Real-Time Scene Mapping with 3D Gaussian Splatting

Amine Elhafsi
Stanford University
Department of Aeronautics and Astronautics
amine@stanford.edu

Abstract

Virtually all autonomous robotic system requires an environment representation to perform its intended task. However, a map of the environment is not always available, necessitating the ability to create a map in real time – particularly for mobile and agile systems. To this end, this work explores the feasibility of applying 3D Gaussian splatting to real-time scene mapping. This work implements an approach based on [21] which accelerates mapping by focusing on a local region at a given instant, creating environment submaps. A full map is assembled by gradually appending these submaps to a single persistent global map. By focusing on mapping a local region at a time, we find that this method can operate at roughly 1 Hz with high-resolution images, or 2 to 3 Hz with lower-resolution observations – a promising result towards real-time operation. The code for this project can be found at `https://github.com/AmineElhafsi/CS231A_Project`.

1. Introduction

A key requirement for an autonomous robotic system to navigate or interact with its surroundings is a useful map of its environment. This environment map must accurately encode spatial details to enable the performance of diverse tasks such as motion planning for mobile systems like quadrotors, autonomous vehicles, or rovers, and grasp identification for manipulators. The environment map representation should store information efficiently in a format that facilitates downstream task design and support real-time robot operation. This necessitates the ability to add or modify information based on new observations at a reasonably fast rate, which is particularly important for various robotic systems. For instance, mobile robots often navigate into unexplored areas and must quickly react to their surroundings to avoid collisions.

Inspired by the recent success of 3D Gaussian Splatting (3DGS) [10], a differentiable rasterization approach for novel viewpoint rendering, this work explores the feasibility of applying this method to real-time scene mapping. This method represents a scene as a collection of 3D Gaussians with positions and viewing properties (e.g., color, opacity, etc.) that are optimized to reconstruct the training views. Of interest to this work is the arrangement of these Gaussians in 3D space, which defines the scene geometry.

An environment represented by 3D Gaussian presents several benefits that meet the key desiderata previously put forth. First, the explicit scene representation as a collection of 3D Gaussians is amenable to modification and downstream task design. Map modification is easily achieved through the addition or removal of Gaussians as needed to accommodate new observations or a changing environment. Furthermore, the Gaussians model physical scene elements, which avoids wasting storage representing empty space ensuring that resources are focused only where they are needed. This approach also supports modeling geometry to arbitrary levels of detail,[1] providing detailed and precise environmental maps. Ultimately, this representation produces a collection of 3D Gaussians as an environment representation that can indicate occupied space for navigation, or represent scene geometry and appearance for more complex task reasoning.

Towards a real-time 3DGS-based mapping solution, this work implements an approach based on [21] which accelerates scene mapping by only optimizing a local submap at a given instant. A full scene map is assembled by gradually appending these submaps to a persistent global map. An overview of this approach is shown in Fig. 1.

2. Prior Work

A typical strategy for summarizing a robot's knowledge of its environment is by encoding its observations as an occupancy grid or voxel grid [4, 14]. This representation typically represents occupied and vacant regions of the robot's workspace, making it a convenient explicit representation for motion planning and obstacle avoidance. However, a

[1]This can theoretically be achieved by adding arbitrarily many Gaussians to model a detailed region of space. Of course, this is subject to the data coverage for this region.

key limitation is the resolution of the grid cells which must generally be fixed a priori, although limited adaptability is afforded by hierarchical data structures [7].

To reason about object and environment semantics, robotic perception systems often directly process 2D images by employing an object detector [2, 5, 6] to inform downstream decision making (e.g., stop the autonomous vehicle if a red light is observed), or by training end-to-end pixel-to-action policies [8, 17]. Although these methods have shown some degree of success in robotics, associating object detections with 3D entities in the world requires additional effort. Furthermore, end-to-end trained policies learn implicit internal scene representations that are challenging to interpret. This makes adapting such methods to diverse or novel tasks challenging, and less reliable for safety-critical systems.

