.

Relative Pose Error

The relative pose error (RPE) is a metric for assessing the *local* consistency of a SLAM trajectory. RPE compares the *relative poses* between pairs of frames along the estimated and the reference trajectory. This is based on the delta pose difference, which is specified, for a pair of frames i and j, as

$$\mathbf{E}_{i,j} = ((\mathbf{T}^*_{CW_i})^{-1}\mathbf{T}^*_{CW_j})^{-1}((\bar{\mathbf{T}}_{CW_i})^{-1}\bar{\mathbf{T}}_{CW_j}), \tag{4.5}$$

where, again, $\mathbf{E}_{i,j}$ is a 3D rigid transform. And just as before, we take the translational part and compute the RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i,j} ||\text{trans}(\mathbf{E}_{i,j})||^2}. \tag{4.6}$$

Note that this second metric may not be that sensitive to large local errors (e.g. due to one very low quality frame) and to the resulting drift, as it computes *pairwise errors*. The APE, on the opposite, will have all the terms of the sum in Eq. 4.4 very large after such an error occurs at some point during the experiment. We use the Python package evo Grupp (2017) to evaluate the metrics introduced above.

4.1.2 Evaluation of the novel views synthesis

Now, let us dive into some of the metrics which are the standard for the evaluation of novel view synthesis, i.e. to compare true images $\mathbf{I}_j^*$ from the video sequence and generated images $\hat{\mathbf{I}}(\mathcal{G}, \hat{\mathbf{T}}_{WC_j})$, for timesteps j, with the estimated map $\mathcal{G}$ (the set of Gaussians) and the estimated poses $\hat{\mathbf{T}}_{WC_j}$.

Peak signal-to-noise ratio (PSNR)

PSNR is a metric that measures the quality of a reconstructed image by measuring the amount ratio between the maximum possible power signal and the power of corrupting noise affecting the reconstruction. Because many signals have a very wide dynamic range, PSNR is usually expressed as a logarithmic quantity using the decibel scale. The PSNR is defined as:

$$\text{PSNR} = 10\log_{10}(L^2/\text{MSE}), \tag{4.7}$$

where L is the maximum value of a pixel, usually 255 for 8-bit images, MSE is the mean square error between the original and the reconstructed images, $\mathbf{I}_j^*$ and $\hat{\mathbf{I}}(\mathcal{G}, \hat{\mathbf{T}}_{WC_j})$. The higher, the better, meaning that the reconstructed image is close to the original. Although it is a good indicator, it is well known that PSNR may not

always perfectly correlate with how humans perceive image quality.

Structural Similarity Index (SSIM)

The Structural Similarity Index (SSIM) is a widely used metric to assess the similarity between two images. It is considered to be more aligned with human visual perception than other metrics such as Mean Squared Error (MSE) or Peak Signal-to-Noise Ratio (PSNR). SSIM is designed to reflect how humans perceive image quality, taking into account factors such as luminance masking and contrast masking. SSIM focuses on preserving the structural details of an image, recognizing that pixels have strong interdependencies, especially when they are spatially close. SSIM calculates a score between 0 and 1, where 1 indicates perfect similarity (identical images).

Learned Perceptual Image Patch Similarity (LPIPS).

LPIPS Zhang (2018) is a perceptual similarity metric designed to evaluate how similar two images appear to humans. LPIPS measures the image similarity not based on raw pixel values (like MSE or PSNR), but by comparing deep features extracted from a pre-trained neural network. These features capture higher-level semantic and perceptual information, making LPIPS better aligned with human visual judgment. It works as follows, by taking two images typically, a ground-truth image and a reconstructed image, $\mathbf{I}_j^*$ and $\hat{\mathbf{I}}(\mathcal{G}, \hat{\mathbf{T}}_{WC_j})$ in our case. Both images are passed through a pre-trained network (e.g., AlexNet, VGG, etc.) and feature activations are extracted at multiple layers. At each layer, the L_2 distance between the normalized features is computed. The contribution of each layer is weighted by a learned parameter. These weights are learned from human perceptual similarity judgments. Lower values mean a higher perceptual similarity (0 indicates identical features). It is not bounded between 0 and 1, but has often ranges between 0 and $\sim$ 1, depending on the backbone used for the feature extraction process and the image pair.
We have used the Pytorch Paszke et al. (2019) METRICS package, to evaluate the metrics introduce above, as well as the "AlexNet" network Zhang, Isola, Efros, Shechtman, and Wang (2018) as a backbone to the LPIPS metrics.

Figure 4.1: Sampled images from the Replica dataset.

4.2 Datasets

In the following, we describe the datasets we have been using for evaluating our proposed pipeline.

4.2.1 Replica dataset

Replica was introduced by Facebook Reality Labs/AI Research in 2019 to give researchers a set of 18 fully reconstructed, metrically accurate indoor environments covering apartments, offices, and hotel rooms.

Each scene provides a dense textured mesh with HDR materials, planar mirror & glass reflectors, per-triangle semantic class and instance labels, and high-fidelity lighting. This richness allows physically based rendering to generate photo-realistic RGB, depth, surface-normal, and semantic images from arbitrary viewpoints. The figure 4.1 shows some examples of the dataset.

Figure 4.2: TUM-RGBD dataset.

4.2.2 TUM RGB-D dataset

The Technical University of Munich (TUM) Computer Vision Group released this benchmark in 2012 to make a common yard-stick for RGB-D SLAM and odometry algorithms. It contains 39 handheld or robot-mounted Kinect sequences captured at full sensor resolution (640 × 480 px) and 30 Hz, accompanied by 6-DoF ground-truth poses from an eight-camera Vicon system sampled at 100 Hz.

Sequences span two main indoor venues—"fr1" office rooms and the larger "fr2" industrial hall—with variants that include slow motions for debugging, fast motions, loop closures, and people walking to test robustness to dynamics. The figure 4.2 illustrates some of the scenes in the dataset.

Parameter	Symbol	Description
Dataset		
`pcd_downsample`	N_d	Random downsample for point insertion.
`pcd_downsample_init`	N_{d_0}	Initial random downsample for point insertion.
`adaptive_pointsize`	a_p	Adaptive point size according to depth.
`point_size`	p_s	Point size value.
Calibration		
`fx, fy`	f_x, f_y	Focal length.
`cx, cy`	c_x, c_y	Principal point.
`k1, k2, k3`	k_1, k_2, k_3	Camera radial distortion parameters.
`p1, p2`	p_1, p_2	Camera tangential distortion parameters.
`width, height`	W, H	Image dimensions.
`depth_scale`	s_d	Scene global scale.
`distorted`	d_s	Whether we consider distortion or not.

Table 4.1: Description of dataset and calibration hyperparameters.

4.3 Implementation

We have conducted all our experiments on a single graphic processing unit (GPU) NVIDIA RTX A6000, on a Linux, Ubuntu operating system. We have implemented our code in PyTorch Paszke et al. (2019) and have used two main threads as described in Chapter 1, for the camera pose tracking part ("frontend") and the mapping part ("backend").

