https://doi.org/10.15407csc.2024.02.003
UDC 004.932

A.O. SMIRNOV, PhD Student, Senior Researcher, International Research and Training Center for Information Technologies and Systems of the NAS and MES of Ukraine.
40, Akademika Glushkova Avenue, Kyiv 03187, Ukraine,
ORCID: https://orcid.org/0009-0002-6509-4135,
tonysmn97@gmail.com

DYNAMIC MAP MANAGEMENT FOR GAUSSIAN SPLATTING SLAM

Map representation and management for Simultaneous Localization and Mapping (SLAM) systems is at the core of such algorithms. Being able to efficiently construct new KeyFrames (KF), remove redundant ones, and constructing covisibility graphs has a direct impact on the performance and accuracy of SLAM. In this work, we outline the algorithm for maintaining a dynamic map and its management for the SLAM algorithm based on Gaussian Splatting as the environment representation. Gaussian Splatting allows for high-fidelity photorealistic environment reconstruction using differentiable rasterization and can perform in real-time making it a great candidate for map representation in SLAM. Its end-to-end nature and gradient-based optimization significantly simplify map optimization, camera poses estimation, and KeyFrame management.

Keywords: *radiance fields, scientific computing, slam, bundle adjustment, gaussian splatting, differentiable rendering.*

Introduction

Simultaneous Localization and Mapping [5] is an algorithm that performs reconstruction of the environment using input data from sensors on an agent and simultaneously localizes that agent in the map.

SLAM algorithms are used for robotic navigation and mapping [8] and augmented reality experiences. Depending on the requirements and operational constraints sensors from which the input data is retrieved vary significantly and thus the approaches to solving the problem. However, cameras are one of the most appealing types of sensors as they are extremely cheap and easily available. Algorithms that utilize only cameras are known as Visual SLAM [8], [1] and depending on the camera setup are divided into monocular (single camera), stereo (two-camera setup), and multi-camera setup.

This work uses a monocular camera setup as it is the most challenging (due to ambiguities) yet most appealing in terms of its applicability.

Besides images as input data, the only known parameters we require are the camera's intrinsics: focal length f_{xy}, principal point C_{xy}, and lens distortion coefficients q.

In this setup, input data at each time-step t is an image

$$I_t \in R^{CxWxH},$$

where C is the number of channels in the image, W and H are its width and height.

Cite: Smirnov A.O. Dynamic Map Management for Gaussian Splatting SLAM. *Control Systems and Computers*, 2024, 2, 3—17. https://doi.org/10.15407/csc.2024.02.003

Frame F_t at time-step t is a bundle of current image I_t and a current camera pose estimate T_t.

It is used primarily to refine camera pose T_t before proceeding to the next frame at timestep $t + 1$. However, we cannot build a map of the environment by storing every frame as it is computationally intractable, especially during global map optimization. Therefore, we maintain a list of Key Frames KF_i which are constructed from frames F_t and contain its camera pose T_i, image I_i, and a time step t at which it was created. A map is then a set of Key Frames that optimally represent the environment.

KeyFrame creation requires careful choices as it directly impacts the size of the map and therefore the computational complexity of certain operations with it such as Global Bundle-Adjustment (GBA) or Loop-Closure (LC). Optimal KeyFrame creation algorithms distribute KeyFrames in the space such that the map does not contain redundant KeyFrames yet the camera pose estimation is still able to accurately recover poses.

To perform fast local Bundle-Adjustment [2] on the most recently added KeyFrames and to make choices on new KeyFrame creation we maintain a covisibility graph which consists of K most recent KeyFrames that observe the same part of the environment, plus several additional randomly selected KeyFrames.

Besides KeyFrames, the map consists of Gaussians which are observed by each KeyFrame. The number of Gaussians grows with each new KeyFrame, therefore it is important to insert new KeyFrames only when required, avoiding unnecessary ones that would impact the performance of the SLAM system.

Recently, neural volumetric representations have become a popular choice for environment reconstruction [6, 7] due to their ability to represent the environment with high quality having unified object representation and utilizing differentiable rendering and gradient-based optimization methods that make easily parallelizable leading to efficient implementations on GPUs. However, while some of them encode the environment implicitly, using the weight of the neural network, Gaussian Splatting algorithm [3] has the advantage of explicit representation using 3D anisotropic Gaussians. Additionally, while others use ray marching for the image formation process, it uses differentiable rasterization which significantly improves the performance and therefore is a great candidate for map representation in SLAM algorithms.

