DOI: https://doi.org/10.15407/kvt216.02.015

SMIRNOV A.O., PhD Student,
https://orcid.org/0009-0002-6509-4135, e-mail: tonysmn97@gmail.com
International Research and Training Center for Information Technologies
and Systems of the National Academy of Sciences of Ukraine
and the Ministry of Education and Science of Ukraine,
40, Acad. Glushkov av., Kiyv, 03187, Ukraine

CAMERA POSE ESTIMATION USING A 3D GAUSSIAN SPLATTING RADIANCE FIELD

Introduction. *Accurate camera pose estimation is crucial for many applications ranging from robotics to virtual and augmented reality. The process of determining agents pose from a set of observations is called odometry. This work focuses on visual odometry, which utilizes only images from camera as the input data.*

The purpose of the paper *is to demonstrate an approach for small-scale camera pose estimation using 3D Gaussians as the environment representation.*

Methods. *Given the rise of neural volumetric representations for the environment reconstruction, this work relies on Gaussian Splatting algorithm for high-fidelity volumetric representation.*

Results. *For a trained Gaussian Splatting model and the target image, unseen during training, we estimate its camera pose using differentiable rendering and gradient-based optimization methods. Gradients with respect to camera pose are computed directly from image-space per-pixel loss function via backpropagation.*

The choice of Gaussian Splatting as representation is particularly appealing because it allows for end-to-end estimation and removes several stages that are common for more classical algorithms. And differentiable rasterization as the image formation algorithm provides real-time performance which facilitates its use in real-world applications.

Conclusions. *This end-to-end approach greatly simplifies camera pose estimation, avoiding compounding errors that are common for multi-stage algorithms and provides a high-quality camera pose estimation.*

Keywords: *radiance fields, scientific computing, odometry, slam, pose estimation, Gaussian Splatting, differentiable rendering.*

INTRODUCTION

Accurate pose estimation is an important part of many algorithms. In particular interest are the algorithms that operate only on a given set of images as obtaining them is extremely easy and cheap.

The task of estimating changes to the position over time with the use of data from motion sensors is called odometry [1]. And when motion sensor is a camera and data is images, then such task is called visual odometry [2].

There are many approaches to solving this problem [3], from different camera setup (monocular or stereo), using feature-based (tracking a sparse set of key-points) or direct methods (using whole image and processing pixel intensities). However, these approaches operate in 2D image without fully reconstructing the environment requiring several stages from keypoint detection [4], matching, triangulation and only then tracking. Each such stage introduces errors which

lead to a loss of accuracy.

Recent advancements in scene reconstruction [5, 6] and representation such as Neural Radiance Fields perform high-quality scene reconstruction utilizing only a set of images, where reconstruction itself is performed using differentiable rendering methods and gradient-based optimization.

While original Neural Radiance Fields perform ray-tracing [5] to render an image a more computationally efficient Gaussian Splatting algorithm [7] does it via rasterization which is especially important for real-time applications.

This work demonstrates an approach of how to estimate camera pose given a trained Gaussian Splatting model.

To perform a reconstruction using Gaussian Splatting algorithm an initial set of images must have known camera poses. However, once a reconstruction is obtained, camera poses for new images can be recovered by optimizing camera parameters via gradient descent by minimizing a per-pixel loss function of the rendered image and the target image for which we want to find its corresponding camera pose.

OVERVIEW OF A 3D GAUSSIAN SPLATTING ALGORITHM

Gaussian Splatting algorithm [7] represents the environment with a set of anisotropic 3D Gaussians. Each 3D Gaussian is parametrized by its mean μ_i (position), covariance matrix Σ_i and opacity α_i. View-dependent color of each Gaussian is represented via spherical-harmonics.

Rendering procedure follows the same image formation model as with Neural Radiance Fields (NeRF) [5], specifically:

$$C = \sum_{i=1}^{N} T_i \left(1 - exp\left(-\sigma_i \delta_i\right)\right) c_i$$

$$T_i = exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \tag{1}$$

where C is a color of the ray emitted from the pixel. Each point in the space is characterized by its density σ_i and emitted color c_i, which are taken with δ_i intervals along the ray. T_i is a transmittance or equivalently, a differential probability of a ray being absorbed by particles up to i-th step.

INITIALIZATION AND TRAINING

To train Gaussian Splatting model, it is first necessary to initialize it. Given a set of images $I = I_1, \ldots, I_n$ and known camera intrinsic parameters (focal length, principal point, distortion coefficients) an off-the-shelf Structure-from-Motion algorithm is used [8] to estimate camera poses $T = T_1, \ldots, T_n$.