In recent years, neural rendering methods have seen a rise in popularity thanks to the emergence of Neural Radiance Fields (NeRF) [1, 12, 13]. These approaches train a neural network to implicitly represent a 3D scene from a sparse set of views. At inference time, the network can be queried for a particular point and viewing direction in space to predict volume density and color at that location. Novel views can then be synthesized by querying the network along camera rays and projecting the resulting output colors and densities onto an image. Initial uses in robotics employed NeRFs to estimate occupied space for collision avoidance [3]. However, for robotics applications one would want to dynamically update or modify the scene representation, particularly as the robot explores new spaces. Although real-time NeRF mapping has been achieved [16], this process is cumbersome and risks catastrophic forgetting of previously mapped regions.

Most recently, 3DGS [10] has emerged as a performant method to efficiently and accurately represent scenes as a collection of differentiable Gaussians that are individually parametrized by a location, scale, color, and opacity. Novel views can by synthesized by projecting each Gaussian into 2D from the camera perspective. The Gaussians can also be augmented to carry additional information as needed, such semantic features as shown in [15, 22] derived from 2D foundation models. Furthermore, it has been shown that 3DGS can be performed in real time to perform localization and mapping [9, 11, 21]. As previously mentioned, 3DGS is also appealing for robotics as it constitutes a world representation that adaptively allocates resolution to a scene through the motion of the particles during optimization and through the ability to add or eliminate particles depending on the level of local detail. Furthermore, the explicit nature of the representation makes it amenable to robotic planning, with the 3D Gaussians' density defining occupied space (without wasting memory encoding unoccupied regions of space).

3. Technical Approach

3.1. Overview

This work focuses on real-time scene mapping with 3DGS. We assume that observations are obtained using an RGB-D camera and that the camera's instrinsics and pose are known at all times.[2] In this work, we consider static scenes for simplicity.

Our implementation takes inspiration from the approach proposed in [21]. In the interest of efficient splatting, this approach represents the overall environment map as the aggregation of multiple submaps. For a given observation (i.e., an RGB-D keyframe), an active submap is updated with the newly acquired information and jointly optimized with previous observations obtained while operating within this submap. As the camera enters new regions, a new submap is initialized while the previous submap is frozen and saved. As such, only a local submap needs to be optimized, eliminating the computational burden of optimizing for spatially or temporally distant observations.

An overview of our implemented method is shown in Fig. 1. At each map update step, a keyframe is used to seed 3D Gaussians within the active submap. Next, the submap (i.e., the constituent Gaussian parameters) are optimized to reconstruct the view from currently and recently observed perspectives. Finally, the method checks whether the current camera pose has moved sufficiently far from the initial pose captured for the active submap. If so, the active submap is saved and merged with the global map, and a new submap is initialized for the next map update.

In the following subsections, we briefly review the technical details of 3DGS. Then, we describe additional specifics of our implemented method, which consists of three steps: Gaussian seeding, submap optimization, and global map merging.

3.2. 3D Gaussian Splatting

As previously mentioned, 3DGS [10] is a differentiable rasterization technique which uses multiple views of a scene to optimize a set of 3D Gaussians to represent the environment permitting the generation of novel views. A 3D Gaussian is parametrized by a mean value $\mu \in \mathbb{R}^3$, a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$, an opacity $o \in [0, 1]$, and a color[3] $[r, g, b] \in \mathbb{R}^3$. Rendering a particular camera view given by the world-to-camera transform $T_{cw} \in SE(3)$ can be performed by projecting (splatting) the visible Gaussians onto

[2]We believe it is reasonable to assume that the pose is known since robotic systems generally perform accurate state-estimation (e.g., using differential GPS, wheel odometry, IMU measurements, etc.) and the camera transformation with respect to the robot's body frame is known from the system's design.

[3]For rasterization, a 3D Gaussian's color content uses spherical harmonics coefficients as an intermediate/internal representation to account for view-dependent object color appearance.