In the next section, we give a brief overview of the Gaussian Splatting algorithm. Afterward, each step of map management using Gaussians is described in detail.

Gaussian Splatting

SLAM system in this work uses Gaussian Splatting for the environment representation [3]. Such choice allows for high-fidelity environment reconstruction using differentiable rasterization and gradient-based methods for parameter optimization.

Each Gaussian G_i is defined by its mean μ_i and covariance Σ_i which define its position and orientation in the space. Additionally, color c_i and opacity a_i are defined. Contrary to the original Gaussian Splatting algorithm, which defines color via spherical harmonics, here color is defined as a simple RGB vector both for simplicity and performance. By using RGB vector, we disregard view-dependent effects, such as reflectance.

To render an image using Gaussian Splatting, we perform rasterization and compute colors for each pixel in the image as follows:

$$C(x, y) = \sum_{i=1}^{N} c_i a_i \prod_{j=1}^{i-1} (1 - a_i). \tag{1}$$

We iterate over N Gaussians and splat their 2D representations to compute the final color. Since their positions are known, there is no need to skip empty spaces, making it very efficient.

Before rasterizing Gaussians, we must transform them from the world frame to the camera frame using world-to-camera transformation T_{WC} for a given camera pose T_i:

$$\begin{aligned} \mu_c &= \pi_K \left(T_{WC} \mu_i \right), \\ \Sigma_c &= JW\Sigma_i W^T J^T, \end{aligned} \tag{2}$$

where π_K is the projective transformation given camera intrinsics K, W is the rotational compo-

nent of T_{WC} and J is the Jacobian of the linear approximation of the projective transformation.

Similarly, we can render an uncertainty image, which for each pixel sets a value $u \in [0, 1]$, specifying how reliable this pixel is:

$$U(x, y) = \sum_{i=1}^{N} a_i \prod_{j=1}^{i-1} (1 - a_j). \quad (3)$$

A value close to 0 means that the pixel is unreliable, while 1 means that it reliable. This information can then be used to weight loss term during camera pose optimization or Bundle-Adjustment.

Additionally, given a set of Gaussians G_i we want to know what Gaussians are visible and participated in the image formation during rendering for a given camera pose T_{WC}. To do that, we compute Boolean visibility vector, where i-th value is set to true for G_i Gaussian if that Gaussian participates in rendering and accumulated transmittance is <0.5 during front-to-back splatting. This means we discard occluded Gaussians if the pixel opacity is already ≥0.5.

During map optimization, Gaussians are split or pruned depending on several criteria, which allows more precise environment representation and filters out unreliable Gaussians that violate multi-view consistency.

Finally, a loss is a simple $L2$ loss between the target Frame image I_i and a rendered image I. Afterwards, a gradient is computed w.r.t. Gaussian parameters or camera poses and Adam optimizer [4] is used to update them.ˆ

Dynamic Mapping

The goal of the Mapping stage is to create a map of the environment given ordered in the time sequence of images. We start the process by initializing the map with a KeyFrame that has an identity pose.

Since in the monocular RGB scenario, the depth for each frame is not available, we initialize Gaussians randomly in front of the KeyFrame, by back-projecting pixels from the image plane and assigning random noise in $[t_{near}, t_{far}]$ range. We then randomly subsample those Gaussians and compute scales for each of them by taking the mean distance to 3 closest neighbors as in [3]. Colors are initialized from respective pixel values.

We then perform initial optimization for 500 iterations to allow Gaussians to properly represent the image from that camera pose, even if the depth is incorrect.

KeyFrame Insertion

As new Frames arrive, the system tracks and updates its camera pose using them until it requests for a new KeyFrame to be inserted in the map. Only KeyFrame insertion allows the map to grow and adds new Gaussians to previously unseen regions. So when a new KeyFrame is requested it is usually due to a lack of Gaussians in that region. However, there are several criteria that may trigger KeyFrame insertion:

However, several criteria may trigger KeyFrame insertion:

- Time check: if the map contains less than K KeyFrames (uninitialized) and the last KeyFrame was added more than 5 timesteps ago.
- Distance check: if the distance between the current Frame and the last KeyFrame is bigger than some distance threshold. Distance is computed as follows:

$$t = \left(T_{F,WC} \cdot T_{KF}\right)\left[\left[1, 2, 3\right], 4\right], \quad D = \left\| t \right\|, \quad (4)$$

where $T_{F,WC}$ is the world-to-camera transformation for the current Frame F, $T_{F,WC}$ is the camera-to-world transformation for the last KeyFrame KF. The criteria is then computed as $D > \theta_D$.

