MSc Artificial Intelligence
Master Thesis

Gaussian mapping for Evolving Scenes

by
Thies Kersten
15102483

July 14, 2025

36EC
January 6, 2025 - July 3, 2025

Supervisor:
Dr. rer. nat. Martin Oswald
Examiner:
Vladimir Yugay MSc

Second reader:
Vladimir Yugay MSc

Universiteit van Amsterdam

Contents

Abstract

Mapping systems with novel view synthesis (NVS) capabilities are widely used in computer vision, with applications in augmented reality, robotics, and autonomous driving. Most notably, 3D Gaussian Splatting -based systems show high NVS performance; however, many current approaches are limited to static scenes. Although recent work started addressing short-term dynamics (motion within view of the camera), long-term dynamics (the scene evolving through changes out of view) remain less explored. To overcome this limitation, we introduce a dynamic scene adaptation mechanism that continuously updates the 3D representation to reflect the latest changes. In addition, since maintaining geometric and semantic consistency remains challenging due to stale observations disrupting the reconstruction process, we propose a novel keyframe management mechanism that discards outdated observations while preserving as much information as possible. We evaluated Gaussian Mapping for Evolving Scenes (GaME) on synthetic and real-world datasets and found it more accurate than the state of the art.

Chapter 1

Introduction

1.1 Challenges in Mapping Evolving Environments

Visual mapping enables a machine to build a 3D representation of its surroundings based on its camera input. Over the years, visual mapping systems have advanced [26, 19], learning to handle complex scenes with increasing accuracy. For these systems to move safely and make decisions, they need a reliable understanding of their surroundings. This capability forms the backbone for various systems from self-driving cars to virtual reality glasses, where spatial understanding is crucial for navigation, planning, and interaction. Imagine a delivery robot

Figure 1.1: Examples of realistic long-term changes of a scene in the Flat dataset [29]. (left) Room before changes. (middle) Room after changes. (right) Highlighted scene changes. Colors indicate changes, red is *removed*, green is *added* and blue *changed*.

that delivers packages to an office. Over months, it is very common that the environment will change significantly. Furniture will be moved, new decorations will be added, older objects will be discarded. To have a tangible example, of how human environments change, think about the changes in the composition and content of your desk in the last month. To be able to react to those changes, it is essential that the mapping system is able to cope with these changes. Current systems for mapping environments do not model this and would either fail completely or require and entire remapping of the environment, which can be costly and time intensive.

1.2 NVS capable mapping systems

Recent mapping systems have been enhanced with novel view synthesis (NVS) capabilities [19, 11], allowing them to generate realistic, immersive views of scenes. This allows more detailed scene

Figure 1.2: Examples of long-term and short-term changes. (top) example from Flat dataset [29]. As an example, the chairs have moved and are seen in a different position after being out of the cameras view for an arbitrary amount of time. (right) Example from TUM-rgbd dynamic dataset [30]. A person is moving continuously, while being observed by the camera.

exploration and supports the creation of high-quality virtual environments. An application for this might be augmented reality. If virtual objects were to interact with the real environment, this would require a map which is consistent with the current state of the surroundings. In this context, it is desirable, that virtual objects are able to adapt interaction with the real world, just as real objects would.

1.3 Categories of changes in the environment

These NVS-capable approaches typically assume that the scene is *static* such that optimization over multiple frames is well conditioned. However, real-world environments are rarely static, but include both *short-* and *long-term* dynamics as seen in figure Fig. 1.2.

- *short-term* dynamic effects are things moving *within* view of the camera. This is usually any recorded movement or interaction with the scene. Examples could range from an object falling over or a human interacting with the environment by just occluding the scene partially or moving objects.

- *long-term* dynamics describe the scene evolving through changes happening *outside* the view of the camera Fig. 1.1. This can be the result of any change that occurs, even while not recording, such as reorganizing furniture in a room or leaving a coffee mug on a desk.

1.4 Limitations of current methods

While some of the most recent NVS approaches address short-term dynamic objects [41, 35], long-term changes remain less explored. As a result, modern reconstruction methods with NVS capabilities struggle to capture changes as the scene evolves over time, preventing them from

Figure 1.3: **Example of a partially observed change.** First a bed is first fully modeled in another room (left). After a while, the place, in which it was visible, is observed by the camera (middle). Without observing the second room after the change it is clear that the bed is fully removed, rather than just partially missing from where it was before (right).

being deployed in long-term reconstruction pipelines. There are several issues that are not addressed.

1.4.1 Time consistency

During long-term mapping, systems must continuously update the 3D representation to capture the scene's evolution and maintain reliable operation. This is especially challenging compared to classical *multi-session* mapping as changes can occur *at any time* during data collection. Considering this, the importance of addressing this problem becomes clear. Making sure a scene that is to be captured is static for the entire duration of capturing the data is beyond impractical in many cases. In addition to this, some use cases possibly *require* adapting to changes. A concrete example of this could be a digital twin of an art gallery. While it would be possible to keep it updated by recreating it every time a change occurs, just capturing and updating regions with changes with a smooth integration in an existing reconstruction would be helpful.

1.4.2 Partial observability

Objects usually don't Partially change. While current systems don't handle this, it is an intuitive concept for a human observer. Seeing a changed object partially, makes it possible to infer details about the rest of the scene without actually observing it. For instance, a bed in another room is removed, but this is only partially observed through a door. As illustrated in Fig. 1.3, it can be semantically inferred that the other part of the bed, without observing that area, must also be gone.

1.4.3 Optimization conflicts

In a long-term dynamic setting, it can occur to first observe an area with an object present and later to observe the same area without it. If such an anomaly is saved in keyframes without further precautions, the optimization becomes corrupted with conflicting data. Recently, initial methods have started to address these challenges. A first approach is presented in Panoptic Multi-TSDFs [29], building semantically consistent submaps and reasoning about changes on the level of submaps. The following works [6, 24, 28] similarly focus on object-level mapping to handle evolving scenes. However, all these methods rely on map representations that cannot easily be re-rendered, preventing their use in AR/VR or digital maps where realistic rendering is essential. Having a system capable to continuously map an environment and model the changes

Figure 1.4: **Illustration of optimization conflict.** Training image of the scene at evaluation (left). Rendered image of the scene at evaluation (right). As seen, the reconstruction can get corrupted if frames are used, in which objects are both present and not present at the same time.

without having degenerated maps caused by older and newer observations conflicting, while being capable of photorealistic NVS-synthesis, is the goal addressed in this work.

1.5 Our Approach: Gaussian Mapping for Evolving Scenes

This work addresses the challenge of long-term NVS mapping using 3D Gaussian Splatting (3DGS) in evolving scenes. The key insight of our approach is that environment changes are not random, but typically follow semantically consistent patterns. We therefore integrate semantic consistency with the inherent properties of 3DGS to efficiently detect and adapt to environment changes in the incrementally built 3DGS model. As multi-view optimization is essential for accurate 3DGS mapping, we introduce a keyframe management method that appropriately masks stale areas in keyframes to retain useful information while accounting for changes, resulting in a well-conditioned 3DGS optimization process even in evolving scenes. We make the following contributions:

- We present GaME, the first NVS-capable mapping system for long-term evolving scenes.
- A Dynamic Scene Adaptation (DSA) mechanism to incrementally update a 3DGS model.
- An efficient keyframe management strategy for accurate 3D reconstruction through changes.
- We thoroughly evaluate GaME on synthetic and real-world data, showing a performance increase of 90+% in depth and 20+% in color rendering. We release the code open-source[1].

[1]Released upon acceptance for anonymous review.

Chapter 2

Related work

2.1 3D Change Detection

The goal of change detection is to handle long-term dynamic effects, *i.e.* changes to the scene occurring *outside* in the view of the sensor. Typically, this is addressed in a *multi-session* setting. Most importantly, once a map for each session is constructed, changes can be identified through *geometric scene differencing*. For example, Meta-rooms [1] proposed a method which operates on subsequently recorded point clouds of the same room. A clustering approach is used to associate points with groups that represent changing objects in the scene. Based on this, objects are added and removed. However, this method is limited to point clouds and is focused on single-room settings only. An approach using TSDF's is introduced by Fehr et al. [5]. Here, similarly to [1], for each session, the current reconstruction of the scene is aligned and then globally compared with the new observation, to iteratively model the changes. The LiSTA and LT-mapper methods [13, 27] also frame the problem as a multisession SLAM rather than a unified input that tracks changes over time. To achieve an understanding of object-level change, this has been extended using semantic information [14], where recent trends focus on the extraction of more specialized object features, including learned shape descriptors [31, 8, 27], neural object representations [6, 42], and language embeddings [25]. Most related to us, Schmid

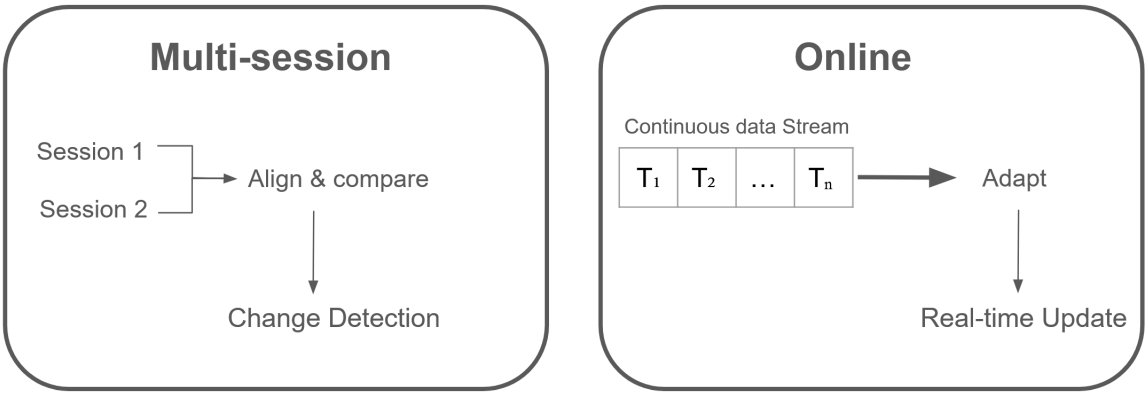

Figure 2.1: **Comparison between the multi-session and online setting**. Multi-session requires discrete captures, which are each constrained to be static. Offline processing is used to determine the changes, while the online approach continuously adapts to changes without the need for information about when to expect change.

et al.[17] recently presented a 3D Gaussian Splatting-based [12] approach by re-rendering the scene to newly collected views and using EfficientSAM [34] for 2D change detection. However, the assumption that the scene is static during each session and the offline processing is highly

limiting, as changes in human-centric and evolving scenes can occur *at any time* during the mapping process. In contrast, our mapping is designed to operate online and does not impose any limitations on scene changes.