This is typically done by finding keypoints, then matching them across im- ages, obtaining an initial point-cloud with triangulation and then performing global Bundle-Adjustment procedure to optimize both camera poses and point-cloud [8].

After that, Gaussian Splatting model can be initialized with the point-cloud obtained from Structure-from-Motion step.

Training is then done by randomly selecting an image $I_k \in I$, rendering the model as described in (1) to obtain an image $\hat{I}$ computing the loss function

$$\lambda\left(\hat{I} - I_k\right(+ \left(1-\lambda\right) SSIM\left(\hat{I}, I_k\right), \tag{2}$$

where SSIM is a Structural Similarity Index Measure [9] and $\lambda \in [0, 1]$ is a scalar value.

Finally, given a loss function we compute gradients w.r.t. Gaussian Splatting model which are then used to update its parameters. This procedure repeats until convergence.

Given that we compute a gradient, the entire rendering procedure must be differentiable. In contrast to other Neural Radiance Fields which use raymarching for image formation, Gaussian Splatting does it with rastereization. This has several benefits, as rasterization is more efficient and because positions of 3D Gaussians are known there is no need to skip empty spaces.

However, since each 3D Gaussian is parametrized by its density (opacity) σ, in order to compute the final pixel color we need to perform alpha-composing of L ordered 3D Gaussians overlapping that pixel.

To compute ordering efficiently and avoid recomputing it for each pixel, 3D Gaussian Splatting algorithm performs one global sorting of every 3D Gaussian according to the following key [7]:

```
# Depth value of a 3 D Gaussian
depth::Float32 = ...
# Id of the 16 x16 tile that this 3 D Gaussian overlaps.
tile_id::UInt64 = ...
key::UInt64 = (tile_id << 32) | reinterpret(UInt64, depth)
```

3D Gaussian Splatting uses a tiled renderer, where it divides image into $16x16$ tiles and each tile is processes on GPU by a block of 256 threads.

Therefore, tile id is the ID of the 16×16 tile that the current 3D Gaussian overlaps and depth is the distance to the camera. Using this key, gives a list of 3D Gaussians that are first sorted (and thus grouped) by tiles they overlap and within each tile by depth. This way sorting can be done only once globally, afterwards a final color can be computed efficiently by fetching 3D Gaussians from the sorted list.

DENSIFICATION

At the beginning of the optimization gaussians are initialized with the initial point cloud obtained from Structure-from-Motion algorithms (e.g. COLMAP [8]) that was used to compute camera poses. However, the initial set of points may not be enough for high-quality representation of the environment. Therefore, during training the Gaussians may grow in its quantity, either by

– **cloning** existing Gaussian and shifting its position along the positional gra- dient. This may be done in under-reconstructed regions with small fine details.
– or by **splitting** Gaussians if they are too big in size. In this case they are split into two smaller ones with randomly sampled position within the original Gaussian.

This allows 3D Gaussians to represent fine-details structures given enough training time.

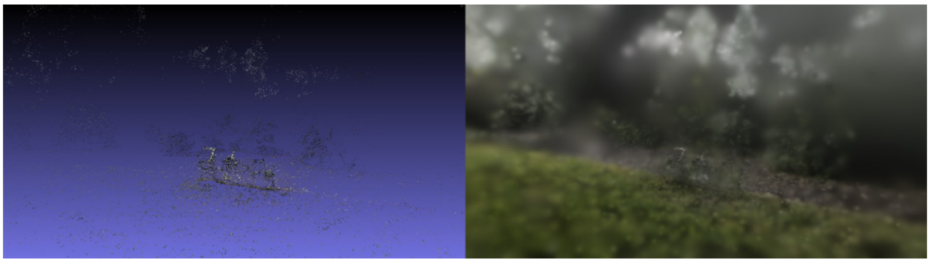

Fig. 1. Point-cloud obtained from Structure-from-Motion (left) and initial Gaussian Splatting model (right).