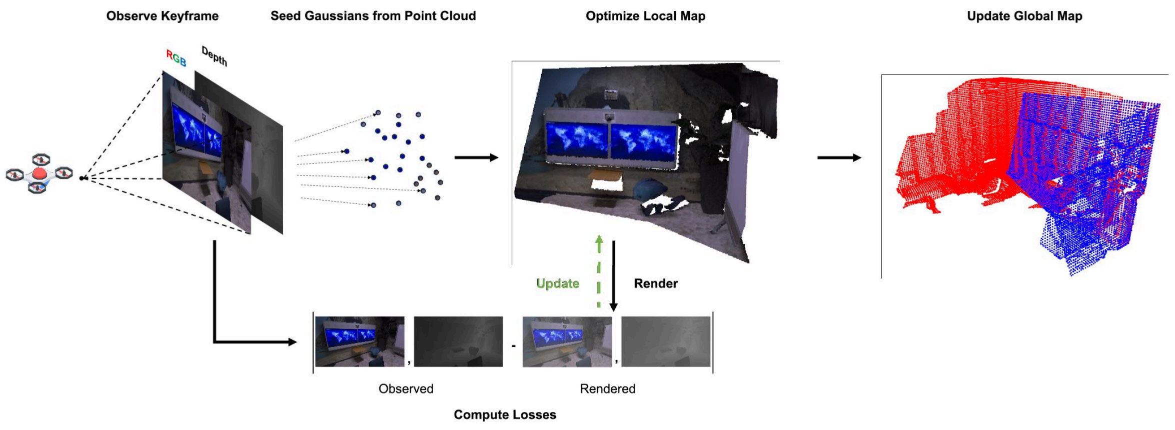

Figure 1. An overview of the implemented real-time 3DGS mapping approach. This performs the following steps at each map update: First, a keyframe is used to add seed 3D Gaussians with the observed pixel colors at the observed positions using the depth measurements. Second, all 3D Gaussians in the submap are optimized over a specified number of iterations by rendering the view from the current and recent observations and minimizing color and depth reconstruction losses. Finally, the method checks whether the current camera pose has moved sufficiently far from the initial pose captured for the active submap. If so, the active submap is saved and merged with the global map (as shown by merging of the blue and red submaps), and a new submap is initialized for the next map update.

the image. The mean can be projected onto the image plane as $\mu^{\text{image}} = M\mu$, where $M \in \mathbb{R}^{3\times4}$ is a projection matrix mapping homogeneous world coordinates to homogeneous image pixel coordinates. Note that M is a function of the camera matrix $K \in \mathbb{R}^{3\times3}$ and camera pose T_{cw}. Similarly, as given by [23], the 3D covariance Σ can be projected to the 2D image as $\Sigma^{\text{image}} = JR_{cw}\Sigma R_{cw}^T J^T$, where $J \in \mathbb{R}^{2\times3}$ is the Jacobian of the affine approximation of the projective transformation and R_{cw} is the rotation matrix extracted from T_{cw}. To maintain positive semi-definiteness of the covariance matrices, we decompose Σ as RSS^TR^T, where R and S are respectively rotation and scaling matrices.

The color c for a given channel at pixel i is computed from m ordered Gaussians (by distance from the camera) as

$$c_i = \sum_{j\leq m} c_j o_j \exp(-\sigma_j)\left(\prod_{k<j}(1 - o_k\exp(-\sigma_k))\right),$$
$$\text{with } \sigma_j = \frac{1}{2}\Delta_j^T \Sigma_j^{\text{image}} \Delta_j.$$

Here, Δ_j represents the distance between the evaluated pixel coordinates and μ_j^{image}. Essentially, this sums the color contribution from each Gaussian accounting for its distribution (i.e., projected mean and covariance) and occluding Gaussians as given by the product term within the summation.

3.3. Gaussian Seeding

To add information to a submap, Gaussians are seeded from a point cloud generated from a subset of the pixels observed in a given RGB-D keyframe. The Gaussians are initialized with the color of the corresponding pixel at corresponding 3D position observed by the depth camera, with covariances initialized isotropically. When a new submap is initialized, Gaussians are seeded from significant subset of the observed pixels (on the order of 50-75% of the total number of image pixels). For an existing submap, significantly fewer Gaussians (10-20% of the total number of image pixels) are instead seeded from pixels corresponding to detected edges[4] or regions of space that have not been observed by previous keyframes. This strategy attempts to focus the mapping on salient scene geometry and previously unseen regions. Furthermore, new Gaussians are only seeded if there are no existing Gaussians within a specified radius around the proposed location to avoid allocating unnecessary information to previously mapped regions.