2.2 Online Long-Term Reconstruction.

Recently, initial methods have started to address this more general online problem. A first approach is presented by Schmid et al.[29], which generates locally and semantically consistent submaps and incrementally reasons about changes on the submap level.

Fu et al.[6] propose neural object descriptors to build an object-level pose graph and detect changes in the graph configuration. This has recently been extended with $SE(3)$-equivariant descriptors [7]. Qian et al.[24] presents a frame-to-map-tracking approach, using a probabilistic update rule to detect changes at the object level, which is further extended to a variational factor-graph approach in [23]. A unified formulation of short- and long-term dynamic reconstruction is presented in Khronos [28], where objects are reconstructed locally and changes are verified using a library of rays. Nonetheless, these methods rely on map representations that cannot easily be re-rendered. In contrast, GaME provides scene reconstruction capable of real-time color and depth rendering.

2.3 Dynamic Gaussian Splatting

3D Gaussian Splatting [12] has revolutionized novel view synthesis (NVS) by enabling photorealistic, real-time rendering at over 100 FPS. Compared to neural radiance fields [20], 3DGS is significantly more memory-efficient and faster to optimize. The problem of dynamic or 4D GS has attracted wide interest. A series of works [3, 18, 33, 38] optimize a canonical set of Gaussians from the initial frame and model temporal variations through a deformation field. However, these methods are limited to short video sequences, as they cannot introduce new Gaussians after the initial frame. Another class of approaches [4, 10, 37] directly models temporal Gaussians that can exist over subsets of frames. Despite their improved flexibility, these methods require offline optimization and multi-view input, making scalability a significant bottleneck for high-quality dynamic reconstruction. In contrast, our mapping operates online using only a single RGB-D camera.

2.4 Online Gaussian Mapping

Most related to us, 3D Gaussian Splatting has sparked a wave of online RGB-D mapping methods. There are methods which solely assume *static scenes* and also methods which already start to model change.

2.4.1 Static Methods

MonoGS [19] applies Gaussian Splatting for the first time in the monocular SLAM setting, while it's also possible to use RGB-D data if available. In contrast to the original 3DGS algorithm, this approach achieves tracking by directly optimizing on the 3D Gaussians, instead of relying on an external SfM solution for poses. Splatam [11] is another solution for SLAM, only for RGB-D data. Similarly it employs 3DGS and utilizes the explicit nature of the Gaussians and directly optimizes tracking on the gaussians. However, it is limited by blurry or fast-paced input sequences. Photo-SLAM [9] uses hyper primitives based on geometric features, while also using the explicit component of those for tracking. Two more variants of using Gaussian Splatting

Method	Input type	short-term change	long-term change
MonoGS	RGB / RGB-D	✗	✗
SplaTAM	RGB-D	✗	✗
Photo-SLAM	RGB-D	✗	✗
GS-SLAM	RGB-D	✗	✗
G-SLAM	RGB-D	✗	✗
Wildgs-slam	RGB-D	✓	✗
DG-SLAM	RGB-D	✓	✗

Table 2.1: Table of related methods with similar input, mostly using explicit representations such as Gaussian splatting. Only a small number addressed short-term changes, while all of them completely neglect long-term change

are GS-SLAM and G-SLAM[36, 39]. G-SLAM for example, introduces an new efficient seeding strategy and divides the map into sub-maps. What these methods all have in common is that almost all adopt 3D Gaussians as their primary scene representation. The common denominator is that the map is based on geometrically explicit models, which is also an essential property used for GaME. While these methods perform well in static environments, they struggle in dynamic environments.

2.4.2 Dynamic Methods

More recent approaches DG-SLAM and Wildgs-slam[35, 41] tackle *short-term* dynamics within the camera's view such as people or small moving elements. DG-SLAM uses both semantic segmentation models and analysis of the residual depth of adjacent frames. The changes modeled by DG-SLAM are exclusively *inside* the view of the camera. Based on this assumption, the depth maps of frames in a sliding window are reprojected to the same frame. When comparing the reprojected depth maps from the same view, the residuals from the inconsistent depth highlight which regions are a dynamic part of the scene. This is then combined with semantic masks, to infer full semantic objects that are moving. For example, a person might move only the arms, but not legs or torso, but will be detected as dynamic by combining the depth and semantic information. Wildgs-slam defines the dynamic parts of the scene as distractors and addresses the problem of ignoring these during training. To identify which parts of the input stream are likely distractors, a DINOv2 feature extractor, which has been fine-tuned to be 3D aware is used to get a feature map per frame. The feature maps are fed into a MLP, which predicts an uncertainty map. The MLP for uncertainty maps is trained with their newly proposed Uncertainty Loss Function, which is also used for the mapping optimization. The idea of keeping a map with information about scene changes for each frame is also similar to what GaME used to manage older observations. Those two examples above show how it is plausible to use another pre-trained model to infuse a change detection system with semantics. However, all prior works remain limited in addressing *long-term* scene changes, as stale observations from outside the camera view corrupt the optimization process. In contrast, GaME is designed to robustly handle evolving environments by filtering outdated information and maintaining high rendering quality throughout reconstruction.

2.4.3 Goal of this work

Recent work has had good progress with addressing dynamic scene changes. However, they still have fundamental limitations, that hinder them to be deployed in real-world evolving

environments. More traditional methods used to cope with 3D changes used an multi-session approach, which is impractical for real-world deployment, where the scene can unpredictably change at any time. Online methods using Gaussian Splatting have strongly contributed to making mapping systems faster and better in photorealistic novel view synthesis. Only recently such approaches started to be robust regarding short-term changes only. Although first steps have been taken to solve the problem of long-term changes, there hasn't been an approach which combines the fast NVS capabilities with the ability to detect and deal with long-term changes. Our work GaME, will adress this gap exactly.

Chapter 3

Background

3.1 Background: 3D Gaussian Splatting

To make a change-robust mapping system one needs to choose the right map-representation. As discussed before, we need a scene representation that is easily manipulated and quick to render. As we choose to base our mapping system on 3D Gaussians, it's details are introduced in the following.

3.1.1 Introduction and Core Ideas

3D Gaussian splatting (3DGS) [12] is an effective method for representing 3D scenes with novel-view synthesis capability. This approach is notable for its speed, without compromising the rendering quality. 3DGS represents the scene as a collection of gaussian distributions rather than discrete points. Since some visual effects, such as reflections, have view-dependent properties, the color of a Gaussian is not just represented by an RGB value. The colors are represented as Spherical harmonics, which are a set of basis functions, which are defined on the surface of a sphere so that it is possible to model things such as specular highlights or directional shading. To render a view from this explicit model, 3D Gaussians are "splatted" by approximating a projection of them on the 2D image plane, to be then composed into an image.

3.1.2 The 3D Gaussian Representation

Gaussian Splatting explicitly represents a scene as a collection of 3D Gaussians. Each Gaussian is parameterized by mean $\mu \in \mathbb{R}^3$, covariance $\Sigma \in \mathbb{R}^{3\times3}$, opacity $o \in \mathbb{R}$, and RGB color $C \in \mathbb{R}^3$. With this representation, the Gaussian is fully defined in a 3D space. The mean is the position of each Gaussian, while the covariance matrix can fully describe the shape and orientation of a 3D Gaussian. How much each point p in a 3D space is influenced by this Gaussian is described by

$$f(p) = \sigma(o) \cdot \exp\left(-\frac{1}{2}(p-\mu)\Sigma^{-1}(p-\mu)\right), \tag{3.1}$$

Since Σ is a covariance matrix, it can be decomposed as $\Sigma = R^T D R$, where $R \in SO(3)$ and $D = \mathrm{diag}(d)$, with $d \in \mathbb{R}^3_{>0}$ being a positive diagonal matrix. Thus, the inverse is

$$\Sigma^{-1} = R^T D^{-1} R. \tag{3.2}$$

Intuitively, in Eq. (3.1) it is evaluated, how far the vector p is from μ, measured in squared Mahalanobis distance

$$d^2(x) = (x-\mu)^T \Sigma^{-1}(x-\mu). \tag{3.3}$$

The distance is Euclidean, but squared and inversely scaled along each of the Σ's eigenvectors, or principal components, by the respective eigenvalues. This value is then scaled by a factor of $-\frac{1}{2}$, the exponential function, and multiplied by an opacity factor. For example, this makes points exactly at μ have the activation of the Gaussians opacity o. The further it is placed from μ, the lower it will get. Because of the defined metric, this decay is relative to the amount of variance along the direction in which the point is moving away.

3.1.3 Spherical Harmonics for view-dependent Color

To faithfully represent a 3D Gaussian scene, it would not be enough to define an RGB-value for each Gaussian. There are several situations, such as specular highlights or directional shading, representing shiny surfaces or more complex materials, in which something wouldn't be accurately represented. Spherical harmonics are a set of functions defined on S^2, the surface of the unit sphere. One such function is defined for each of the color channels. For each channel the set of functions is summed up to approximate the desired view-dependent color. The concept is essentially comparable to a Fourier series approximation, just defined for a function on the S^2 sphere.