Now, we make a brief description of all the hyperparameters of our experimental setup, in Tables 4.1, 4.2, and 4.3. Table 4.1 lists the hyperparameters related to the data and to the camera. Table 4.2 focuses on the Gaussian splatting parameters. Finally, Table 4.3 gives the hyperparameters related to the Gaussians optimization. For each scene in TUM and Replica, we repeat the run *five times* so that we can analyze the robustness and repeatability. Note that there are some aleatoric factors involved in the performance of each experiment, these are:

1. for each newly selected key-frame, we randomly select some pixels in the image to insert new Gaussians;

2. additionally, the behavior of the Adaptive Density Control could change from

Parameter	Symbol	Description
Training (Initialization)		
`init_itr_num`	I_{init}	Initial mapping iteration number.
`init_gaussian_update`	U_{init}	Initial Gaussian update interval.
`init_gaussian_reset`	R_{init}	Initial Gaussian reset interval.
`init_gaussian_th`	θ_{init}	Initial Gaussian threshold for densify and prune.
`init_gaussian_extent`	ε_{init}	Initial Gaussian extension threshold.
Training (Tracking & Mapping)		
`tracking_itr_num`	I_{track}	Number of tracking iterations per frame.
`mapping_itr_num`	I_{map}	Number of mapping iterations when the window is full.
`gaussian_update_every`	U_{every}	Gaussian update interval.
`gaussian_th`	θ	Gaussian threshold for densify and prune.
`gaussian_extent`	ε	Gaussian extension threshold.
`gaussian_reset`	R	Gaussian reset interval.
`size_threshold`	T_s	Upper bound for Gaussian screen size.
`window_size`	W_s	Size of the sliding window used in the backend.
`pose_window`	W_p	Number of frames within the sliding window that refine camera poses.
`edge_threshold`	T_e	Threshold used by the image gradient mask.
`rgb_boundary_threshold`	δ_{rgb}	Threshold used by the image RGB mask.
`alpha`	α	Mediator weight between photometric and depth loss.
`kf_translation`	T_x	Max translation for key-frame selection.
`kf_min_translation`	T_{x_min}	Min translation for key-frame selection.
`kf_overlap`	γ	Overlap threshold used by Gaussian co-visibility key-frame selection.
`spherical_harmonics`	s_h	Weather to use or not spherical harmonics color parametrization.
`lr.cam_rot_delta`	Δ_{rot}	Learning rate of camera rotation delta.
`lr.cam_trans_delta`	Δ_t	Learning rate of camera translation delta.
Training (surfelSLAM)		
`normal_loss`	η_{nl}	Weight contribution of normal loss.
`kf_policy`	π_{kf}	Key-frame policy, Gaussian co-visibility or fixed interval.
`kf_interval`	I_{kf}	Key-frame interval selection.

Table 4.2: Description of Training hyperparameters

Parameter	Symbol	Description
2DGS: Optimization		
`opt_params.position_lr_init`	η_{p0}	Initial position learning rate.
`opt_params.position_lr_final`	η_{pF}	Final position learning rate.
`opt_params.position_lr_delay_mult`	μ_{pd}	Position learning rate delay multiplier.
`opt_params.position_lr_max_steps`	S_p	Max optimization steps.
`opt_params.feature_lr`	η_f	Features learning rate.
`opt_params.opacity_lr`	η_o	Opacity learning rate.
`opt_params.scaling_lr`	η_s	Scaling learning rate.
`opt_params.rotation_lr`	η_r	Rotation learning rate.
`opt_params.percent_dense`	ρ_d	Percent amount of densification times.
`opt_params.lambda_dssim`	λ_{dssim}	Weight contribution of SSIM loss.
`opt_params.densification_interval`	I_d	Densification interval.
`opt_params.opacity_reset_interval`	R_o	Opacity reset interval.
`opt_params.densify_from_iter`	I_{df}	Start densification iteration.
`opt_params.densify_until_iter`	I_{du}	End densification iteration.
`opt_params.densify_grad_threshold`	τ_g	Densify grad threshold, above split, below clone.
2DGS: Model & Pipeline		
`model_params.sh_degree`	d_{sh}	Spherical harmonics degree.
`model_params.white_background`	w_b	Weather to use white background or black background.

Table 4.3: Description of Gaussian Splatting optimization hyperparameters.

Scene	ATE (m) ↓	RPE (m/m) ↓
office0	0.203 ± 0.006	0.0300 ± 0.0006
office1	0.203 ± 0.006	0.0160 ± 0.0004
office2	0.194 ± 0.003	0.0340 ± 0.0004
office3	0.296 ± 0.010	0.0400 ± 0.0006
office4	0.560 ± 0.014	0.0490 ± 0.0007
room0	0.411 ± 0.003	0.0410 ± 0.0002
room1	0.370 ± 0.007	0.0420 ± 0.0009
room2	0.144 ± 0.006	0.0370 ± 0.0006

Table 4.4: Replica dataset: per-scene trajectory error (mean $\pm$ std). Lower is better (↓).

run to run, inevitably inducing randomness.

4.4 Discussion

In the following, we discuss the results obtained with the two datasets.

4.4.1 Replica dataset

Quantitative results

The table 4.4 shows the corresponding trajectory errors (ATE and RPE) as well as the standard deviation measurements along the five runs that we perform per scene. As we can see, the errors are in the order of the centimeter and show high stability across the five runs, reflected in the low values of the standard deviation measurements.
The table 4.5 shows the quantitative results for the appearance evaluation. In the table 4.6, we can also observe the values that we have obtained for the number of Gaussians in our map $\mathcal{G}$ and therefore for the memory allocation. These numbers are very sparse and have relatively high standard deviations, in the order of a hundred thousands Gaussians. This could be explained by the complex behavior of Adaptive Density Control and the different complexity of each scene. We can visualize more clearly this sparsity in the plots 4.3 and 4.4
As we can observe, the "office4" scene shows more sparsity than other office scenes, as well as the "room1" and "room2" scenes show more sparsity compared to the "room0"

Scene	PSNR (dB) ↑	SSIM ↑	LPIPS ↓
office0	27.367 ± 0.375	0.843 ± 0.004	0.299 ± 0.002
office1	33.817 ± 0.241	0.918 ± 0.002	0.167 ± 0.004
office2	23.812 ± 0.295	0.835 ± 0.003	0.300 ± 0.008
office3	23.032 ± 0.172	0.792 ± 0.002	0.311 ± 0.007
office4	22.698 ± 0.087	0.838 ± 0.002	0.371 ± 0.004
room0	21.630 ± 0.124	0.679 ± 0.002	0.426 ± 0.007
room1	21.348 ± 0.264	0.732 ± 0.005	0.421 ± 0.010
room2	24.295 ± 0.277	0.814 ± 0.003	0.303 ± 0.012

Table 4.5: Per-scene appearance and memory metrics (mean ± std). Arrows: ↑ higher better, ↓ lower better.

Scene	N_{Gauss} ↓	Mem./Gauss [MB] ↓
office0	$106\,980.000 \pm 6039.276$	6.529 ± 0.368
office1	$94\,773.000 \pm 6735.839$	5.784 ± 0.411
office2	$144\,523.000 \pm 10\,011.585$	8.821 ± 0.611
office3	$88\,102.000 \pm 2464.928$	5.377 ± 0.150
office4	$87\,143.000 \pm 14\,161.640$	5.319 ± 0.864
room0	$108\,500.000 \pm 6654.528$	6.622 ± 0.406
room1	$115\,837.000 \pm 15\,215.805$	7.070 ± 0.928
room2	$106\,958.000 \pm 6259.324$	6.528 ± 0.382

Table 4.6: Per-scene appearance and memory metrics (mean ± std). Arrows: ↑ higher better, ↓ lower better.