3.4. Submap Optimization

Additionally, Gaussians that are larger or smaller than specified thresholds are respectively split and cloned, while Gaussians that are below an opacity threshold are pruned.

[4]We take the color image corresponding to the current keyframe, convert it to a grayscale image and apply a Sobel filter along both vertical and horizontal grayscale image directions. We then compute the magnitude of the resulting gradients at each pixel and the resulting grid of magnitudes is normalized to form a probability distribution from which we can sample pixels.

The parameters of these Gaussians are then iteratively optimized via gradient descent to minimize color, depth, and isotropic regularization losses. Compared to [10], the depth and isotropic losses are new. The depth loss ensures that Gaussians do not "float" in the scene, but rather remain localized to physical scene surfaces. The isotropic loss discourages Gaussians from becoming excessively long or skinny. Formally, the overall loss function is

$$L = \lambda_{\text{color}} \cdot L_{\text{color}} + \lambda_{\text{depth}} \cdot L_{\text{depth}} + \lambda_{\text{isotropic}} \cdot L_{\text{isotropic}},$$

where

$$\begin{aligned} L_{\text{color}} &= (1-\lambda) \cdot \|\hat{I} - I\|_1 + \lambda \cdot (1 - \text{SSIM}(\hat{I}, I)) \\ L_{\text{depth}} &= \|\hat{D} - D\|_1 \\ L_{\text{isotropic}} &= \frac{\sum_{k \in K} \|s_k - \bar{s}_k\|_1}{|K|}. \end{aligned}$$

Here, I and D correspond to the ground truth color and depth images, respectively. The variable $s_k \in \mathbb{R}^3$ represents the scale of Gaussian k from which we construct the scaling matrix S_k (from the aforemention covariance matrix decomposition) as $\text{diag}(s_k)$. A hat indicates the corresponding reconstruction. The depth at pixel i is rendered in a similar manner to color; for m ordered Gaussians D_i is computed as

$$D_i = \sum_{j \leq m} \mu_j^z \cdot o_j \exp(-o_j) \cdot \left(\prod_{k<j} (1 - o_k \exp(-o_k)) \right),$$

where μ_j^z is the z-coordinate of the mean of Gaussian j. We set the weighting parameters $\lambda_{\text{color}} = \lambda_{\text{depth}} = \lambda_{\text{isotropic}} = 1$ and $\lambda = 0.2$.

3.5. Global Map Merging

To ensure computational efficiency, only a local "submap" is optimized at each iteration. This means that only Gaussians corresponding to the camera's local vicinity are updated from a recent set of keyframes. A submap starts from an initial reference camera pose and grows incrementally as new keyframes are observed and Gaussians are added to the map and optimized.

Once a submap accumulates more than a specified number of frames, or the camera surpasses a distance or rotation threshold (e.g., in this work, 2 meters translation or 45 degrees rotation in any Euler angle), the active submap is saved to a global map stored to disk freeing GPU memory. This process is quite straightforward and appends the Gaussians, along with associated parameters, to the existing global map up to that point. Once this is complete, a new camera reference pose is registered from the robot's instantaneous state and this process starts again with the next submap. This strategy is suggested to bound the mapping computation cost to ensure that the Gaussian parameter optimization remains fast [21].

Optionally, after mapping is completed, all submaps may be jointly refined by performing additional gradient steps using all observed keyframes. Additional post-processing may be performed to prune Gaussians from regions where submaps may overlap to further reduce the scene storage size.

4. Evaluation and Results

In this section, evaluate our implementation along several dimensions. We evaluate the scene reconstruction quality and runtime performance of our method on two datasets: the Replica dataset [19] a synthetic, high-resolution dataset, and the TUM-RGBD Dataset [20], a real-world dataset, consisting of lower-resolution images. Dataset information is provided in Tab. 1 along with the corresponding hyperparameters used for Gaussian splatting in Tab. 2.