3.1.4 Splatting 3D Gaussians to 2D

Exactly projecting Gaussians to 2D, results in them not being Gaussians in the 2D image plane anymore. 3DGS addresses this issue with an approximation of the projected distribution in image space as a 2D Gaussian. This efficient alternative to calculating the nonlinear transformation of the original 3D distribution, is precisely the "Splatting" in Gaussian splatting. To splat a 3D Gaussian to image space, the mean $\mu_{3D} \in \mathbb{R}^3$ is projected to the corresponding $\mu_{2D} \in \mathbb{R}^2$ in image space and a covariance matrix $\Sigma^I \in \mathbb{R}^2$ for a 2D Gaussian in image space is approximated to represent what the 3D Gaussian with $\Sigma \in \mathbb{R}^3$ would look like in image space. The projection of the mean into image space is given by the equation

$$\mu_{2D} = \pi\big(P(T_{wc}\mu'_{3\mathrm{D}})\big). \tag{3.4}$$

The projection of μ_{3D} is done by first converting it to its homogeneous counterpart μ'_{3D}. With this, the given camera position is reversed with the world to camera transform $T_{wc} \in SE(3)$. Now, since the mean is in camera space (sometimes called view space), it is projected into clip space with the intrinsics matrix $P \in \mathbb{R}^{4\times4}$. Usually, points are filtered out at this moment of the rendering pipeline, if they are not in the view of the camera. After the visibility filtering, the homogenous coordinate can be transformed back into the Cartesian format and the and z component can be discarded with $\pi : \mathbb{R}^4 \to \mathbb{R}^2$ given by

$$\pi\left(\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}\right) = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \end{pmatrix}, \tag{3.5}$$

resulting in the final 2D image coordinate. Instead of doing the full projection, the resulting distribution of the 2D projection of the 3D Gaussian is approximated using a first-order Taylor expansion of the projection's Jacobian around its mean. This results in an elliptical 2D approximation of the projection. The 2D covariance Σ^I of a splatted Gaussian is given by

$$\Sigma^I = J_\mu \Sigma_{cam} J_\mu^T = J_\mu R_{wc} \Sigma R_{wc}^T J_\mu^T \ , \tag{3.6}$$

where Σ_{cam} is the covariance of the 3D Gaussian rotated into camera orientation by the rotation component $R_{wc} \in SO(3)$ of the T_{wc} in Eq. (3.4). $J \in \mathbb{R}^{2\times3}$ is a linearization of the Jacobian at μ of the non-linear part π of the projection from camera space to image space, making it most accurate around the mean of the Gaussian. This is described to be a good choice for balance of computational demand and quality in a similar preceding approach called Surface Splatting [43].

3.1.5 Composition of color and depth with Alpha-Blending

In NeRFs, for each pixel samples along a ray are taken and composed. This is also possible with the Gaussian representation, but would be highly inefficient, since for each pixel and each sample, each gaussian would have to be sampled for calculating the image. 3DGS sorts the Splatted gaussians according to depth and then samples in screen space per pixel using Alpha-Blending. The color C for one channel ch at a pixel i is influenced by m depth-ordered Gaussians and rendered as:

$$C_i^{ch} = \sum_{j \leq m} C_j^{ch} \cdot \alpha_j \cdot \prod_{k<j} (1 - \alpha_k), \tag{3.7}$$

with alpha α_j defined as

$$\alpha_j = o_j \cdot \exp(-\sigma_j), \text{ and} \tag{3.8}$$

$$\sigma_j = \frac{1}{2} \Delta_j^T \Sigma_j^{I-1} \Delta_j \ , \tag{3.9}$$

where $\Delta_j \in \mathbb{R}^2$ is the offset between the pixel coordinates and the 2D mean of a splatted Gaussian. To render a depth image, the depths of the gaussians are similarly composed to get a alpha-weighted average of the contributing gaussians per pixel.

$$D_i = \sum_{j \leq m} z_j \cdot \alpha_j \cdot \prod_{k<j} (1 - \alpha_k), \tag{3.10}$$

where z_j is the camera-space depth of the mean of the j-th Gaussian. The transmittance term $\prod_{k<j}(1 - \alpha_k)$ ensures that contributions are weighted by how much visibility remains after accounting for closer Gaussians. This formulation mirrors the color composition in Eq. (3.7), making it consistent with the rendered color.

3.1.6 Optimization of 3D Gaussian Parameters

In the paper from kerbl [12] the gaussians are initialized from SfM points and further optimized via gradient optimization. The parameters of the 3D Gaussians are iteratively optimized by minimizing the photometric loss between the rendered and training images, which is given by

$$\mathcal{L}_{Photo} = \sum_i ||C_i^{\text{rendered}} - C_i^{\text{grount truth}}||^2. \tag{3.11}$$

During optimization, C is encoded with spherical harmonics $SH \in \mathbb{R}^{16}$ to account for direction-based color variations. Covariance is decomposed as $\Sigma = RSS^TR^T$, where $R \in SE(3)$ and $S = \text{diag}(s) \in \mathbb{R}^{3\times3}$ are rotation and scale, respectively. This factorization for S is chosen to preserve the positive semi-definite property of the covariance during gradient-based optimization. While there is no specific regularization terms in the paper from kerbl [12], there is the method of densification and pruning, to manage more and less detailed gaussians. When Gaussians become too unexpressive, e.g. either too big or too small for a certain detail, they get split and copied, densifying the scene. If an area has more gaussians than it needs, then individual individual ones contribute too little to the scene, they get pruned away. An overview can be seen in Fig. 4.1.

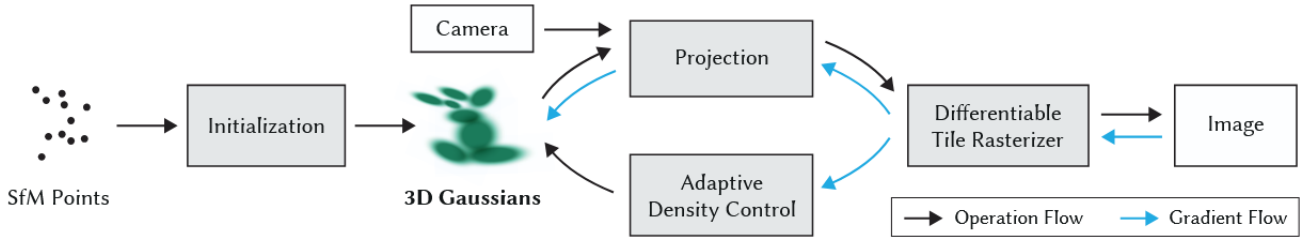

Figure 3.1: Overview of the gaussian splatting pipeline. From Kerbl et al. [12]

Figure 3.2: **Comparison with NeRF method Mip-NeRF360 and Gaussian splatting** (named Ours). While the qualitative differences are not too significant, the discrepancy in rendering and training speed are immense. Illustration taken from 3DGS paper [12].

3.1.7 Comparison with and Advantages over Other Novel-View Synthesis Methods

The advantages of choosing Gaussians Splatting as a scene representation has advantages for training time, inference time (renderings NVS) and practical aspects of the explicit nature. Other learned scene representations, like NeRFs take up more time in general. Having an explicit representation, no ray marching sampling and the approximation of splats, make it fast and efficient for training. Due to its rapid sampling speed, the rendering and training of Gaussian splatting much more suitable for Slam methods compared to other neural methods. Specifically, a NeRF needs to be trained for a significantly longer time for the same performance as a 3DGS model as seen in Fig. 3.2. Another advantage of having the explicit representation is the ability of fine grained editing of the model. While some gaussians still are ambiguously conected to multiple objects, compared to a TSDF or NeRF this provides an important flexibility for editing. Combining both the performane in speed and explicit representation is the key to our method GaME.

Chapter 4

Method

GaME builds and maintains a 3D map capable of novel view synthesis of an evolving environment, *i.e.* one where changes can occur during scanning but outside the view of the sensor at any time. An overview of our system is shown in Fig. 4.1. GaME processes depth and color images from an RGB-D sensor, using camera poses and panoptic segmentation from external estimators [2, 15]. For each keyframe, GaME triggers the Dynamic Scene Adaptation (DSA) module to incorporate new geometry and update stale regions. Simultaneously, the keyframe management system updates the state of the participating keyframes to ensure consistent multi-view optimization despite scene changes.

4.1 Online Mapping Framework

4.1.1 One global map instead of submaps

We model our map as a set of 3D Gaussians $\{\mathbf{G}_i\}_{i=1}^{N}$. The goal is to create a map with object-level *semantic consistency* [29]. Intuitively, this reflects the prior knowledge that objects tend to move as a whole, even under partial observations of changed objects. Modeling objects organized in submaps is possible, but has several problems. To do this successfully, one would

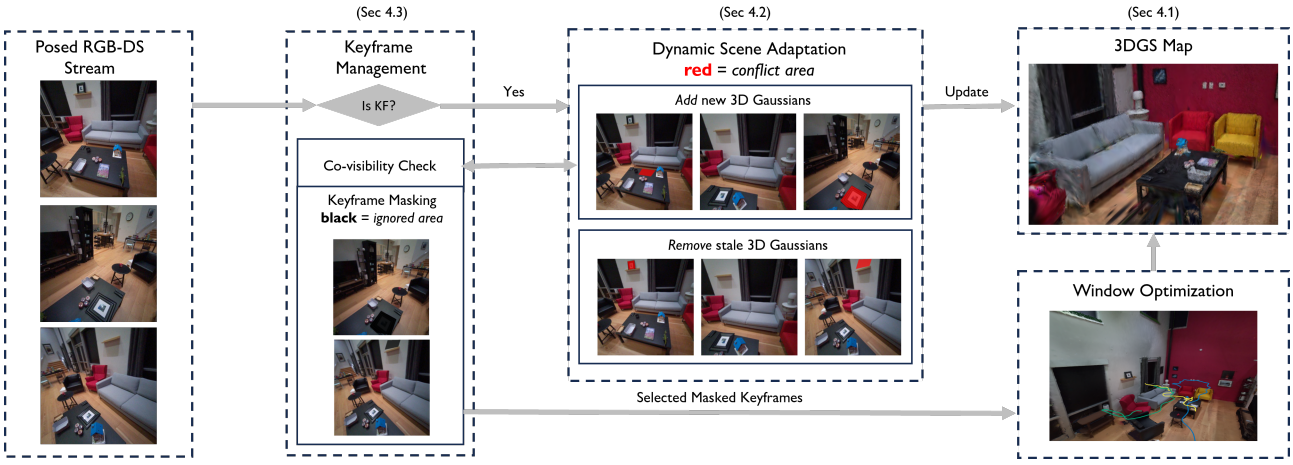

Figure 4.1: **GaME Architecture.** Given a segmented RGB-D input stream, the keyframe management system selects keyframes and triggers the dynamic scene adaptation (DSA) module. DSA first integrates newly observed geometry, then removes outdated geometry using covisible keyframes from the 3D Gaussian Splatting map. The keyframe manager then masks stale regions, and the mapping system uses the processed keyframes for local covisibility window optimization.