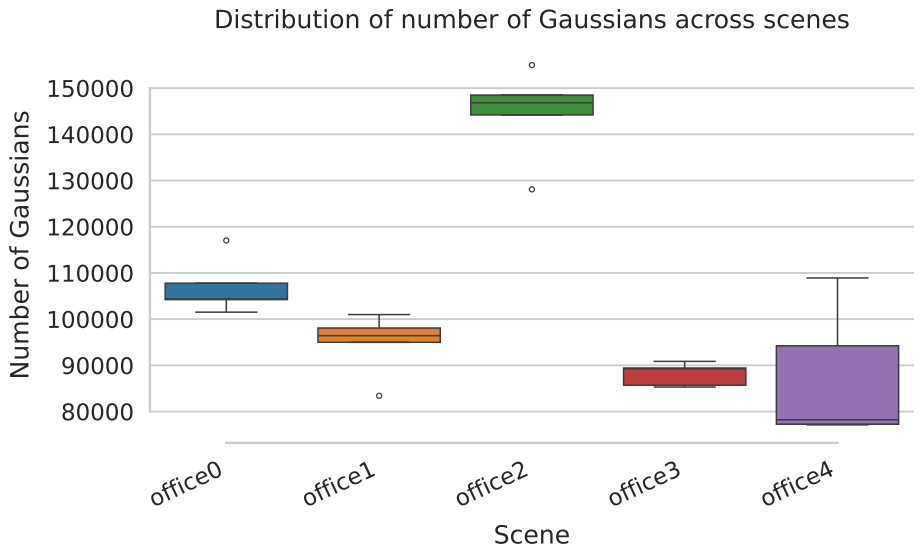

Figure 4.3: Distribution of number of Gaussians across the offices scenes.

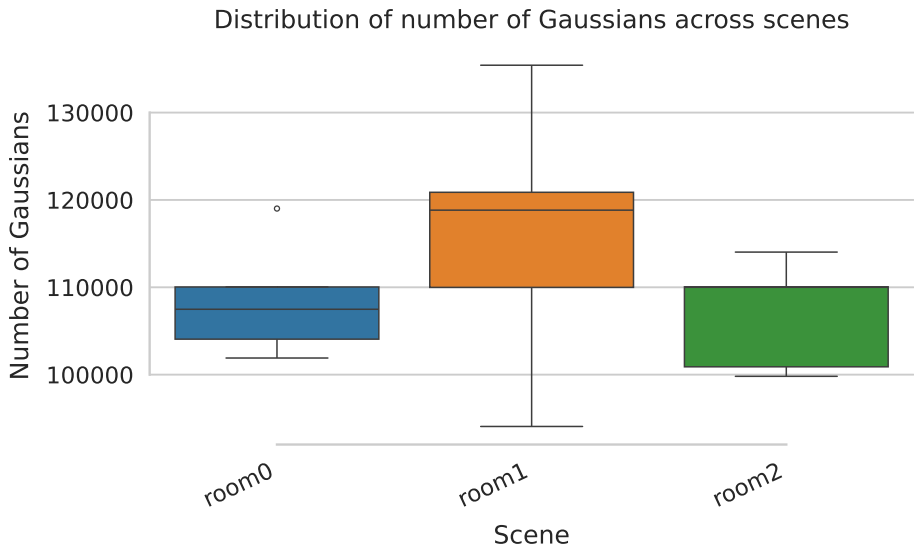

Figure 4.4: Distribution of number of Gaussians across the rooms scenes.

scene. It is important to highlight the distinction between the "offices" and "rooms" scenes, because they show different distributions of structures, textures, objects and sizes.

Now, let us analyze the best and worst scenes for the ATE metric. For the "office2" scene, we obtain the best ATE results with an error of 0.195m. Since tracking and mapping are unified through the same world representation $\mathcal{G}$, it is natural to establish a correlation between the spatial precision and the appearance quality. This is reflected through a relatively good appearance result of 23.812(dB) in PSNR and in the little sparsity in the distribution of number of Gaussians compared with the rest of scenes. The plot from Figure 4.5 shows the projections to the xy axis of the ground-truth trajectory and the estimated one, with colors indicating the level of error, as well as the confidence interval of 99% shaded in gray to illustrated the sparsity of the estimated trajectories across different runs.

For the worst result, we have the "office4" with an ATE value of 0.561m. As we have mentioned it above, this scene has high variance in the distribution of number of Gaussians, showing the importance to have a stable map for robust tracking.

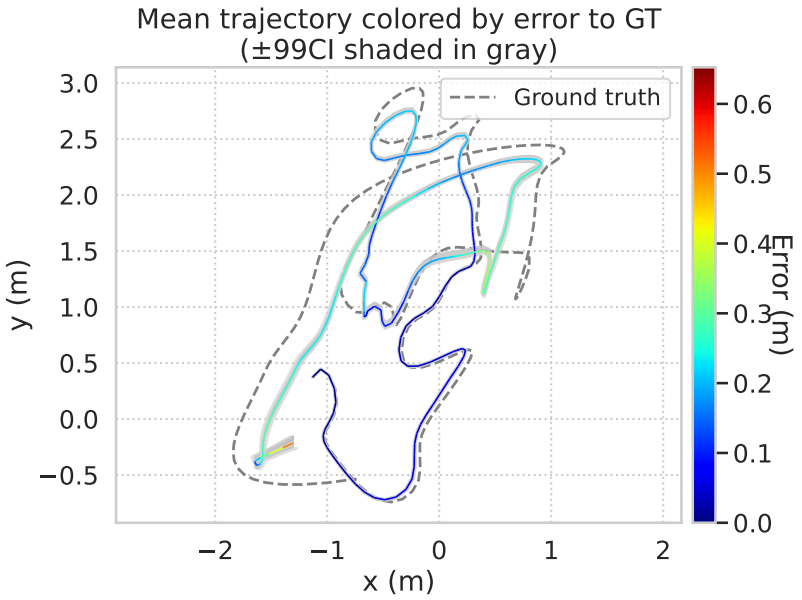

Figure 4.5: "office2" ground-truth trajectory vs. estimated mean trajectory and 99% confidence interval shaded in gray.

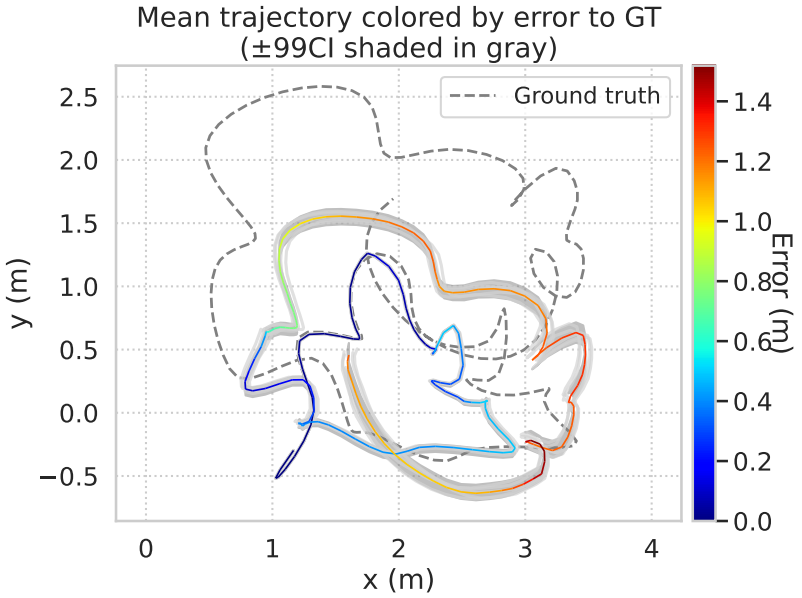

Figure 4.6: "office4" ground-truth trajectory vs. estimated mean trajectory and 99% confidence interval shaded in gray.