- Visibility check: if the current Frame sees less than some percentage of Gaussians that are observed by the last KeyFrame. Visibility check is computed as Intersection-over-Union (IoU) of the current Frame visibility vector V_F and last KeyFrame visibility vector V_{KF}:

$$IoU = \frac{V_F \cap V_{KF}}{V_F \cup V_{KF}}. \quad (5)$$

The criteria is then computed as $IoU < \theta_{IoU}$.

Fig. 1. Map immediately after first KeyFrame insertion: (left) target image; (center) color rendering mode; (right) depth rendering mode

Fig. 2. Example of map state during Bundle-Adjustment: (left) Gaussians before Bundle-Adjustment and densification or pruning; (middle) Gaussians after pruning removed unstable points; (right) Gaussians after Bundle-Adjustment and densification

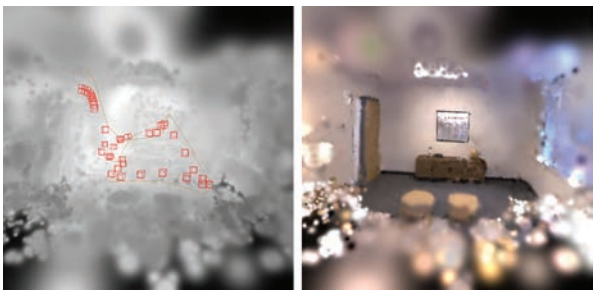

Fig. 3. Camera trajectory: (left) visualized trajectory in-depth rendering mode; (right) color rendering mode

We consider the map initialized when it has at least K KeyFrames, where K is equal to the size of the covisibility graph (e.g. 8).

Bundle Adjustment

During KeyFrame insertion and every mapping step, we optimize both KeyFrame positions and Gaussians.

Since optimizing the whole map is intractable we maintain covisibility graph of the most recent KeyFrames W and a randomly selected subset from the map S, so the optimization is done over $B = W \cup S$ KeyFrames.

To enforce multi-view consistency we rasterize Gaussians into all image planes from B at once. This means that we get $|B|$ images for which the loss is computed with respect to image in each KeyFrame and finally aggregated.

$$L = \sum_{i=1}^{(B)} \left\| R_i \left| G \right| - I_i \right\|^2, \tag{6}$$

where R_i is the Gaussian rasterizer initialized with a pose from i-th KeyFrame from B covisibility graph, I_i is the respective image from i-th KeyFrame and G is the set of all Gaussians.

The gradient is then computed over L with respect to Gaussians G and KeyFrames poses from W. We do not update poses for randomly selected KeyFrames in S since for every mapping iteration we re-sample the set S to encourage consistency across the whole map.

Because the gradient is computed over all images at once it enforces stronger multi-view consistency than when computing gradient over individual frames. However, the downside is that it requires $O(|B|)$ memory.

Additionally, we periodically run densification and pruning for Gaussians as in [3] to encourage precise and more robust environment representation.

Running Bundle-Adjustment as often as possible reduces accumulated errors in camera poses and improves accuracy. Figure 3 visualizes the map state including KeyFrames (red boxes), rough trajectory (orange lines) that the camera took, and the Gaussians themselves. The trajectory is rough because we connect only KeyFrames and not every frame that is processed.

Gaussian Pruning

During densification and KeyFrame insertion, many newly added Gaussians will violate multi-view consistency and therefore impact camera pose estimation. Some of them will vanish when pruning during Bundle-Adjustment, if:

- Gaussians occupy too much region in pixel space after rasterization;
- opacity of the Gaussian is less than some small value ϵ;
- accumulated gradient for the Gaussians or their scale in the world space is below some threshold.

However, we can improve on the pruning Gaussians that are observed by less than K KeyFrames. As during KeyFrame insertion or densification some of the Gaussians may be too noisy, but for camera pose estimation we want to have only robust Gaussians as they directly impact the tracking accuracy.

Therefore, for each KeyFrame in the covisibility graph W, we compute visibility vector and count how many KeyFrames observe each Gaussian. If the Gaussian is observed by less than K KeyFrames, we prune it as it is too unreliable.

Odometry Mode Pruning

Often, we may be interested only in camera pose and trajectory estimation and not in actual environment reconstruction as is often the case for SLAM algorithms.

In this case, we consider pruning Gaussians that are far away from the most recent KeyFrames in covisibility graph W and that are not observed by any such KeyFrame during rendering.