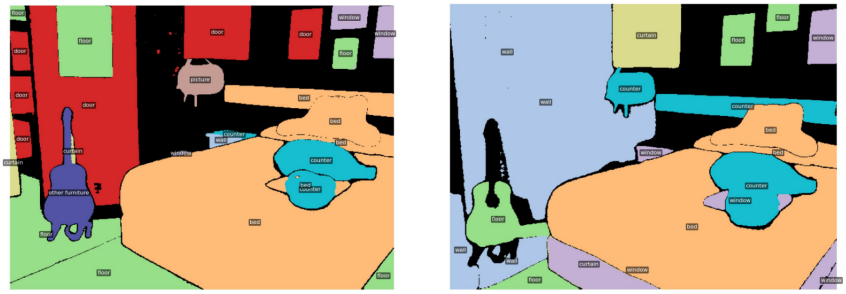

Figure 4.2: Example of sequentially inconsistent 2D panoptic segmentations. Sequential consistency is not guranteed and thus not reliable for state of the art models. Example taken from work of Li et al. [16].

need accurate panoptic segmentations, which are also consistent over frames. In practice, using state of the art models such as [15] is very challenging. Several factors make this paradigm of globally mapping every object as a separate submap impractial. First and most notable, the semantic or panoptic labels are ambiguous and are almost guranteed to change in practice. While it is not impossible to address this issue by tracking segmentation masks by a simple Intersection over Union algorithm, this would only be stable if not the labels but at least the masked areas of objects would be consistent over frames. This is not generally the case. There are many such cases, where an object is in frame, but not segmented at all. This happens when the object is on the side or further away from the camera, which almost always will happen in any environment when moving around. This is a common problem that is also discussed in other current research, where these segmentation models are utilized [16]. An example of such inconsistent segmentation can be seen in Fig. 4.2. Directly extracting object-level submaps is challenging, as noisy observations can lead to many unnecessary object allocation and de-allocation operations [29]. This problem is exacerbated for 3DGS, where optimizing individual object clouds is difficult since the splatting mechanism (3.7) inherently correlates all Gaussians. To work around these limitations, we propose to build a singular 3DGS representation and extract objects on an 'as needed' basis during *Dynamic Scene Adaptation* (Sec. 4.2).

4.1.2 Loss Functions

During online mapping, we optimize the 3DGS parameters of the scene using a set of covisible keyframes for supervision, minimizing the loss:

$$L = \lambda_{\text{color}} \cdot L_{\text{color}}(\hat{I}, I) + \lambda_{\text{depth}} \cdot L_{\text{depth}}(\hat{D}, D), \tag{4.1}$$

where I is the original image, $\hat{I}$ is the rendered image, D and $\hat{D}$ are the measured and reconstructed depth maps, and $\lambda_{\text{color}}, \lambda_{\text{depth}} \in \mathbb{R}_{\geq 0}$ are loss weights. The color loss and depth losses are defined as:

$$L_{\text{color}}(\hat{I}, I) = (1 - \lambda) \cdot \frac{1}{K} \sum_{p} \|\hat{I}(p) - I(p)\| + \lambda\big(1 - \text{SSIM}(\hat{I}, I)\big), \tag{4.2}$$

$$L_{\text{depth}}(\hat{D}, D) = \frac{1}{K} \sum_{p} |\hat{D}(p) - D(p)|, \tag{4.3}$$

where K is the number of rendered pixels in the rendered image or depth map, $p \in \mathbb{Z}^2$ denotes the pixel coordinates, and $SSIM$ is a structure similarity loss [32].

4.2 Dynamic Scene Adaptation

GaME addresses three potential scenarios: the addition, movement, or removal of an object. Note that movement can be decomposed into a removal, tracking, and re-addition step. While GaME can be readily combined with object re-localization techniques [31, 7, 27], tracking does not affect NVS and the problem can thus be simplified into two core operations: *Add* and *Remove*.

4.2.1 Add operation

In contrast to adding 3DGS in static scenes, in evolving scenes, it is essential to distinguish between adding new geometry for areas that were previously unseen and for objects that have newly appeared. We identify under-reconstructed regions by low Gaussian opacity and high re-rendering depth loss. Formally, the regions $\mathcal{R}_{\text{unobserved}}$ where new Gaussians are added are given by:

$$\mathcal{R}_{\text{unobserved}} = \{\mathbf{p} \in \mathbb{Z}^2 : L_{\text{depth}}(\mathbf{p}) > \theta_{\text{depth}} \wedge \alpha(\mathbf{p}) < \theta_{\text{opacity}}\}, \tag{4.4}$$

where $L_{\text{depth}}(\mathbf{p})$ denotes the re-rendering depth loss at pixel $p \in \mathbb{Z}^2$, $\alpha(\mathbf{p})$ represents rendered opacity, and $\theta_{\text{depth}} \in \mathbb{R}$, and $\theta_{\text{opacity}} \in \mathbb{R}$ are the respective thresholds. The method lifts the input RGB-D frame in $\mathcal{R}_{\text{unobserved}}$ to 3D and uses the resulting points to initialize the means for new Gaussians.

DSA further determines which parts of the input correspond to newly added objects, where regions of high opacity but with an input depth significantly smaller than the rendered depth indicate previously reconstructed areas occluded by newly added geometry. Formally, the set of Gaussians that might need adaptation is defined as:

$$\mathcal{G} = \{\mathbf{G}_i : \alpha(\mathbf{p}) \geq \theta_{\text{opacity}} \wedge \hat{D}(\mathbf{p}) > D(\mathbf{p}) + \epsilon_{\text{depth}}\}, \tag{4.5}$$

where $\epsilon_{\text{depth}} \in \mathbb{R}$ accounts for depth measurement noise and ensures robustness against Gaussians still converging to their final values. To enforce semantic consistency, GaME then examines each object mask of the current frame. If a sufficient portion of the mask overlaps with the rendering of $\mathcal{G}$, it marks the corresponding Gaussians from $\mathcal{G}$ as needing adaptation:

$$\mathcal{G}_{\text{add}} = \left\{ \mathbf{G}_i \in \mathcal{G} : \frac{|\text{mask}(\mathbf{p}) \cap \text{render}(\mathcal{G})|}{|\text{mask}(\mathbf{p})|} \geq \theta_{\text{mask}} \right\}, \tag{4.6}$$

where $\text{mask}(\mathbf{p})$ denotes the binary mask indicating the presence of the object at pixel $\mathbf{p}$, and $\theta_{\text{mask}} \in \mathbb{R}$ is the overlap threshold. This further increases robustness against noisy measurements to avoid allocating spurious Gaussians or 'floaters'.

4.2.2 Remove operation

Similarly, DSA identifies objects that have disappeared from the scene. Specifically, changed parts of the model can be identified as areas where the opacity is high (the model is well reconstructed), but the model disagrees (visually or geometrically) with the measurements:

$$\mathcal{G}_{\text{remove}} = \{\mathbf{G}_i : \alpha(\mathbf{p}) > \theta_{\text{opacity}} \wedge L_{\text{color}}(\hat{I}, I) > \theta_{\text{color}} \wedge \hat{D}(\mathbf{p}) < D(\mathbf{p}) - \epsilon_{\text{depth}}\}, \tag{4.7}$$

where $\theta_{\text{color}} \in \mathbb{R}$ is the threshold. There are two notable changes compared to the *Add* condition. First, the sign of the depth criterion is inverted, reflecting areas where the rays of the observation would penetrate into the current model, thus indicating that the model geometry can no longer be present. This also avoids spurious detections where objects are simply occluded rather than absent. Second, we leverage the high visual fidelity of 3DGS as another conflict signal

Figure 4.3: **Illustration of *Add* and *Remove* operations.** The input disagrees with the rendered model (red). For disappearance (top), the conflicting region is projected from previous keyframes for removal (red), where semantic consistency is enforced through the object mask (blue). This allows GaME to extract complete objects even under partial observations and occlusion. When a new object appears on the scene (bottom), new Gaussians are added (red) and the area of the new object is marked as stale in previous keyframes to prevent the contamination of the optimization process.

by adding a color term. In contrast to most long-term mapping methods, which rely solely on 3D information [5, 27, 13, 29, 28, 23], this allows GaME to also detect changes that are not geometrically significant, such as a sheet of paper being removed from a table. To ensure semantic consistency of removed objects, it is insufficient to rely on a single frame, as it may capture only partial views, and the object cannot be semantically detected since it is already absent. To address this, GaME renders $\mathcal{G}_{\text{remove}}$ to each covisible keyframe KF and verifies whether the color and depth are consistent with the model:

$$\mathcal{R}^{\text{KF}}_{\text{remove}} = \{\mathbf{p} \in \mathbb{Z}^2 : L_{\text{color}}(\mathbf{p}) > \theta_{\text{color}} \wedge L_{\text{depth}}(\mathbf{p}) > \theta_{\text{depth}} \wedge \alpha(\mathbf{p}) > \theta_{\text{opacity}}\}. \tag{4.8}$$

Next, each mask in the keyframe is evaluated to determine whether a significant portion intersects with the conflicting region, marking the masked Gaussians for removal:

$$\mathcal{G}^{\text{KF}}_{\text{remove}} = \left\{ \mathbf{G}_i : \frac{|\text{mask}(\mathbf{p}) \cap \mathcal{R}^{\text{KF}}_{\text{remove}}|}{|\text{mask}(\mathbf{p})|} \geq \theta_{\text{mask}} \right\} \tag{4.9}$$

Finally, the joint set, $\mathcal{G}_{\text{remove}} \cup \bigcup_{\text{KF}} \mathcal{G}^{\text{KF}}_{\text{remove}}$, is removed from the global Gaussian model. This naturally allows the extraction of individual removed objects from the global model, or deletion of the Gaussians if only an up-to-date reconstruction is desired. The DSA process is illustrated in Fig. 4.3.