First, we present a set of quantitative results that evaluate the image rendering quality along with visualizations to illustrate some of the reconstructed scenes. Finally, we present and discuss runtime statistics and show that our method approaches real-time performance.

4.1. Scene Reconstruction Results

To evaluate the quality of the mapped scene, we generate renders of our mapped scenes from a set of validation viewpoints and compute the standard Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) metrics. Given that the datasets used consist of a fixed sequence of keyframes (i.e., we do not have access to the environments used to generate them), for our validation set we select the $N-1$ unseen inbetween observations that lie between our N training frames. We present rendering quality results in Tab. 3 and Tab. 4 for the Replica and TUM-RGBD datasets, respectively. We note that our results match with those found in [21], validating our implementation. We also present visualizations of an `Office0` and `Freiburg1 Desk` frame along with their respective reconstructions in Fig. 2.

Interestingly, we find that our implementation generally outperforms [21] in the Replica dataset scenes despite having used similar hyperparameters (Tab. 2) during mapping. It is believed that this performance difference is due to a bug in the implementation provided by [21]. Specifically, at each map update Gaussians should be added to the scene at salient regions (e.g., image gradients or regions that have not been observed by previous viewpoints as described in Sec. 3.2). However, their code does not correctly identify the existing Gaussians within the camera frustum of the current training view and as such redundantly seeds Gaussians to these previously mapped regions. This is a relatively

Table 1. Dataset information.

Dataset Name	Image Height	Image Width	Num. Scenes	Num. Training Frames
Replica [19]	680	1200	3	100 per scene
TUM-RGBD [20]	480	640	3	{167, 1030, 750}

Table 2. Hyperparameters used for 3DGS scene mapping.

Dataset Name	Submap Initial Num. Gaussians	Num. Seeded Gaussians Per Frame	Submap Initialization Gradient Steps	Per Frame Gradient Steps
Replica [19]	600000	100000	1000	100
TUM-RGBD [20]	100000	50000	200	100

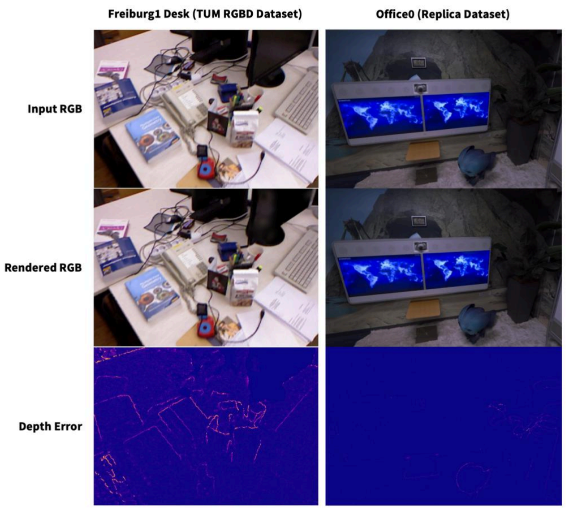

Figure 2. Visualizations of frames from the `Freiburg1 Desk` (left) and `Office0` (right) scenes. In particular, notice that the Gaussian scene representation captures high-frequency scene detail, as seen in the text on the books (left) and in the map and wall art detail (right).

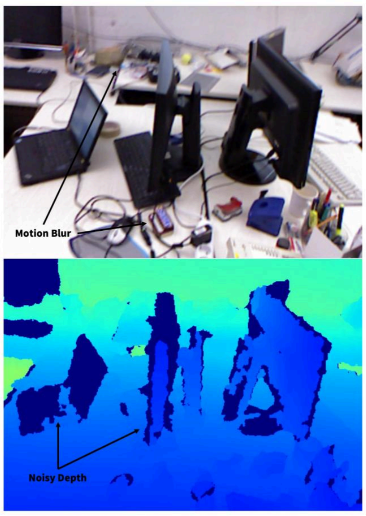

Figure 3. An example of a noisy observation from the Freiburg1 Desk scene from the TUM-RGBD dataset. Note the motion blur in the background and certain foreground elements such as the laptop and chocolate bar. Similarly, the depth measurements are quite noisy and do not always accurately match the scene contents.

minor, non-catastrophic issue in the broader algorithm, but seems to have non-negligible impact particularly in the high resolution Replica scenes.