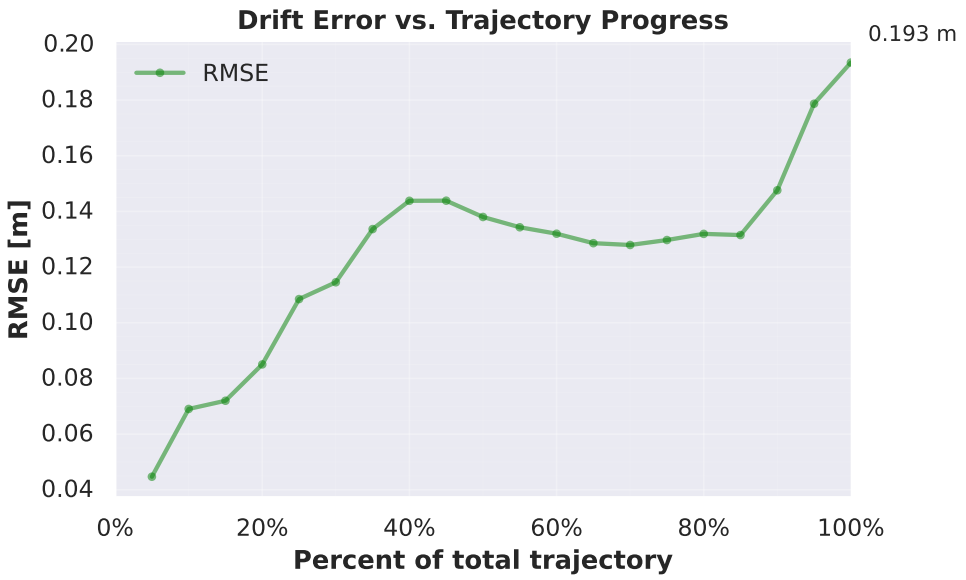

Figure 4.7: Drift error along the trajectory progress at "office0" of Replica dataset.

Similarly as in the previous case, the plot 4.6 shows the ground-truth and estimated trajectories, as well as the 99% confidence intervals. The plot shows that the two trajectories are not well aligned.

A natural behavior of visual odometry systems is the accumulated drift error along the trajectory progress, and we can visualize this phenomena in figure 4.7. At each relative pose estimation, the tracking module adds up some amount of error, and eventually this can lead to significant deviations.

Qualitative results

On average, we obtain high quality rendering results, as illustrated in Figure 4.8. However, we would like to highlight some phenomena we have observed. First, the scene with the highest PSNR measurement is the scene "office1" which we can observe in the second row of figure 4.8. Note that it has black and white as predominant colors, which we think make easier the achievement of high performance by the 2DGS appearance optimization process.

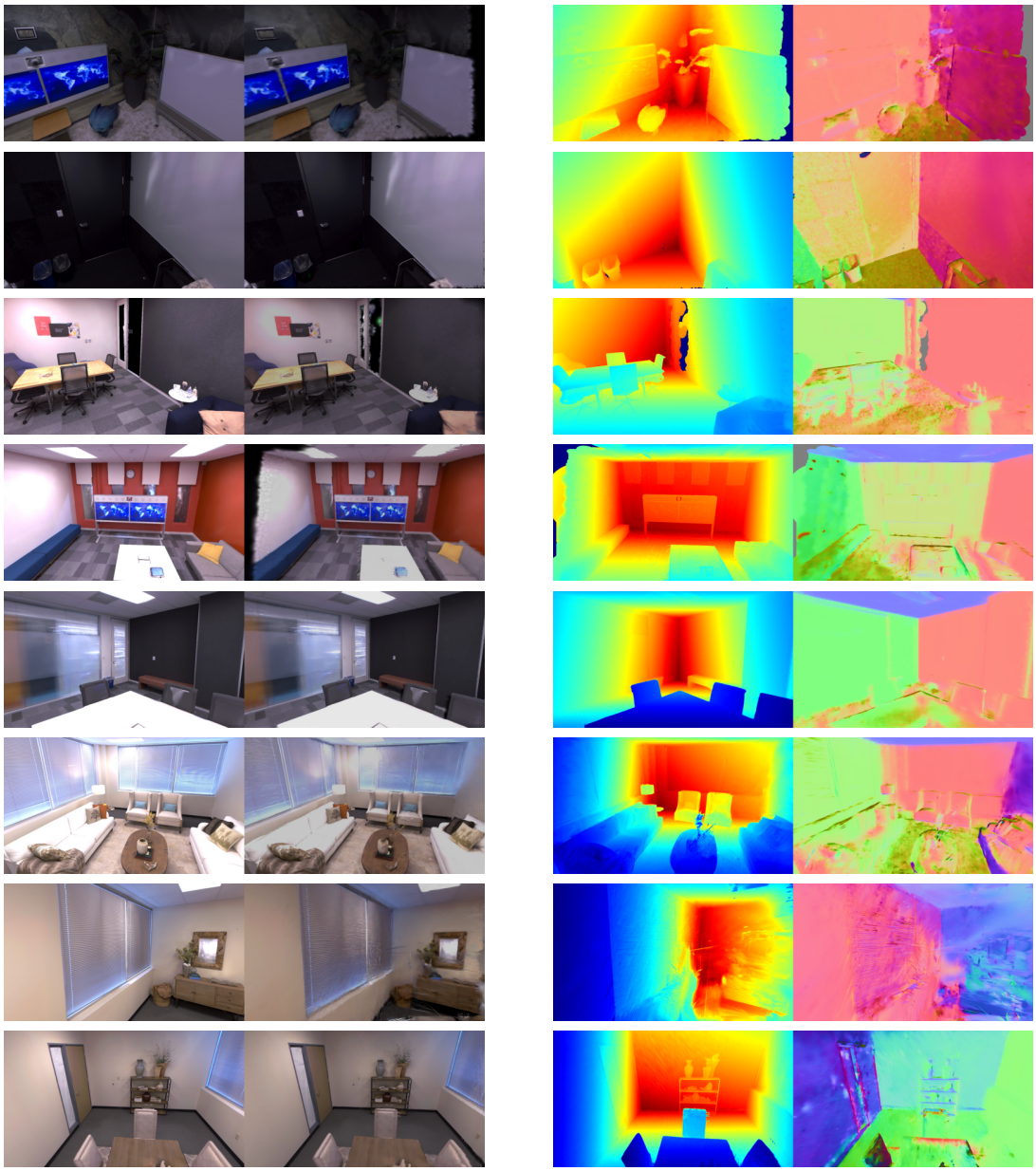

Figure 4.8: Qualitative assessment on Replica dataset. From left to right: Ground truth image, Rendered image, Rendered depth image, Rendered normal image.

Scene	ATE (m) ↓	RPE (m) ↓
desk	0.776 ± 0.050	0.059 ± 0.043
xyz	0.009 ± 0.001	0.004 ± 0.001
office	0.696 ± 0.565	0.032 ± 0.052

Table 4.7: TUM dataset: per-scene trajectory-error metrics (mean $\pm$ std). Lower is better (↓).

Even though $2DGS$ can capture light effects, we note that it struggles with more complex phenomena such as the Moiré effect present in the shutters of "room0" show in the sixth row of the figure 4.8.

$2DGS$ is very good at capturing shadows and complex textures as we can observe from the figure 4.8, but it can fail in representing tiny objects like wall connections, specially if they just appear in a small number of frames.

4.4.2 TUM RGB-D dataset

Quantitative results

Similarly to what we have seen for the Replica dataset, we have conducted a set of experiments on the TUM RGB-D dataset. The table 4.7 shows the corresponding trajectory errors as well as the standard deviation measurements on the five runs that we have performed per scene. Similarly as in the case of Replica dataset, the errors are in the centimeter order, but this time we observe more dispersion of the errors across the five runs, which was to be expected because of the real conditions of these scenes. We report both the absolute trajectory error (ATE) and the relative pose error (RPE).