4.3 Keyframe Management

Managing the keyframes is essential for a system robust to changes, clearly optimizing with pre-change keyframes leads to ill-posed optimization. Using all the frames from the video stream

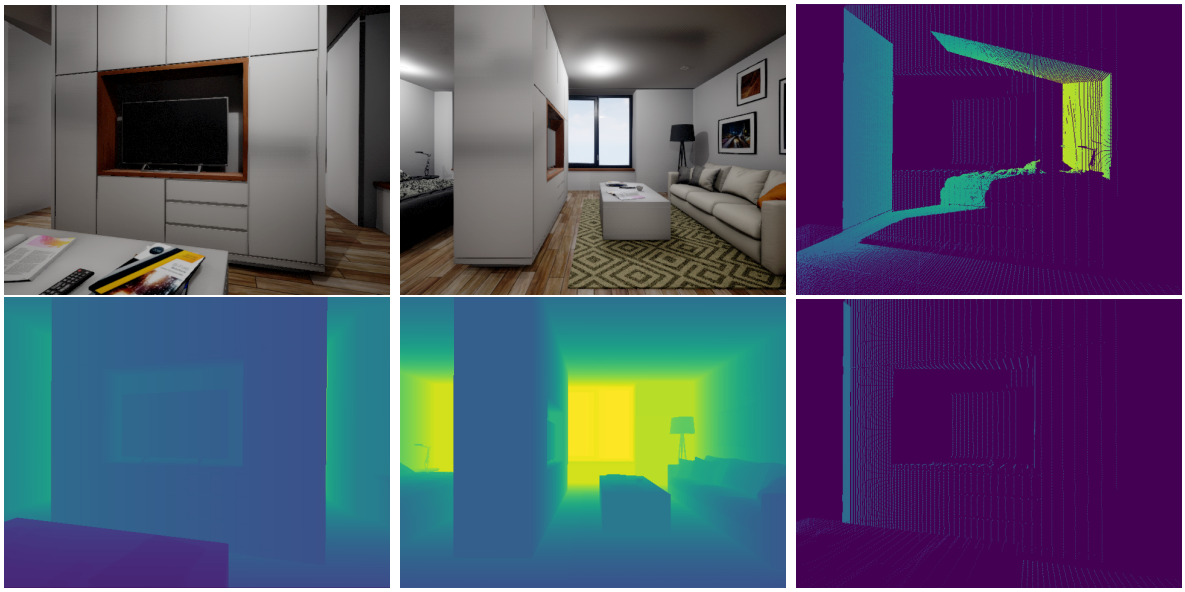

Figure 4.4: **Illustration of point reprojection for covisibility** Ground-truth color and depth of current frame (right). Ground-truth color and depth of covisible frame (middle). Reprojected depth points and occluded reprojection (right). Points are first reprojected from covisible to current frame and then occluded via the given ground-truth depth.

for optimization is computationally infeasible. The selection is based on whether keyframes are covisible and contain changes in the scene. A small window W_k of keyframes based on inter-frame covisibility is maintained by reprojection. Effective keyframe management aims to select non-redundant keyframes that observe the same region while spanning a wide baseline to enforce stronger multiview constraints. In GaME, keyframes are selected every time a frame exceeds a translation $\theta_{\text{translation}} \in \mathbb{R}$ or a rotation $\theta_{\text{rotation}} \in \mathbb{R}$ threshold. This can be replaced with another keyframe suggestion from an external tracking system. After the training, the remaining keyframes are used for a refinement. This is common practice and can stabilize the smaller influence more recent frames would have. While it helps it will be shown it helps with performance, it is not a must have for GaME to work.

4.3.1 Covisibility

Upon triggering *Add* or *Remove* operations by DSA, all covisible keyframes are retrieved by reprojecting 3D points from the current frame and selecting frames where these points are not occluded. The reprojection for frame F_i to check if frame F_j is covisible is done such that points from F_j are reprojected into the view of F_i and then occluded by the depth mask of F_i, to account for occlusion. This is easily done with the standard pinhole camera pipeline, given the poses and depth maps of both frames. An example of this can be seen in Fig. 4.4 This approach differs from conventional 3DGS methods [19, 35], which determine covisibility based on the number of Gaussians used to re-render the keyframes. We found projection to be more reliable than re-rendering in multi-room environments, as Gaussians from different rooms may still appear visible due to the alpha blending, leading to erroneous covisibility estimates. In the case of the example in Fig. 4.4, this would also mark most of frames visible which exclusively cover gaussians behind the wall as covisible. Especially in multi-room environments this can lead to unnecessary calculations and longer runtimes during the DSA.

Figure 4.5: **Paritally ignored frames**. Areas with detected errors are marked with black pixels. Frame with ignored area, where chair was added (left). Frame with only minor errors caused by high depth gradients of angles surface (right). While the chair is added, the floor and background wall is still a valid optimization objective, likewise, as rendered gaussians often cause depth errors on steep surfaces, the overall frame should still be used for optimization despite its false positive error detections.

4.3.2 Keyframe Masking

Importantly, in evolving scenes, excluding outdated visual information is crucial, as the scene may change over time, making previous observations detrimental for the 3DGS optimization process. However, discarding entire keyframes that observe stale geometry can result in losing useful information. While some parts of the scene may change, other regions often remain stable and can provide valuable signals for 3DGS optimization. Specifically, frames could be only partially outdated, as one object being removed, does not change the rest of the scene. For example, a removed chair should not render a picture that contains a full view of a room unusable. Additionally due to the nature of gaussian splatting, it can happen that only minor insignificant errors in depth and color occur, which are thus masked in the frame but also should not let the frame get discarded prematurely. Therefore, rather than directly discarding entire keyframes as stale when only minor parts are outdated, GaME selectively ignores only the stale areas within them, as seen in Fig. 4.5.

To achieve this, before removing $\mathcal{G}_{\text{remove}}$ or $\mathcal{G}^{\text{KF}}_{\text{remove}}$, they are rendered onto the keyframe, and the image regions rendering them are masked out. The same procedure is applied for $\mathcal{G}_{\text{add}}$. Formally, the masked region $\mathcal{M}^{\text{KF}}$ within a keyframe is defined as:

$$\mathcal{M}^{\text{KF}} = \left\{\mathbf{p} \in \mathbb{Z}^2 : \mathbf{p} \in \text{render}(\mathcal{G}_{\text{remove}}) \cup \text{render}(\mathcal{G}^{\text{KF}}_{\text{remove}}) \cup \text{render}(\mathcal{G}_{\text{add}})\right\} \tag{4.10}$$

where $\mathbf{p}$ represents a pixel location within the keyframe, and $\text{render}(\mathcal{G}_*)$ denotes the set of pixels influenced by the Gaussian set $\mathcal{G}_*$ when rendered. During optimization, for all the keyframes that have ignored areas, GaME optimizes the losses:

$$L_{\text{color}}(\hat{I}, I) = \frac{1}{\sum_p \mathbb{I}[p \notin \mathcal{M}^{\text{KF}}]} \sum_p \mathbb{I}[p \notin \mathcal{M}^{\text{KF}}] \cdot \|\hat{I}(p) - I(p)\|, \tag{4.11}$$

$$L_{\text{depth}}(\hat{D}, D) = \frac{1}{\sum_p \mathbb{I}[p \notin \mathcal{M}^{\text{KF}}]} \sum_p \mathbb{I}[p \notin \mathcal{M}^{\text{KF}}] \cdot |\hat{D}(p) - D(p)|, \tag{4.12}$$

Note that compared to (4.2), the structure similarity term is dropped as its convolutional nature does not interact well with masking.

4.3.3 Refinement

An added benefit of GaME is that, the masked keyframes can be directly used for further refinement after the mapping operation, leading to consistent optimization despite changes occurring throughout the mapping stage. This refinement is essentially implemented the same just as a normal optimization loop during training just longer. While the original 3DGS does use densification to add new Gaussians, we currently do not use it. Points are only added in the add operation as mentioned before. For arbitrary long sequences with many changes this extra refinement step is a good choice to stabilize the final reconstruction. In our approach, having the most recent state of the scene accurately reconstructed is essential. Thus, optimizing over all of the frames after the main training, equalizes the scewed influence early frames have compared to the latest and desirably most influencial frames.

Chapter 5

Experiments

Figure 5.1: **Qualitative Results.** Comparison across different long-term scene changes. (A) A black office chair appears in the scene; (B) the toy house and chair are moved, the picture is moved from the table to the shelf; (C) the cutlery on the table is replaced, the painting and the right chair are moved. GaME is the only method that captures the scene evolution and preserves high rendering quality.