On the other hand, both our implementation and [21] see significantly lower PSNR and SSIM values in the TUM-RGBD scenes. We attribute this to both the lower resolution training images, which results in a lower resolution map, and due to the lower quality images. The latter is due to the fact that this is a real-world dataset subject to camera issues such as motion blur as well as noisy depth measurements, examples of which are depicted in Fig. 3. Our implementation is generally outperformed by [21] in these scenes despite, again, using similar hyperparameters. Aside from countless subtle or inscrutable implementation differences, it is possible that this discrepancy arises due to different data preprocessing. In this work, we performed scene mapping with the raw observations, while it is possible that [21] filters and eliminates noisier images.

Finally, we present visualizations of the resulting maps of the `Office3` and `Room0` scenes from the Replica dataset in Fig. 4.[5] We visualize these scenes as they demonstrate desirable environment map characteristics for robotics applications.

In particular, we note that fine geometry is captured such as the potted plant in the `Room0` map. Additionally, the

[5]Note that these images depict the Gaussians in the scene and are *not renders* (i.e., rasterized Gaussians) from the shown viewpoint.

Table 3. Rendering quality results for the Replica dataset [19]. In this table, we present the mean value of the corresponding metric and include the corresponding metric obtained in [21] in parentheses.

Scene Name	**Office0**	**Office1**	**Office2**	**Office3**	**Room0**	**Room1**	**Room2**
PSNR	50.3 (38.9)	48.8 (41.8)	44.4 (42.4)	43.0 (46.4)	43.4 (40.1)	45.4 (39.1)	46.3 (42.7)
SSIM	0.994 (0.993)	0.994 (0.996)	0.990 (0.996)	0.988 (0.998)	0.986 (0.997)	0.990 (0.997)	0.990 (0.997)

Table 4. Rendering quality results for the TUM-RGBD dataset [20]. In this table, we present the mean value of the corresponding metric and include the corresponding metric obtained in [21] in parentheses.

Scene Name	**Freiburg1 Desk**	**Freiburg2 XYZ**	**Freiburg3 Long Office**
PSNR	20.8 (24.0)	23.7 (25.0)	24.8 (26.1)
SSIM	0.797 (0.924)	0.833 (0.924)	0.857 (0.939)

depth loss ensures that Gaussians are localized to physical objects, which minimizes the presence of floating Gaussians in the scene representation, a common issue scene in [10]. This is particularly important for robotic navigation as such "floaters" could be interpreted as obstacles by a downstream motion planner. Similarly, the isotropic regularization term ensures that Gaussians do not become overly long or skinny during optimization, which is relevant to manipulation tasks as it ensures that object geometry can be accurately modeled by a collection of Gaussians' centers (e.g., for grasp identification).

In these maps, we also observe that there are some gaps present in the scene. This is due to the fact that the datasets imperfectly covered the scene, either due to occlusion or due to a limited scene traversal. For online robotic mapping, this introduces an additional challenge: ensuring that the robot's observations sufficiently cover the scene for its task. However, we defer this challenge to future work.

4.2. Runtime Performance

In our experiments, we also recorded computation time statistics to evaluate the feasibility of such an approach for online operation. Over all Replica dataset scenes, we find that the average submap initialization time is 10.2 seconds (st. dev. 0.29 seconds), with subsequent map updates taking 1.06 seconds on average (st. dev. 0.078 seconds). Over the TUM-RGBD scenes, the average submap initialization time is 0.75 seconds (st. dev. 0.055 seconds), with subsequent map updates averaging 0.37 seconds (st. dev. 0.023 seconds). Submap initialization notwithstanding, these results indicate that this mapping approach can run at approximately 1 Hz with high-resolution observations and between 2 to 3 Hz with lower resolution observations, which are promising figures for an initial exploration of this method.