The tables 4.8 and 4.9 show the quantitative results for the appearance and memory evaluation, respectively. In contrast with the Replica dataset, which is a synthetic dataset, here we obtain lower values for PSNR and SSIM and higher values for LPIPS. We can observe again that the scene with the better performance in trajectory estimation has also less variance in the number of Gaussians and memory compared with

Scene	PSNR (dB) ↑	SSIM ↑	LPIPS ↓
desk	12.545 ± 2.991	0.426 ± 0.144	0.592 ± 0.142
xyz	15.589 ± 0.068	0.670 ± 0.004	0.322 ± 0.006
office	16.841 ± 0.497	0.616 ± 0.039	0.465 ± 0.046

Table 4.8: TUM dataset: per-scene appearance metrics (mean ± std). Arrows: ↑ higher is better, ↓ lower is better.

Scene	N_{Gauss} ↓	Mem./Gauss [MB] ↓
desk	$102\,432.000 \pm 61\,144.082$	6.252 ± 3.731
xyz	$34\,860.000 \pm 1607.949$	2.128 ± 0.098
office	$75\,587.000 \pm 13\,840.304$	4.613 ± 0.844

Table 4.9: TUM dataset: per-scene memory metrics (mean ± std). ↓ lower is better.

the rest of the scenes. This difference can be visualized more clearly in the figure 4.9. The figures 4.10, 4.11, 4.12 show the trajectory errors for the three scenes from TUM-RGBD where we perform the evaluation. Similarly as with Replica, we show the confidence interval of 99% percent shaded in gray. In the case of the scene "fr1_desk" depicted in figure 4.10, the system shows robustness across the different runs, while in the case of the scene "fr2_xyz", depicted in figure 4.11, it shows some variation, despite being the scene with the best trajectory estimation according to its Absolute Trajectory Error measurement. This could be explained because of the loops present in the scene and the absence of a closed loop detection module.

Qualitative results

As was expected, there is a drop in appearance quality with respect to the Replica dataset, because of the real-scenario nature of the TUM-RGBD dataset. In this case, we have hand-held camera motion and complex photometric patterns and effects. Despite the inherent complexity to this scenario, we can observe from figure 4.13 that it recovers effectively the photometric appearance. Also, the depth maps look very clear, showing that we can achieve high quality geometric reconstruction. Contrastingly, normal maps look less defined and crisp. In a future work we would like to incorporate regularization terms to enhance the normals alignment, specially in the

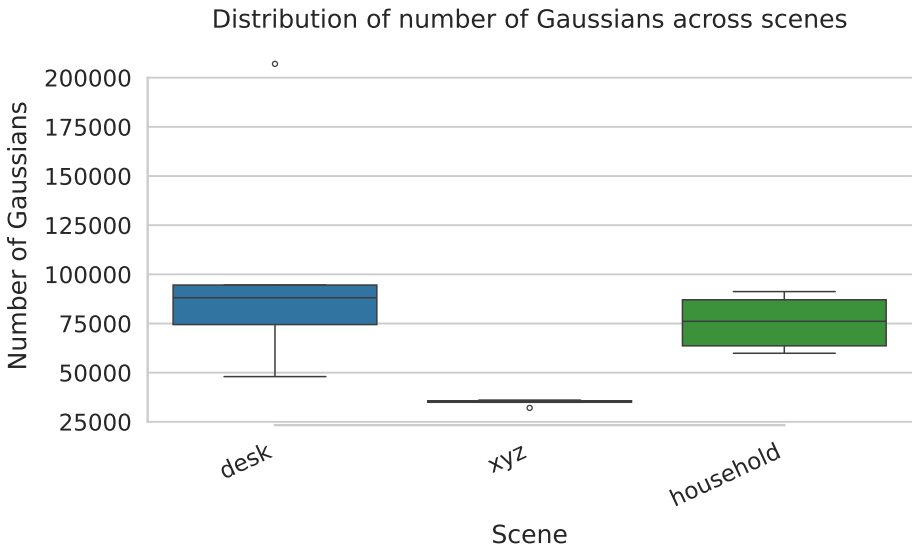

Figure 4.9: Distribution of number of Gaussians across TUM datasets

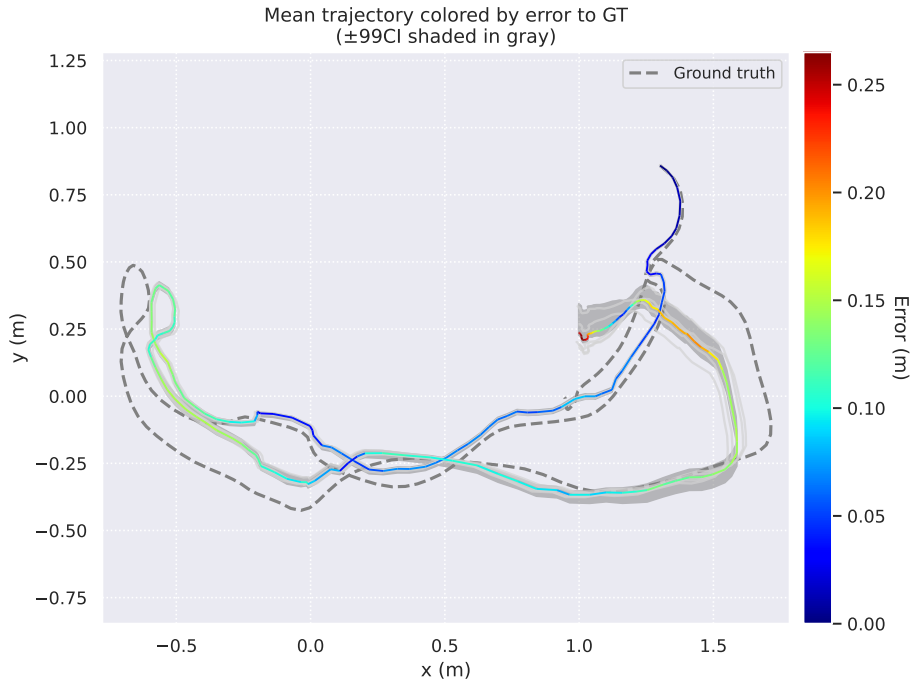

Figure 4.10: "fr1_desk" ground-truth trajectory vs. estimated mean trajectory and 99% confidence interval shaded in gray.

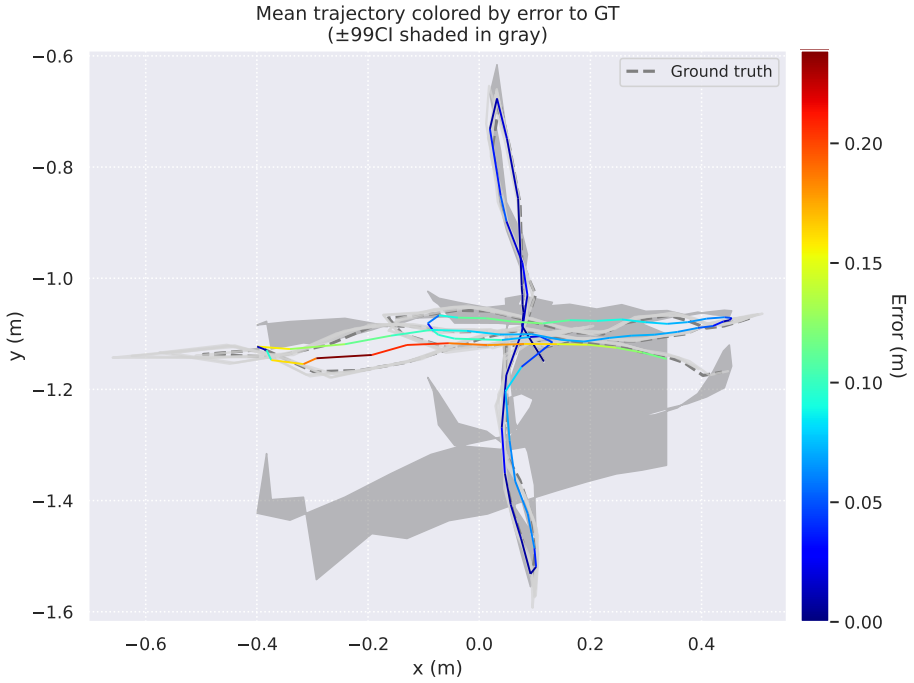

Figure 4.11: "fr2_xyz" ground-truth trajectory vs mean estimated trajectory and 99% confidence interval shaded in gray.