This helps both with performance (during rasterization we test each Gaussian if it is visible) and memory consumption especially if we are tracking over big regions. Such Gaussians can be offloaded from GPU and stored on disk for later retrieval if ever needed.

Results and Limitations

The table below showcases the information about the resulting map after running SLAM on different datasets. Replica [11] room 0 is a purely synthetic dataset where all images have the same exposure, no motion blur, and smooth camera motions. However, it contains lots of purely rotational camera movements which are good for testing the tracking capabilities of the SLAM.

The other two TUM [12] freiburg1 desk and TUM freiburg2 xyz are real-world datasets with varying exposure, motion blur, and rapid camera motions. As can be seen, TUM freiburg1 desk contains almost the same amount of KeyFrames as Replica room 0 while having almost 3.2 times fewer frames. This is expected as the more sporadic motions dataset contains the more KeyFrames is needed to accurately track the camera and represent the environment where KeyFrames act as anchor points against which SLAM optimizes.

As the map grows so does the amount of Gaussians which we need to test for visibility every time we render an image. Especially if we are interested both in camera pose and trajectory estimation and environment reconstruction, where we maintain all Gaussians.

At some point, testing for visibility of each Gaussian takes a significant amount of time as it requires projecting each Gaussian from world to the camera space and testing its depth as well as if it projects onto the image plane itself. And of course, it takes a significant amount of memory.

Table. **Comparison of the resulting map size after processing different datasets**

Dataset	Total Frames	Total KeyFrames	Number of Gaussians
Replica room 0	2000	129	6240
TUM freiburg1 desk	613	114	6320
TUM freiburg2 xyz	3669	231	10426

For future directions, it can be fruitful to investigate the dynamic construction of volumetric trees (Octree [9], BVH) or construction of level of details (LoD) for Gaussians [10] to avoid excessive visibility testing and provide stable rendering time.

Volumetric trees would partition Gaussians in space allowing us to efficiently filter out large chunks of Gaussians that are not visible without testing each Gaussian separately. While Level of Details would work in a coarse-to-fine fashion, replacing fine-detailed Gaussians with rough approximations of lower detail when the camera is located far away from those Gaussians and vice versa.

This should significantly speed up the processing time, although dynamically managing such trees comes with its own challenges.

Conclusions

In conclusion, this work presents an algorithm for dynamic map management for a Simultaneous Localization and Mapping (SLAM) problem that utilizes Gaussians as an environment representation.

Gaussian Splatting is an appealing choice for environment representation as it is both efficient and results in high-quality reconstructions utilizing rasterization and gradient-based methods for optimization and thus can be efficiently executed on GPUs.

We describe key operations that need to be performed during SLAM and the impact each operation has on the resulting map and showcase results.

REFERENCES

1. Campos, C., Elvira, R., Rodríguez, J.J.G., Montiel, J.M., & Tardós, J.D. (2021). "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam". *IEEE Transactions on Robotics*, 37 (6), pp. 1874—1890. DOI: 10.1109/tro.2021.3075644.
2. Chen, Y., Chen, Y., & Wang, G. (2019). "Bundle adjustment revisited". *arXiv preprint arXiv*:1912.03858.
3. Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). "3d gaussian splatting for real-time radiance field rendering". *ACM Transactions on Graphics*, 42 (4), pp. 1—14. arXiv: 2308.04079.
4. Kingma, D.P., Ba, J. (2017). "Adam: A Method for Stochastic Optimization". https://doi.org/10.48550/arXiv.1412.6980.
5. Lim, K.L., & Braunl, T. (2020). "A Review of Visual Odometry Methods and Its Applications for Autonomous Driving". arXiv 2020. arXiv preprint arXiv:2009.09193.
6. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., & Ng, R. (2021). "Nerf: Representing scenes as neural radiance fields for view synthesis". Communications of the ACM, 65 (1), pp. 99—106. arXiv: 2003.08934.
7. Muller, T., Evans, A., Schied, C., & Keller, A. (2022). "Instant neural graphics primitives with a multiresolution hash encoding". *ACM transactions on graphics* (*TOG*), 41 (4), pp. 1—15. DOI: 10.1145/3528223.3530127.
8. Mur-Artal, R., Montiel, J. M.M., & Tardos, J.D. (2015). "ORB-SLAM: a versatile and accurate monocular SLAM system". IEEE transactions on robotics, 31 (5), pp. 1147—1163. DOI: 10.1109/tro.2015.2463671.
9. Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., & Dai, B. (2024). "Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians". arXiv preprint arXiv:2403.17898.
10. Shuai, Q., Guo, H., Xu, Zh., Lin, H., Peng, S., Bao, H., Zhou, X. (2024). "Real-Time View Synthesis for Large Scenes with Millions of Square Meters". [online]. Available at: https://zju3dv.github.io/LoG_webpage/ [Accessed 01 March. 2024].
11. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., ... & Newcombe, R. (2019). "The Replica dataset: A digital replica of indoor spaces". arXiv preprint arXiv:1906.05797.
12. Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012). "A benchmark for the evaluation of RGB-D SLAM systems". In 2012 IEEE/RSJ international conference on intelligent robots and systems, pp. 573—580.