5.1 Experiments

5.1.1 Datasets

For datasets we selected both synthetic and real datasets containing change. In addition to that another dataset is selected to assess performance on static scenes. Examples are shown in Fig. 5.2

We test our method on the Flat [29] dataset, which consists of two RGB-D sequences captured in a synthetic environment with significant changes occurring between. Another challenging apsect of this dataset is the room structure. While many datasets only consist of one room, the Flat dataset represents a two-room scene, making it more challenging. Especially for our

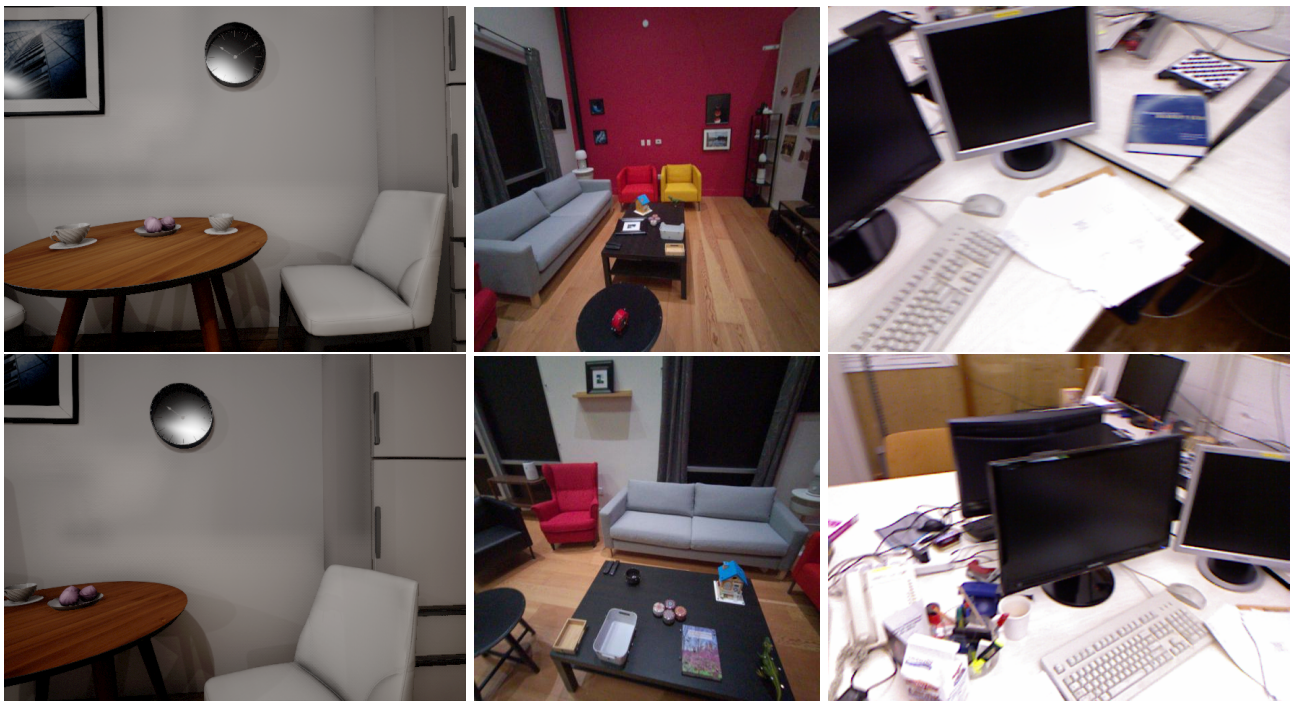

Figure 5.2: **Examples of datasets used for experiments**. Flat dataset (left), First room from Aria-multiagent (middle) and *desk* from TUM-rgbd (right). For Flat, changes such as the removed cup and moved chair are visible. In aria the removed picture, removed book and moved house on the table and moved coffe table on the side are seen. TUM does not contain any changes.

situation, where covisible frames have to be selected, the multi-room setting poses the challenge of defining covisibility not on the influence of the gaussians for rendering the current view but on occlusion based methods as described in Sec. 4.2.

We further evaluate on the Aria [22] dataset to assess performance on real-world data. For the two selected rooms, there are 5 agents and a testing trajectory each, moving through the room. We select two recordings from two rooms each that have undergone long-term changes. To evaluate the change detection, two agents with different states of the room and a testing trajectory have to be selected, such that the testing trajectory is in the final state of the room. For The first room, agent 0 and agent 4 are selected for training and agent 4 for testing. For the second room, agent 4 and agent 0 are selected for training and the test run for testing.

Finally, the TUM-RGBD [30] dataset is used. While this work aims to address long-term changes in scenes its important to validate if the method can perform on different static scenes. We selected scenes *desk*, *xyz* and *office*, following the protocol of [19]. They depict everyday scenarios such as a cluttered office.

5.1.2 Evaluation Metrics

To assess rendering quality, we compute PSNR, SSIM [32] and LPIPS [40]. Rendering metrics on all the datasets is evaluated by rendering full-resolution images along the ground-truth trajectory. We assess the depth error using the L1 norm in centimeters.

Peak Signal-to-Noise Ratio (PSNR) originates from signal processing in information theory, where it comparing the quality of the recieved signal against the original. It is based on the mean squared error (MSE) between the recieved and original signal and measured in decibels (dB). In our case the signals are the images, rendered from the Gaussian model. The

PSNR is defined as

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX^2}{\text{MSE}} \right), \tag{5.1}$$

where MAX denotes the maximum possible value for a pixel (255 for 8-bit format), and MSE for images I,K is defined as

$$\text{MSE}(I, K) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \big(I(i,j) - K(i,j)\big)^2. \tag{5.2}$$

A higher PSNR indicates better reconstuction quality.

Structural Similarity Index Measure (SSIM) is a metric designed to mimic the human perception of image quality. Opposed to traditional metrix like PSNR or MSE, SSIM models similarity based on structural information, lighting and contrast. For this reason, it is a good complementary addition next to PSNR. Given two image patches x and y, SSIM is defined as

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{5.3}$$

where μ_x and μ_y are the mean intensities of x and y, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance between them. The constant terms C_1 and C_2 are used to stabilize the division and are defined as:

$$C_1 = (K_1 L)^2, \quad C_2 = (K_2 L)^2, \tag{5.4}$$

where L is the dynamic range of pixel values (255 for 8-bit format), and $K_1 = 0.01$, $K_2 = 0.03$ are small constants. The values of SSIM range from -1 to 1, where 1 would suggest perfect structural similarity. To obtain a single score for the whole image, the score is calculated for all patches and averaged.

Learned Perceptual Image Patch Similarity (LPIPS) is a similarity measure designed to also better align with human perception of image quality, specifically for natural images. Unlike PSNR and SSIM, which are defined by hand-crafted formulas based on low-level statistics, LPIPS utilizes deep neural networks trained to grasp perceptual similarity. LPIPS works by first extracting image features using a pre-trained convolutional neural network and then computing distances between those features accross multiple layers. The distance is a weighted L2 norm over the corresponding image features. Formally, given two image patches x and y, LPIPS distance is defined as

$$\text{LPIPS}(x, y) = \sum_l w_l \cdot \frac{1}{H_l W_l} \sum_{h,w} \left\| \hat{f}_l^x(h,w) - \hat{f}_l^y(h,w) \right\|_2^2, \tag{5.5}$$

where $\hat{f}_l^x$ and $\hat{f}_l^y$ are the feature activations at layer l for x and y, and w_l are learned weights to scale the influence of each layer. H_l and W_l are the height and width of the feature maps in layer l. Intuitively, since it is calculated by distance, lower LPIPS values represent higher perception similarity. LPIPS has been shown to correlate better with visual similarity than conventional metrics, thus making it specifically useful for tasks involving generative model or in our case image synthesis.

5.1.3 Baselines

We compare GaME with state-of-the-art 3DGS online reconstruction systems MonoGS [19] and SplaTAM [11]. While those two also do tracking, only the mapping part will be evaluated using the gt poses provided by the dataset. We compare with DG-SLAM [17], a recent dynamic 3DGS method, to assess the ability of dynamic 3DGS SLAM to handle evolving scenes.

Methods	PSNR [dB] ↑	SSIM ↑	LPIPS ↓	Depth L1 [cm] ↓
SplaTAM [11]	16.15 / 13.18	0.471 / 0.301	0.648 / 0.729	238.4 / 308.2
MonoGS [19]	21.24 / 21.33	0.771 / 0.769	0.398 / 0.396	30.95 / 29.91
DG-SLAM [17]	13.72 / 13.70	0.586 / 0.603	0.737 / 0.739	73.76 / 73.90
GaME (Ours)	**23.47 / 24.79**	**0.880 / 0.899**	**0.266 / 0.227**	**15.59 / 16.63**

Table 5.1: **Rendering performance on the Flat dataset.** GaME shows superior performance in both color and depth rendering. Cells show metrics for input / novel views.

5.1.4 Evolving Scene Evaluation Protocol

The goal of mapping evolving scenes is to render only the most up-to-date reconstruction without prior information on when the scene was changed. In addition, the system should be able to accurately render both the views it observed (input views) and unobserved frames (novel views). To achieve this capability, we merge RGB-D captures from every scene in the Aria and Flat datasets into a single continuous sequence to recreate this real-world behavior. For rendering evaluation, every 10th frame from each scene's last RGB-D sequence is held out for novel view synthesis testing. The remaining 90% of the frames are used to evaluate input view synthesis. To isolate the mapping performance, ground-truth poses are used for GaME and all baselines.

5.2 Evolving Scene Mapping Results

We run GaME on the Flat and Aria datasets to evaluate rendering quality in evolving scenes, shown in Tabs. 5.1 and 5.2. The Flat dataset is designed to test mapping under scene changes and includes more substantial long-term dynamics. We note that all losses are computed over complete images, where changed areas typically occupy a smaller part. Nonetheless, we observe notable differences in rendering performance on both datasets, where GaME is the only method that accurately adapts the reconstruction to even subtle scene changes without compromising rendering quality. This granularity is shown in qualitative results in Fig. 5.1. Beyond high rendering quality, GaME is able to accurately resolve changes also for challenging cases such as small cutlery on the table and flat paintings.