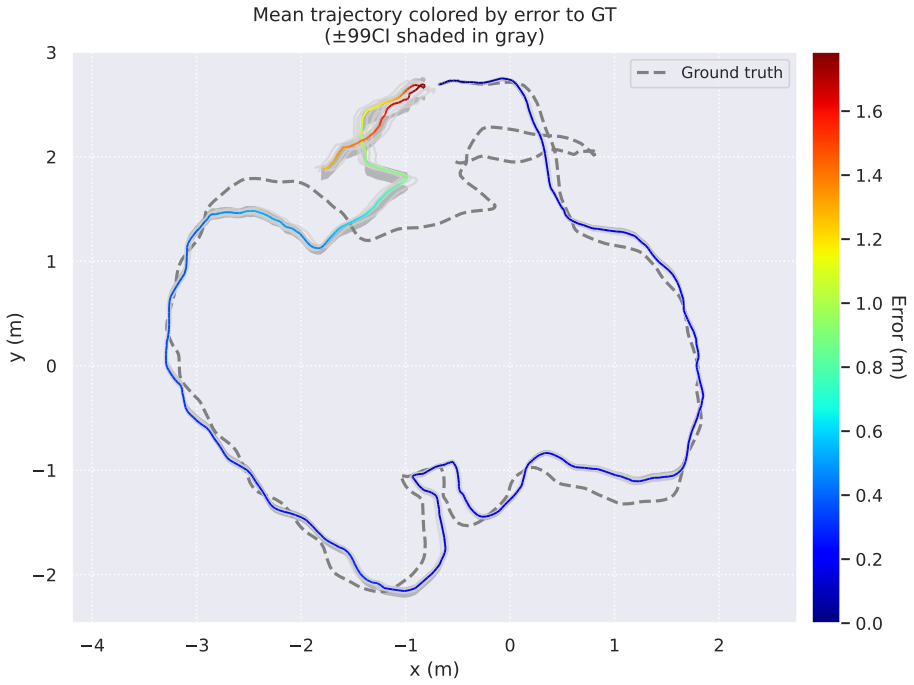

Figure 4.12: "fr3_office" ground-truth trajectory vs mean estimated trajectory and 99% confidence interval shaded in gray.

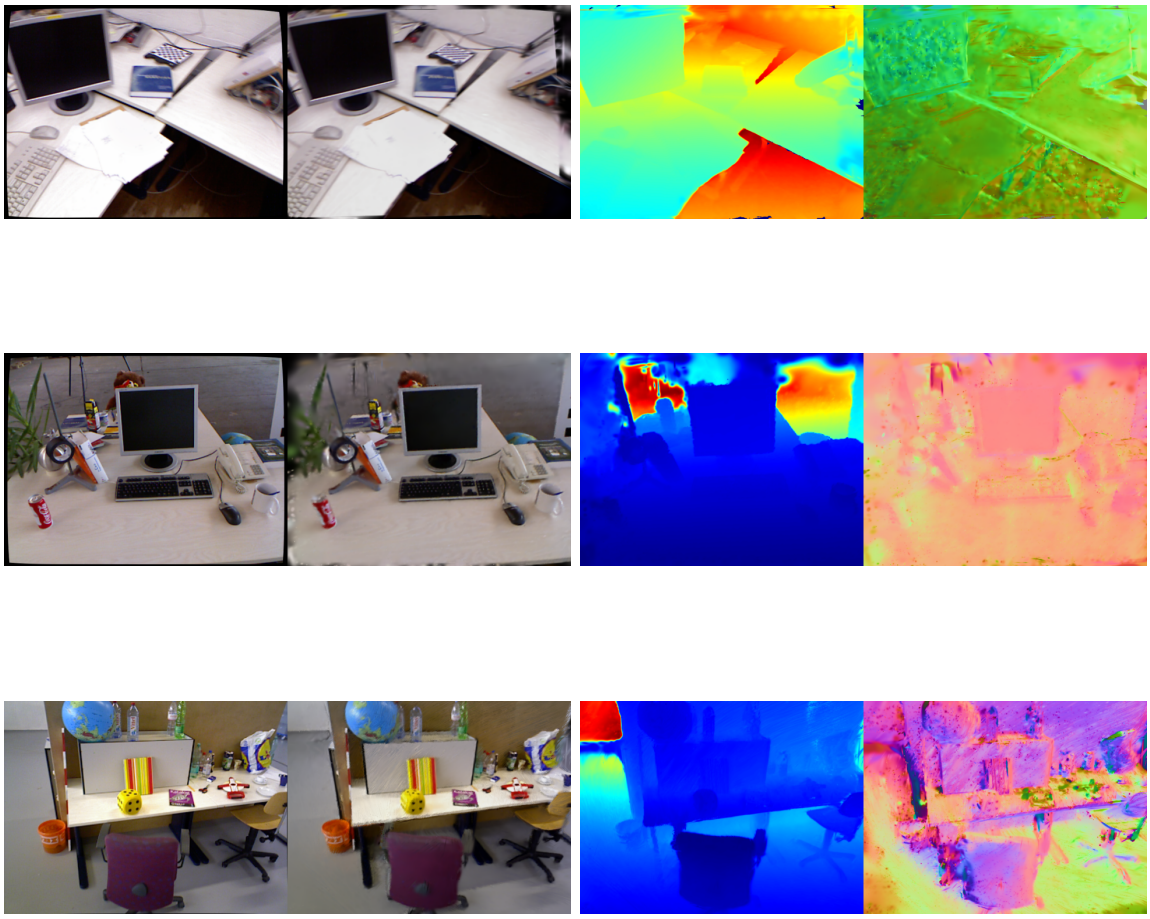

Figure 4.13: Qualitative assessment on TUM-RGBD dataset. From left to right: Ground truth image, Rendered image, Rendered depth image, Rendered normal image.

borders, where there is an ambiguity on how to properly orientate normals.

4.5 Baseline comparisons

Two recent methods closely related to our work are Gaussian Splatting SLAM Davison et al. (2024) and Photo-SLAM H. Huang (2023). Both systems aim to recover accurate camera trajectories and dense photometric maps, but they differ in their representations and trade-offs. We briefly summarize their key results before comparing them to our 2D Gaussian splatting (2DGS) method.