Received 15.03.2024

А.О. Смирнов, аспірант, Міжнародний науково-навчальний центр
інформаційних технологій і систем НАН та МОН України,
03187, м. Київ, просп. Академіка Глушкова, 40, Україна,
ORCID: https://orcid.org/0009-0002-6509-4135,
tonysmn97@gmail.com

ДИНАМІЧНА ПОБУДОВА МАПИ
ДЛЯ АЛГОРИТМУ *SLAM* НА ОСНОВІ ГАУСИАН

Вступ. У задачах комп'ютерного зору та робототехніки часто виникає необхідність знаходження розташування агента в середовищі. Проте, зазвичай мапа навколишнього середовища наперед не є відомою. Тому виникає необхідність одночасної побудови мапи та локалізації агента в ній, а для розв'язання такої задачі застосовують алгоритм Одночасної Локалізації та Картографування (*Simultaneous Localization and Mapping*)

У випадку, якщо мапу необхідно використовувати не лише для задач локалізації, така мапа має бути високоякісною та чітко представляти навколишнє середовище. Нещодавні методи для реконструкції навколишнього середовища на основі диференційованого об'ємного рендерингу, такі як Нейронні Поля Випромінювання (*Neural Radiance Fields*) та Накладання Гаусиан (*Gaussian Splatting*) дають змогу отримувати фотореалістичні результати. Проте, обмеженням таких методів є те, що вони вимагають наперед відомі дані про розташування камер та відомих наборів зображень, які використовуватимуться для реконструкції.

Диференційовані алгоритми об'ємного рендерингу дають змогу сформувати задачу, за якої розташування камер не є необхідною умовою для роботи алгоритму та може бути обчислено одночасно з побудовою мапи або в процесі надходження нових зображень з камери на основі попередніх спостережень.

Побудова та вибір представлення мапи є ключовим етапом таких алгоритмів.

Мета. Метою роботи є представлення алгоритму побудови динамічної мапи та її подальше використання алгоритмом Одночасної Локалізації та Картографування (*SLAM*) та методу Накладання Гаусиан для представлення навколишнього середовища.

Можливість ефективно додавати нові Ключові кадри (*Keyframes*), прибирати надлишкові та конструювати графи спів-видимості (*co-visibility*) має безпосередній вплив на швидкість роботи та точність *SLAM* алгоритмів.

Методи. Для розв'язання задачі було використано алгоритми Накладання Гаусиан та Градієнтного спуску.

Для реалізації роботи алгоритму було використано мову програмування *Julia*, яка має широку підтримку графічних процесорів *GPU* та дозволяє їх агностичне програмування, що значно спрощує саму реалізацію.

Результати. Експериментальна реалізація цього алгоритму показала ефективність такого підходу. Метод Накладання Гаусиан дає змогу здійснювати високоякісну реконструкцію навколишнього середовища з використанням диференційованої растеризації та підтримує роботу в режимі реального часу, що робить його чудовим кандидатом для побудови мапи в алгоритмах *SLAM*.

Можливість наскрізної оптимізації на основі методу градієнтного спуску значно спрощує оптимізацію побудованої мапи, оцінку положення камери та роботу з Ключовими кадрами. Використання графічних обчислювальних процесорів *GPU* дає змогу працювати в режимі реального часу, що сприяє її практичному застосуванню.

Висновки. Експерименти показали доцільність запропонованого алгоритму динамічної побудови мапи навколишнього середовища з використання методу Накладання Гаусиан. Подальша робота алгоритму може бути покращена завдяки знаходженню циклів у траєкторії камери та подальшій оптимізації реалізації алгоритму для зменшення обчислювальних вимог.

Ключові слова: *поля випромінювання, наукові обчислення, диференційований рендеринг, локалізація, реконструкція, комп'ютерний зір.*