We plot the computation time per update over an entire mapping sequence from each dataset in Fig. 5. We note that the submap initialization imposes a high computation cost in Replica scenes due to the large number of Gaussians seeded. However, the computation time does not vary significantly in subsequent frames once the submap is already established. This suggests that the criteria set for initializing a new submap were too stringent; a strategy that reinitializes the submap less frequently is possible and would likely mitigate such spikes in computation time greatly. In contrast, the submap initialization is much less prohibitive for TUM-RGBD scenes as these scenes seed fewer Gaussians and require a fraction of the optimization steps to initialize a submap (see Tab. 2).

We also evaluate the effect of map size, as measured by the number of Gaussians modeling the scene, on optimization step time in Fig. 6. For smaller numbers of Gaussians (as in the TUM-RGBD scenes), the optimization step times appear relatively insensitive to map size and cluster around a lower bound likely determined by PyTorch overhead. In contrast, once the map size grows sufficiently, the step time appears to scale linearly with map size. This linear trend appears roughly consistent for range of map sizes resulting from the Replica scenes. This suggests that a hierarchical tree-based data structure [18] (e.g., analogous to octrees for grid-based representations) could benefit computation time for sufficiently large scenes or scenes with significantly varying scales of detail.

5. Conclusion

In this work, we explored the feasibility of applying 3DGS to real-time scene mapping, motivated by applications to autonomous robotics. This is also relevant to settings beyond robotics such as construction inspection, disaster site surveying for search and rescue, or the mapping of historical or archaeological sites. By focusing on mapping a local region at a time, we find that this method can operate at roughly 1 Hz with high-resolution imagery, or 2 to 3 Hz with lower-resolution observations.

Immediate improvements to this work could explore the development of keyframe selection heuristics. In this work, we simply optimize the map based on keyframe stream in the sequence provided. However, when multiple keyframes

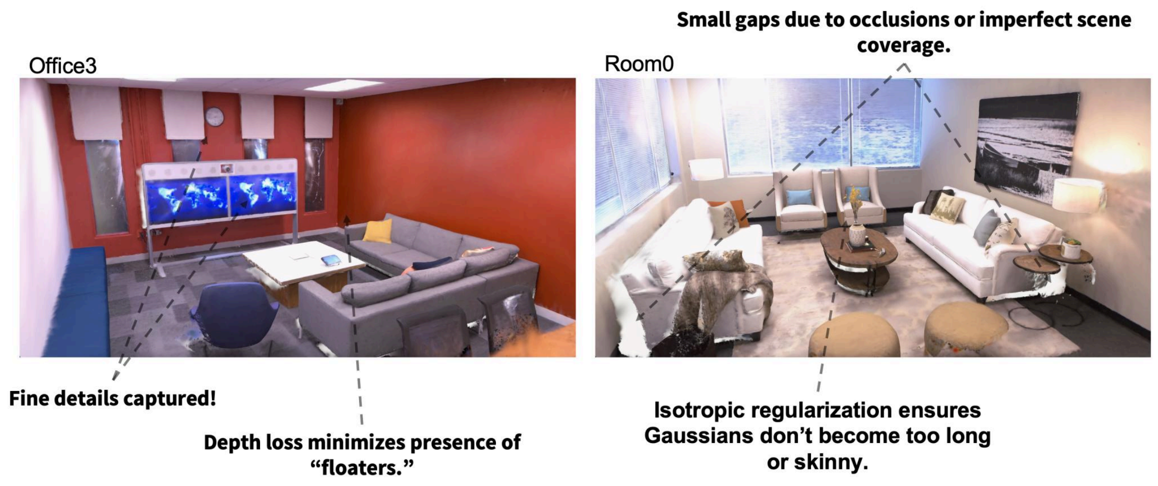
Office3
Room0
Small gaps due to occlusions or imperfect scene coverage.
Fine details captured!
Depth loss minimizes presence of "floaters."
Isotropic regularization ensures Gaussians don't become too long or skinny.

Figure 4. Gaussian splat maps of the Replica `Office3` (left) and `Room0` (right) scenes. Note that these images depict the Gaussians in the scene and are not renders (i.e., rasterized Gaussians) from the shown viewpoint.