3D Gaussian Splatting SLAM (3DGS SLAM) uses a fully 3-D Gaussian map for tracking and rendering. On the TUM RGB-D benchmark, their camera tracking results show that 3DGS SLAM achieves an ATE-RMSE below 0.8 cm across monocular and RGB-D sequences, outperforming classical methods (iMAP, NICE-SLAM, Vox-Fusion, ESLAM, Point-SLAM) and even matching several RGB-D systems. Their map is very compact: the memory analysis reports that 3DGS SLAM requires 2.6 MB for monocular SLAM and 3.97 MB for RGB-D, whereas NICE-SLAM needs 101.6 MB and Co-SLAM uses 1.6 MB. For novel-view rendering on the Replica dataset 3DGS SLAM achieves an average PSNR of 37.50 dB, SSIM$\sim$0.960 and LPIPS$\sim$0.070 with a rendering FPS of $\sim$769, greatly exceeding Nice-SLAM (24.42 dB PSNR, 0.54 FPS), Vox-Fusion (24.41 dB PSNR, 2.17 FPS) and Point-SLAM (35.17 dB PSNR, 1.33 FPS). 3DGS SLAM thus provides state-of-the-art photometric quality and fast rendering, but it uses a complex 3-D representation and an optimization loop that operates on thousands of 3-D Gaussians.

Photo-SLAM introduces a hyper-primitives map that stores explicit ORB features together with implicit spherical-harmonic coefficients and uses a neural decoder to learn photometric appearance. On the synthetic Replica dataset, their system achieves PSNR = 34.958 dB, SSIM = 0.942 and LPIPS = 0.059 while rendering 1084 FPS using $\sim$5 GB memory; these numbers are the best or second best across all com-

pared methods, outperforming Nice-SLAM, ESLAM, Co-SLAM and Go-SLAM. On the real-world TUM RGB-D benchmark, Photo-SLAM yields PSNR values in the range 20.5–21.6 dB with SSIM $\sim$ 0.73–0.78 and LPIPS $\sim$ 0.15–0.25, while achieving ATE-RMSE around 0.85–0.98 cm for monocular sequences and $\sim$ 0.35 cm for RGB-D sequences. The method therefore obtains more photorealistic renderings than NICE-SLAM and our 2DGS, but it relies on a hybrid explicit–implicit representation and requires training a neural decoder per scene.

$2DGS$ formulation uses a simpler representation: each 2-D Gaussian lies on a plane defined by its normal rather than occupying 3-D space. This representation leads to a different set of trade-offs:

Memory and efficiency: Because only 2-D Gaussians are stored, our map remains compact. For the Replica scenes we use between $\sim$87k and $\sim$144k Gaussians, each storing 6.5–8.8 MB of appearance parameters (Table 4.6). The total memory footprint (mean $\pm$ std.) is therefore on the order of 5–8 MB per scene, comparable to 3DGS SLAM's 2.6–3.97 MB and significantly less than the $\sim$5 GB used by Photo-SLAM. Our runtime is not yet optimized for real-time rendering, but the simple representation lends itself to efficient GPU splatting.

Camera tracking: On the Replica dataset our method achieves ATE errors around 0.20–0.56 m and RPE errors around 0.03–0.05 m (Table 4.4). On the TUM sequences the ATE errors lie in the centimeter range (0.009–0.776 m), though with higher variance due to real-world motion (Table 4.7). While these errors are larger than the sub-centimeter ATE reported by 3DGS SLAM and Photo-SLAM, our system does not yet include loop-closure.

Photometric quality: Our 2DGS method yields PSNR values in the range 21–34 dB on the Replica dataset (Table 4.5) and $\sim$12–17 dB on TUM (Table 4.8). These values are higher than Nice-SLAM ($\sim$24.4 dB) but below Photo-SLAM's 34.96 dB

and 3DGS SLAM's 37.50 dB on Replica. The main limitation arises from representing scene geometry with planar Gaussians; complex 3-D structures and occlusions cannot be fully captured, leading to lower PSNR and higher LPIPS. Nonetheless, the qualitative results in Figures 4.8 and 4.13 show that our system faithfully recovers color and shading and can model subtle light effects.

In summary, 3DGS SLAM and Photo-SLAM currently achieve state-of-the-art accuracy and photorealism thanks to 3-D splatting and hyper-primitives combined with neural decoding. 2DGS method favors simplicity and memory efficiency, resulting in competitive appearance quality compared to early neural-field SLAM systems (e.g., Nice-SLAM) but lagging behind 3DGS SLAM and Photo-SLAM in terms of PSNR and ATE. Future work will address these limitations by incorporating loop closures, and improved appearance modeling.

4.6 Evaluation on dynamic scenes

We also evaluate our approach on the "balloon" scene from the Bonn RGB-D Dynamic Dataset Palazzolo, Behley, Lottes, Giguère, and Stachniss (2019), which follows a format similar to TUM-RGBD but includes dynamic entities. Running the method without any modification results in a noticeable drift in tracking, as shown in Figure 4.14, with an ATE of approximately 30 cm. Likewise, the reconstructed appearance and geometry exhibit poor quality, as illustrated in Figures 4.15 and 4.16, yielding a PSNR of 20.5, an SSIM of 0.72, and an LPIPS of 0.40. These results indicate that the method struggles to handle dynamic scenes effectively, highlighting an important direction for future research.

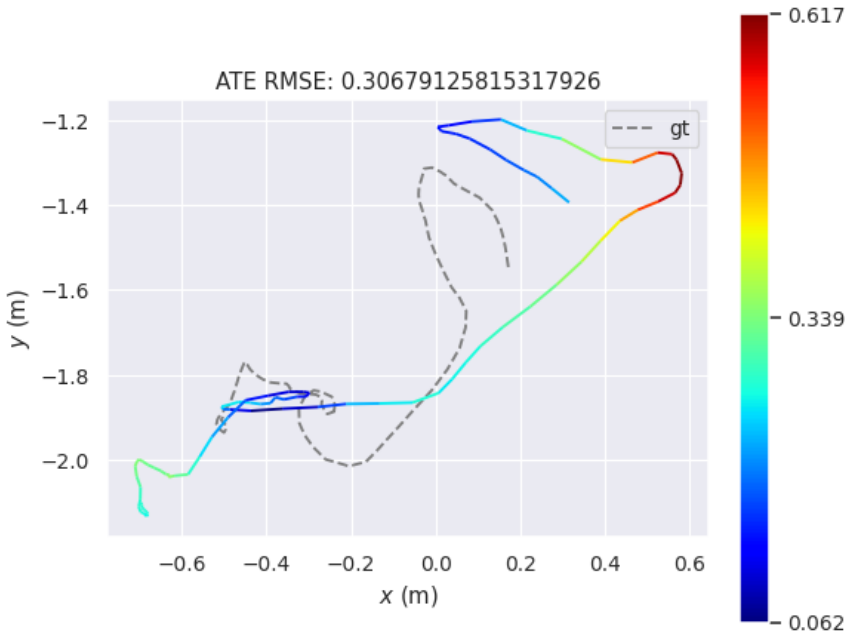

Figure 4.14: Trajectory estimation of "ballon" scene of Bonn RGB-D Dynamic Dataset.

Figure 4.15: "Ballon" scene of Bonn RGB-D Dynamic Dataset: Original vs rasterized image comparison.

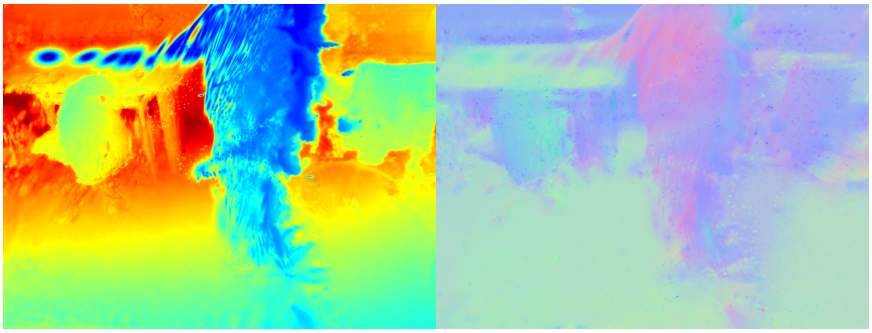

Figure 4.16: "Ballon" scene of Bonn RGB-D Dynamic Dataset: Depth-map vs Normal-map rasterizations.