5.3 Ablation Studies

Although GaME performs better using DSA, it is important to thoroughly validate the approach. We conduct ablation studies to assess key design choices and analyze the reconstruction performance of both evolving and static scenes. Additional experiments on runtime and robustness to noisy input poses are also done.

5.3.1 Add and Remove operations ablation for DSA

We ablate the components of DSA in Tab. 5.3, showing that its presence significantly improves performance. While differences between individual DSA variants are less pronounced in final rendering quality, due to keyframe optimization correcting finer detail, combining both operations consistently yields the best performance and fastest adaptation. While the adding operation is more simple, it is less trivial that the removal operation works as intended. To be more confident, it can be observed how the removal operation works in practice. In **??** the removal of a bed in another room is illustrated.

Methods	Scene	PSNR [dB] ↑	SSIM ↑	LPIPS ↓	Depth L1 [cm] ↓
SplaTAM [11]	room0	1.80 / 16.75	0.780 / 0.447	0.239 / 0.444	4.12 / 17.79
	room1	22.94 / 17.90	0.810 / 0.543	0.219 / 0.440	2.84 / 15.9
	Avg	22.37 / 17.33	0.795 / 0.495	0.229 / 0.442	3.48 / 16.8
MonoGS [19]	room0	25.28 / 25.19	0.781 / 0.779	0.277 / 0.273	5.15 / 5.09
	room1	23.12 / 23.02	0.844 / 0.842	0.236 / 0.241	4.99 / 5.01
	Avg	24.20 / 24.11	0.813 / 0.811	0.257 / 0.257	5.07 / 5.05
DG-SLAM [17]	room0	15.78 / 15.63	0.578 / 0.580	0.761 / 0.763	67.60 / 69.05
	room1	12.62 / 12.44	0.651 / 0.643	0.702 / 0.708	55.47 / 57.03
	Avg	14.20 / 14.04	0.615 / 0.612	0.732 / 0.736	61.54 / 63.04
GaME (Ours)	room0	**28.87 / 28.94**	**0.934 / 0.935**	**0.166 / 0.166**	**2.3 / 2.2**
	room1	**30.83 / 30.54**	**0.961 / 0.958**	**0.107 / 0.114**	**1.8 / 1.9**
	Avg.	**29.85 / 29.74**	**0.948 / 0.947**	**0.137 / 0.140**	**2.05 / 2.05**

Table 5.2: **Rendering performance on the Aria dataset.** GaME shows superior performance in both color and depth rendering. Cells show metrics for input / novel views.

Figure 5.3: **Demonstration of removal process** First a frame (top right) is observed and compared to the current view of the model (top middle). After the adding operation, the removal operation is triggered and finds several collections of gaussians getting overlapped in covisible frames (bottom half). Notice, how unseen gaussians of the bed get marked that are occluded by the corner of the wall. This is the partial obervability property we wanted to achieve. Finally, those gaussians are removed and the model is ready for the next frame in the sequence (top right)

Method	PSNR [dB] ↑	SSIM ↑	LPIPS ↓	Depth L1 [cm] ↓
No DSA	21.28 / 20.75	0.846 / 0.838	0.265 / 0.289	44.6 / 42.9
Add	24.55 / 23.37	**0.900 / 0.884**	0.229 / 0.268	17.9 / 17.3
Remove	24.31 / 23.14	0.896 / 0.880	0.232 / 0.267	17.6 / 16.3
Add and Remove (Ours)	**24.76 / 23.47**	0.898 / 0.881	**0.228 / 0.267**	**16.9 / 15.8**

Table 5.3: **Dynamic Scene Adaptation (DSA) ablation in the Flat dataset.** Combination of *Add* and *Remove* operations give the best performance. Cells show metrics for input / novel views.

Method	PSNR [dB] ↑	SSIM ↑	LPIPS ↓	Depth L1 [cm] ↓
No KF Filtering	22.29 / 21.51	0.853 / 0.853	0.254 / 0.280	30.5 / 29.5
Full KF Filtering	17.63 / 16.58	0.675 / 0.614	0.500 / 0.525	215.0 / 244.1
Partial KF Filtering (Ours)	**24.76 / 23.47**	**0.898 / 0.881**	**0.229 / 0.267**	**16.9 / 15.8**

Table 5.4: **Keyframe management ablation on Flat dataset.** Retaining keyframes is essential to constrain the background. However, to avoid artifacts, conflicting regions must be accurately filtered.

5.3.2 Importance of Keyframe Management

In Tab. 5.4, we compare keeping all keyframes against ignoring stale keyframes and ignoring only stale regions. The naive approach to dealing with scene changes in 3DGS is to completely remove keyframes from optimization when any change has been detected. Using the naive approach for evolving scenes of fully discarding conflicting keyframes can leave parts of the scene severely under-constrained, leading to the worst outcome. It is also important to check whether keyframe filtering achieves any improvement at all. We tested not filtering out any keyframes, regardless of changes being detected. Not filtering out any keyframes achieves clear background rendering. It is important to note that the background accounts for the majority of pixels. These pixels dominate the losses. Thus, the overall mapping does not completely fail. However, it inevitably results in artifacts and inconsistencies for changed regions, as seen in Fig. 5.4.Our proposed approach is to solely remove keyframes which are sufficiently flawed. Specifically, this partial keyframe filtering achieves sufficient constraints for the Gaussian optimization while removing conflicting regions. Since partial keyframe filtering resulted in the highest performance in this ablation, this highlights the importance of retaining information about the background *and* resolving conflicts in keyframes for 3DGS optimization.

While the quantitative results hint at the differences in performance, visually it is very clear what effect the keyframe management has in Fig. 5.4. Although DSA is used for all 3 ablations, having no keyframe filtering will ultimately lead to optimization conflicts, most notable for half chairs or outlines of the picture moved on the wall. While changes are still correctly managed, fully discarding frames as soon as they contain conflicts severely deteriorates the performance. Finally, using both keyframe management with partial keyframe filtering and DSA will yield the best result.

5.3.3 Static scene reconstruction

While our method is specifically designed to reconstruct evolving scenes, it should not lose the ability to reconstruct static scenes. To verify this, we trained on differing scenes of the TUM-RGBD dataset, which does not contain any changes. Results on the TUM-RGBD dataset

Figure 5.4: **Qualitative comparison for keyframe management ablation.** Rendered frame for no keyframe Filtering (left), full keyframe Filtering (middle) and partial keyframe filtering (right).

Method	desk	`xyz`	`office`	**Average**
SplaTAM [11]	20.92	**21.03**	21.61	**21.19**
MonoGS [19]	17.41	15.09	19.93	17.48
GaME (Ours)	**20.94**	20.54	**21.96**	21.15

Table 5.5: **Rendering performance on TUM-RGBD dataset.** GaME performs on par with state-of-the art mapping systems for static scenes. PSNR [dB]↓ shown for input views.

shown in Tab. 5.5 show that GaME performs on par with state-of-the-art systems. This suggests that the introduced DSA and keyframe management are robust to noise and false positives in model updates and masking, which could deteriorate performance.

5.3.4 GaME is an online mapping system

Our approach is of an incremental nature with partial optimization performed on every new keyframe. Optionally, our approach lends itself to further final refinement. To test how relevant the refinement is for overall performance, we do an ablation with and without refinement after the main training. Both options are shown in Tab. 5.6. While refinement can potentially improve the final rendering quality, we note that the model is already well converged after incremental processing.

Method	**PSNR [dB] ↑**	**SSIM ↑**	**LPIPS ↓**	**Depth L1 [cm] ↓**
No Refinement	24.48 / 23.30	**0.899** / **0.883**	0.231 / **0.266**	17.00 / 15.90
With Refinement	**24.76** / **23.47**	0.898 / 0.881	**0.229** / 0.267	**16.90** / **15.80**

Table 5.6: **Final refinement ablation on Flat dataset.** While refinement can potentially improve the final rendering quality, GaME already converges well during incremental processing.

5.3.5 Noisy poses

We evaluate the camera poses on the Aria dataset with an external SLAM [19] system and use the estimated poses and keyframes instead of ground truth in Tab. 5.7. GaME is robust even when the pose estimation is not precise, exhibiting only a marginal performance drop under noisy pose conditions.

Camera Poses	PSNR [dB] ↑	SSIM ↑	LPIPS ↓	Depth L1 [cm] ↓
Ground-truth	26.63 / 26.71	0.920 / 0.920	0.179 / 0.180	2.4 / 2.4
Estimated	24.83 / 24.93	0.873 / 0.876	0.234 / 0.232	3.6 / 3.6

Table 5.7: **Usage of noisy poses on Aria room0.** GaME is robust to camera pose noise, exhibiting only a slight performance drop when using noisy poses instead of ground-truth. Camera poses and keyframes are estimated with an off-the-shelf reconstruction system [19].

5.3.6 Run-time Analysis

We compare the mapping runtime of our method with other state-of-the-art Gaussian-based systems in Tab. 5.8. Scenes with changing geometry naturally slow the mapping process, as existing methods struggle with seeding new geometry and optimizing under conflicting observations. In contrast, GaME is designed to explicitly handle such dynamic scene adaptation while being comparably efficient to the state-of-the-art mapping systems.

Metric	SplaTAM [11]	MonoGS [19]	DG-SLAM [35]	GaME (Ours)
FPS↑	0.129	**4.212**	0.346	0.165

Table 5.8: **Runtime Analysis on Flat dataset**. Evolving scenes significantly slow down the mapping process. While GaME supports scene adaptation mechanisms, it performs comparably with state-of-the-art mapping systems. The frame per second is calculated by dividing the time spent on mapping by the total number of processed frames. All metrics are profiled using an NVIDIA RTX 3090 GPU.