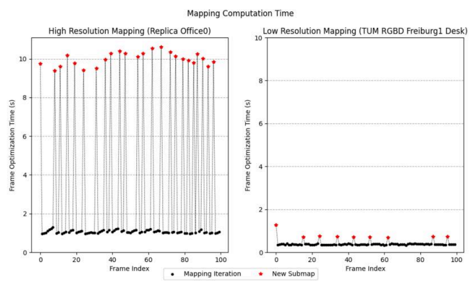

Figure 5. Computation time vs. map update step for two mapping sequences (Replica `Office0` left, TUM-RGBD `Freiburg1 Desk` right).

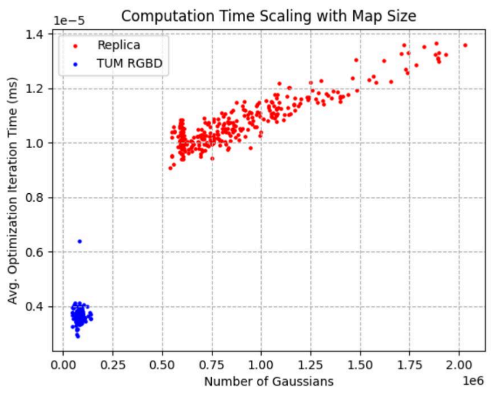

Figure 6. Optimization *iteration* time (i.e., gradient step) vs. map size measured as the number of Gaussians present.

are too similar, computation time is wasted for minimal information gain. We also find that submap initialization time can be quite costly, especially when a large number of Gaussians need to be seeded, which could be addressed by tuning the submap initialization strategy. Alternatively, a more sophisticated approach could explore "continuously" updating the submap as opposed to the current discrete strategy. This would involve maintaining a set of "optimizable Gaussians" (i.e., a persistently active submap) to which we could dynamically add Gaussians to new regions or removing Gaussians that are far away and out of view (of course, saving them to a global map).

Longer term research directions could consider identifying and handling dynamic objects in real-time. Another direction, particularly relevant to robotics, could explore embedding semantics in the scene map to develop more sophisticated environment interactions (e.g., segment and locate specific objects, semantic or social context-dependent behaviors, etc.).

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 2

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[3] Timothy Chen, Preston Culbertson, and Mac Schwager. Catnips: Collision avoidance through neural implicit probabilis-

tic scenes. *IEEE Transactions on Robotics*, 2024. 2

[4] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 1

[5] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 2

[7] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34:189–206, 2013. 2

[8] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zak Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 2

[9] 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. *arXiv preprint arXiv:2312.02126*, 2023. 2

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. 1, 2, 4, 6

[11] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. *arXiv preprint arXiv:2312.06741*, 2023. 2

[12] 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. In *ECCV*, 2020. 2

[13] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 2

[14] Trung-Tin Nguyen, Phuc Nguyen, Chuong Nguyen, and Minh Tran. Examination of sampling-based path planning for indoor uav using voxel grid-based visual slam. In *Proceedings of the 8th International Conference on Robotics and Artificial Intelligence*, pages 26–31, 2022. 1

[15] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. *arXiv preprint arXiv:2312.16084*, 2023. 2

[16] Antoni Rosinol, John J Leonard, and Luca Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3437–3444. IEEE, 2023. 2

[17] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR, 2022. 2

[18] Qing Shuai, Haoyu Guo, Zhen Xu, Haotong Lin, Sida Peng, Hujun Bao, and Xiaowei Zhou. Real-time view synthesis for large scenes with millions of square meters. 2024. 6

[19] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 4, 5, 6

[20] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012. 4, 5, 6

[21] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R Oswald. Gaussian-slam: Photo-realistic dense slam with gaussian splatting. *arXiv preprint arXiv:2312.10070*, 2023. 1, 2, 4, 5, 6

[22] Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suya You, Zhangyang Wang, and Achuta Kadambi. Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. *arXiv preprint arXiv:2312.03203*, 2023. 2

[23] 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. 3