Chapter 5

General Conclusions

In this thesis, we have addressed the problem of Visual Simultaneous Localization and Mapping (SLAM), a central task in computer vision and robotics. We have explored the use of a recent view-synthesis technique, *2D Gaussian Splatting*, which allows to achieve real-time rendering performance. We have adopted it as the core map representation in the SLAM pipeline presented in Chapter 3. We have evaluated our method on the Replica Straub et al. (2019) and TUM RGB-D Sturm et al. (2012) datasets, and discussed the results in detail in Chapter 4.
Our experiments have demonstrated several key findings:

- **Tracking robustness.** The proposed map representation alone—without additional mechanisms such as loop closure—already provides strong tracking performance. On Replica and TUM RGB-D, we have obtained trajectory errors on the order of centimeters, achieving results that are competitive with state-of-the-art methods in several benchmark scenarios. This highlights the value of a unified representation for both tracking and mapping.

- **Appearance reconstruction.** The method is capable of modeling complex appearance patterns, shadows, and textures, producing high-quality novel views. Nevertheless, fine geometric details and more complex photometric phenomena, such as Moiré effects, remain challenging to handle. This indicates that, while 2D Gaussian splatting is effective in capturing global appearance, additional

regularization or hybrid representations may be necessary to improve fidelity at finer scales.

- **Dataset differences.** Results vary significantly between synthetic (Replica) and real-world (TUM RGB-D) datasets. While Replica sequences produce higher PSNR and SSIM scores with lower LPIPS values, the TUM dataset reveals more drift and larger variance due to its handheld camera motion and photometric complexities. This suggests that real-world conditions introduce non-trivial challenges that synthetic benchmarks alone cannot capture.

- **Comparison with related work.** In comparison with 3D Gaussian Splatting SLAM and Photo-SLAM, our method trades off some photometric fidelity and trajectory accuracy for simplicity and memory efficiency. While 3DGS and Photo-SLAM achieve higher PSNR and lower ATE by leveraging 3D or hybrid representations, our 2DGS formulation remains more compact and lightweight, which makes it attractive for scenarios with limited computational resources.

- **Dynamic scenes.** A critical limitation of our system emerges in the presence of dynamic entities. On the Bonn RGB-D Dynamic Dataset, the method exhibits significant tracking drift and degraded appearance quality, achieving an ATE of around 30 cm and lower perceptual metrics (PSNR of 20.5, SSIM of 0.72, LPIPS of 0.40). These results confirm that explicitly addressing dynamic objects is essential for extending the applicability of the approach to real-world robotics.

Future work. Building on these findings, several research directions arise naturally. First, incorporating *loop closure* or global optimization strategies could mitigate long-term drift. Second, *appearance modeling* may be improved with additional regularization terms (e.g., on normals and borders) or hybrid explicit-implicit representations that better capture fine details and complex lighting. Third, addressing *dynamic environments* through semantic segmentation, motion modeling, or uncertainty-aware Gaussian maps is a crucial step towards robust real-world deployment. Finally, optimizing runtime for real-time rendering remains an open challenge, though the compactness of the representation provides a promising starting point.

References

Davison, A. J., Kelly, P. H., Matsuki, H., & Murai, R. (2024). Gaussian splatting slam. *arXiv preprint arXiv:2312.06741*. Retrieved from https://arxiv.org/abs/2312.06741

Davison, A. J., Reid, I. D., Molton, N. D., & Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Retrieved from https://doi.org/10.1109/tpami.2007.1049

D.W. Eggert, A. L. . R. F. (1997). Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Springer, Machine Vision and Applications.*.

Engel, J., Schöps, T., & Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. *Lecture Notes in Computer Science, Computer Vision – ECCV 2014*. Retrieved from https://doi.org/10.1007/978-3-319-10605-2_54

enVista Corporation. (2025). *Improve accuracy with warehouse slam systems.* https://envistacorp.com/blog/revolutionizing-warehouse-efficiency-slam-solutions/. (Insights into integrating SLAM robots with WMS, infrastructure requirements, deployment challenges.)

Green, R. (2003). Spherical harmonic lighting: The gritty details. In *Course notes: Game developers conference.* ACM SIGGRAPH. Retrieved from https://3dvar.com/Green2003Spherical.pdf

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

Hidenobu Matsuki, A. J. D., Gwangbin Bae. (2025). 4dtam: Non-rigid tracking and mapping via dynamic surface gaussians. *arXiv preprint arXiv:2505.22859*.

Retrieved from https://arxiv.org/abs/2505.22859

Huang, B., Zehao Yu, A. C., Geiger, A., & Gao, S. (2024). 2d gaussian splatting for geometrically accurate radiance fields. *arXiv preprint arXiv:2403.17888*. Retrieved from https://arxiv.org/abs/2403.17888

Huang, H. (2023). Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras. *arXiv*. Retrieved from https://arxiv.org/abs/2311.16728

Joan Solà, D. A., Jeremie Deray. (2018). A micro lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*. Retrieved from https://arxiv.org/abs/1812.01537

Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3d gaussian splatting for real-time radiance field rendering. *arXiv preprint arXiv:2308.04079*. Retrieved from https://arxiv.org/abs/2308.04079

Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. *International Symposium on Mixed and Augmented Reality (ISMAR)*. Retrieved from https://ieeexplore.ieee.org/document/4538852

LLC, W. (2020). *How our highly-detailed maps help unlock new locations.* https://waymo.com/blog/2020/09/the-waymo-driver-handbook-mapping. (Describes mapping techniques used in Waymo's autonomous driving systems)

Meta, F. R. L. . (2019). *The story behind facebook's oculus insight technology and a positional tracking system for quest.* https://tech.facebook.com/reality-labs/2019/8/the-story-behind-oculus-insight-technology/. (Describes using vision + IMU + SLAM for inside-out tracking on Quest.)

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*. Retrieved from https://arxiv.org/abs/2003.08934

Murai, R., Dexheimer, E., & Davison, A. J. (2024). Mast3r-slam: Real-time dense slam with 3d reconstruction priors. *arXiv*. Retrieved from https://doi.org/

10.48550/arXiv.2412.12392

Mur-Artal, R., Montiel, J. M. M., & Tardós, J. D. (2015). Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*. Retrieved from https://doi.org/10.1109/tro.2015.2463671

Palazzolo, E., Behley, J., Lottes, P., Giguère, P., & Stachniss, C. (2019). ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *iros.* Retrieved from https://www.ipb.uni-bonn.de/pdfs/palazzolo2019iros.pdf

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035).

Schönberger, J. L., & Frahm, J.-M. (2016). Structure-from-motion revisited. In *Conference on computer vision and pattern recognition (cvpr).*

Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., ... Newcombe, R. (2019). The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012, Oct.). A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the international conference on intelligent robot systems (iros).*

Teed, Z., & Deng, J. (2021). Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *arXiv*. Retrieved from https://doi.org/10.48550/arXiv.2108.10869

Vincent Leroy, J. R., Yohann Cabon. (2024). Grounding image matching in 3d with mast3r. *arXiv*. Retrieved from https://arxiv.org/abs/2406.09756

Zhang, R. (2018). The unreasonable effectiveness of deep features as a perceptual metric. *arXiv*. Retrieved from https://arxiv.org/abs/1801.03924

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. In *Cvpr.*