Chapter 6

Limitations

While GaME currently demonstrates robust rendering performance on evolving long-term dynamic scenes, several notable limitations exist. First, it does not yet handle *short-term* dynamic objects, which would result in inefficient addition and removal. Extending the dynamic scene adaptation mechanism to consider both dynamics is an exciting future direction. Second, we primarily study *mapping with external pose tracking*. Although sparse odometry systems can reject many changes as outliers [2], change-aware pose refinement and integration of a dedicated tracking mechanism is an exciting future direction that could improve accuracy. Third, while GaME is an online algorithm that can process sequences captured in evolving scenes, the current implementation has *not been optimized for performance*, which limits real-time deployment. Consistently updating the 3DGS map through incremental segmentation, rather than iterating over segmentation masks, could significantly speed up the approach. Finally, *DSA is currently performed only per incoming keyframe*. This has the advantage of not storing semantic information in the 3DGS model and comes with some inherent robustness through the multi-view optimization process. However, individual semantic masks can be noisy or missing, where a delayed decision or global optimization could improve performance by considering more semantic observations.

Chapter 7

Conclusions

7.1 Conclusions

We presented GaME, the first online mapping system for evolving scenes with novel view synthesis capabilities. GaME utilizes novel dynamic scene adaptation operations to detect and correct conflicts in the incrementally built 3DGS model. Our keyframe management method furthermore appropriately ignores stale areas in keyframes, retaining useful information while correcting for changed regions, resulting in a well-conditioned 3DGS optimization process. We thoroughly evaluate our methods on synthetic and real-world datasets, demonstrating 90+% in depth and 20+% color rendering performance and artifact-free novel view synthesis in long-term dynamic evolving scenes.

Chapter 8

Future Work

While this work addresses-long-term changes of the scene, it is unclear what impact short-term dynamics might have in the scene. Although it was not tested during experiments, since it was not the goal of this work, since the evaluation was done with datasets that do not contain short term dynamics. Datasets that could be used to evaluate this short-term change capability, could be both the TUM RGB-D [30] dataset, as it also contains scenes with dynamic components as well as BONN RGB-D [21] or any other RGB-D dataset, as long it contains moving parts such as human interaction with the environment on camera. It would be the logical next step to also address ignoring short-term dynamics and add it to the DSA module to make it a powerful solution for Gaussian Mapping that can handle short- and long-term changes. This can be addressed by building on works like DG-SLAM and Wildgs-slam [35, 41], which, as outlined in the Introduction chapter, tackle specifically this problem. As GaME is already using both depth data and panoptic-segmentation input from an external model, it would not impose a drastic pipeline refactor, rather an addition.

Since GT poses are rarely available in real-world applications, an important addition to this framework would be a built in tracking system, as it has been shown that the change detection would work with estimated poses. This would greatly improve usability of this method.

Chapter 9

Appendix

Note on Baselines Wild-GS [41] cannot be run with ground-truth poses, as it relies on an off-the-shelf depth estimator that may be inconsistent with them. At the same time, it fails to track the camera poses on the Aria and Flat datasets. For this reason, we were not able to compare our method with it.

Implementation Details. In Tab. 9.1 we provide the hyperparameters used in our experiments.

Dataset	λ_{color}	λ_{depth}	θ_{depth}	θ_{opacity}	θ_{color}	θ_{mask}	ϵ_{depth}	θ_{rotation}	$\theta_{\text{translation}}$	θ_{ignore}
Flat [29]	1.0	1.0	4.0	0.3	0.1	0.4	2.0	50	5.0	0.9
Aria [22]	1.0	1.0	4.0	0.3	0.3	0.4	4.0	25	0.5	0.2

Table 9.1: Hyperparameters for Flat [29] and Aria [22] datasets.

Bibliography

[1] Rareş Ambruş, Nils Bore, John Folkesson, and Patric Jensfelt. Meta-rooms: Building and maintaining long term spatial models in a dynamic world. pages 1854–1861. IEEE, 2014.

[2] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE transactions on robotics*, 37(6):1874–1890, 2021.

[3] Devikalyan Das, Christopher Wewer, Raza Yunus, Eddy Ilg, and Jan Eric Lenssen. Neural parametric gaussians for monocular non-rigid object reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10715–10725, 2024.

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

[5] Marius Fehr, Fadri Furrer, Ivan Dryanovski, Jürgen Sturm, Igor Gilitschenski, Roland Siegwart, and Cesar Cadena. Tsdf-based change detection for consistent long-term dense reconstruction and dynamic object discovery. pages 5237–5244. IEEE, 2017.

[6] Jiahui Fu, Yilun Du, Kurran Singh, Joshua B Tenenbaum, and John J Leonard. Robust change detection based on neural descriptor fields. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2817–2824. IEEE, 2022.

[7] Jiahui Fu, Yilun Du, Kurran Singh, Joshua B Tenenbaum, and John J Leonard. Neuse: Neural se (3)-equivariant embedding for consistent spatial understanding with objects. 2023.

[8] Jiahui Fu, Chengyuan Lin, Yuichi Taguchi, Andrea Cohen, Yifu Zhang, Stephen Mylabathula, and John J Leonard. Planesdf-based change detection for long-term dense mapping. *IEEE Robotics and Automation Letters*, 7(4):9667–9674, 2022.

[9] Huajian Huang, Longwei Li, Cheng Hui, and Sai-Kit Yeung. Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[10] Kai Katsumata, Duc Minh Vo, and Hideki Nakayama. A compact dynamic 3d gaussian representation fornbsp;real-time dynamic view synthesis. page 394–412, Berlin, Heidelberg, 2024. Springer-Verlag.

[11] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat, track & map 3d gaussians for dense rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

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

[13] Giseop Kim and Ayoung Kim. Lt-mapper: A modular framework for lidar-based lifelong mapping. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7995–8002. IEEE, 2022.

[14] Edith Langer, Timothy Patten, and Markus Vincze. Robust and efficient object change detection by combining global semantic information and local geometric verification. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8453–8460. IEEE, 2020.

[15] Feng Li, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation, 2022.

[16] Yue Li, Qi Ma, Runyi Yang, Huapeng Li, Mengjiao Ma, Bin Ren, Nikola Popovic, Nicu Sebe, Ender Konukoglu, Theo Gevers, Luc Van Gool, Martin R. Oswald, and Danda Pani Paudel. Scenesplat: Gaussian splatting-based scene understanding with vision-language pretraining, 2025.

[17] Ziqi Lu, Jianbo Ye, and John Leonard. 3dgs-cd: 3d gaussian splatting-based change detection for physical object rearrangement. *IEEE Robotics and Automation Letters*, 2025.

[18] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *IEEE International Conference on 3D Vision*, 2024.

[19] Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. Gaussian Splatting SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[20] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, December 2021.

[21] E. Palazzolo, J. Behley, P. Lottes, P. Giguère, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. 2019.

[22] Xiaqing Pan, Nicholas Charron, Yongqian Yang, Scott Peters, Thomas Whelan, Chen Kong, Omkar Parkhi, Richard Newcombe, and Carl Yuheng Ren. Aria digital twin: A new benchmark dataset for egocentric 3d machine perception, 2023.

[23] Jingxing Qian, Veronica Chatrath, James Servos, Aaron Mavrinac, Wolfram Burgard, Steven L Waslander, and Angela P Schoellig. Pov-slam: Probabilistic object-aware variational slam in semi-static environments. 2023.

[24] Jingxing Qian, Veronica Chatrath, Jun Yang, James Servos, Angela P Schoellig, and Steven L Waslander. Pocd: Probabilistic object-level change detection and volumetric mapping in semi-static scenes. 2022.

[25] Adam Rashid, Chung Min Kim, Justin Kerr, Letian Fu, Kush Hari, Ayah Ahmad, Kaiyuan Chen, Huang Huang, Marcus Gualtieri, Michael Wang, et al. Lifelong lerf: Local 3d semantic inventory monitoring using fogros2. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7740–7747. IEEE, 2024.

[26] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *The International Journal of Robotics Research*, 40(12-14):1510–1546, 2021.

[27] Joseph Rowell, Lintong Zhang, and Maurice Fallon. Lista: Geometric object-based change detection in cluttered environments. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3632–3638. IEEE, 2024.

[28] Lukas Schmid, Marcus Abate, Yun Chang, and Luca Carlone. Khronos: A unified approach for spatio-temporal metric-semantic slam in dynamic environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2024.

[29] Lukas Schmid, Jeffrey Delmerico, Johannes L. Schönberger, Juan Nieto, Marc Pollefeys, Roland Siegwart, and Cesar Cadena. Panoptic Multi-TSDFs: a Flexible Representation for Online Multi-resolution Volumetric Mapping and Long-term Dynamic Scene Consistency. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8018–8024, May 2022. ICRA.

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

[31] Johanna Wald, Armen Avetisyan, Nassir Navab, Federico Tombari, and Matthias Nießner. Rio: 3d object instance re-localization in changing indoor environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7658–7667, 2019.

[32] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[33] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, June 2024.

[34] Yunyang Xiong, Bala Varadarajan, Lemeng Wu, Xiaoyu Xiang, Fanyi Xiao, Chenchen Zhu, Xiaoliang Dai, Dilin Wang, Fei Sun, Forrest Iandola, et al. Efficientsam: Leveraged masked image pretraining for efficient segment anything. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16111–16121, 2024.

[35] Yueming Xu, Haochen Jiang, Zhongyang Xiao, Jianfeng Feng, and Li Zhang. DG-SLAM: Robust dynamic gaussian splatting SLAM with hybrid pose optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[36] Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. In *CVPR*, 2024.

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

[38] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[39] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R. Oswald. Gaussian-slam: Photo-realistic dense slam with gaussian splatting, 2023.

[40] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[41] Jianhao Zheng, Zihan Zhu, Valentin Bieri, Marc Pollefeys, Songyou Peng, and Iro Armeni. Wildgs-slam: Monocular gaussian splatting slam in dynamic environments. *arXiv preprint arXiv:2504.03886*, 2025.

[42] Liyuan Zhu, Shengyu Huang, Konrad Schindler, and Iro Armeni. Living scenes: Multi-object relocalization and reconstruction in changing 3d environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28014–28024, 2024.

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