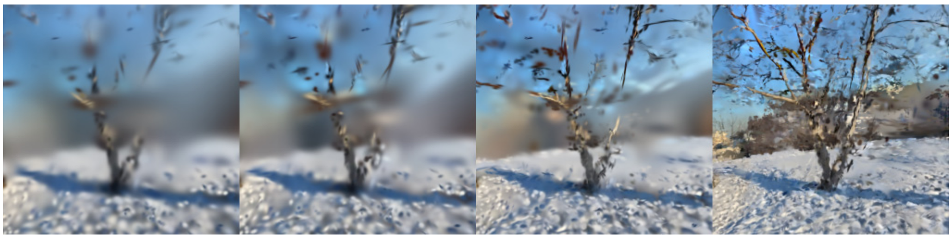

Fig. 2. Example of a several rounds of densification.

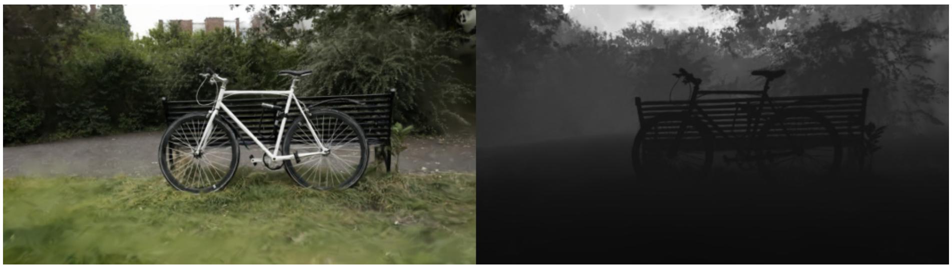

Fig. 3. Example of reconstruction of the bicycle, where densification procedure al lows to represent thin spokes. Left figure is the color rendering mode, right is the depth rendering mode.

In the next section we describe camera pose estimation task and outline the algorithm to efficiently optimize camera poses given trained Gaussian Splatting model.

CAMERA POSE ESTIMATION

As was mentioned above, camera pose estimation is an important part of many algorithms such as Simultaneous Localization and Mapping (SLAM), where the goal is to estimate both camera pose and reconstruct map of the environment. Also, Gaussian Splatting algorithm currently relies on off-the-shelf Structure-from-Motion algorithm to obtain its initialization. Having two sepa rate stages introduces its own challenges such as increasing engineering complex ing by having to integrate two separate algorithms and introducing errors from previous stages which may decrease reconstruction quality.

Therefore, it is tempting to have a single ”stage” [10], where both the environ- ment map and camera poses can be learned using Gaussian Splatting algorithm. Here we consider a simplified scenario, where given a trained Gaussian Splat- ting model from the set of images I and known camera poses T our goal is to estimate camera pose for the new, previously unseen, image $I_{\text{new}} \notin$ I.

However, finding camera pose for the arbitrary image is an ill-posed problem and out of scope of this work, therefore we make following restrictions:

- New image I_{new} must observe the same environment as the other images I.
- The true camera pose T_I new that we want to estimate must be relatively close to other camera poses from T (small-scale constraint).

These restrictions will be explained a greater detail later on, but in short, it is related to the gradient-based optimization procedure. And our goal with this work is to target scenarios similar to SLAM, where new unseen images are close to the previous ones.

As can be seen in Fig. 4, a target camera pose that we want to estimate (blue frustum) located around images that were used during training (green frustum).

To represent a camera pose we use an unnormalized quaternion $q = \{w, x, y, z\}$ for the rotational component and 3D vector $t = \{x, y, z\}$ for the translational component. Usage of quaternions allows for a gradient-based optimizations comparing to 3×3 rotation matrices for which their properties are not guaranteed to hold during optimization.

Another point to note, is that because of gradient-based optimization, co-variance matrix Σ is decomposed into to a scaling matrix S and rotation matrix R, since Σ has to be positive semi-definite, which gradient descent cannot pre- serve. Furthermore, scaling matrix S is represented as a 3-component vector $s = \{s_x, s_y, s_z\}$ and rotation matrix R again as a quaternion $q_R = \{w, x, y, z\}$. Therefore, covariance matrix is computed as [7]:

$$Y = RSS^T R^T . \quad (3)$$

Starting with the initial pose estimation $T_0 = (q_0, t_0)$ at each optimization step we apply the following steps:

1. Before passing 3D Gaussians to the rasterizer, we first have to transform their means μ_k to the camera view given camera pose $T_i = (q_i, t_i)$. To do that we first obtain a normalized quaternion $\hat{q} = \frac{q}{\|q\|2}$, then obtain a 3×3 rotation matrix R from it (Fig. 4).

$$\begin{pmatrix} 1-2\left(y^2+z^2\right) & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2\left(x^2+z^2\right) & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2\left(x^2+y^2\right) \end{pmatrix} \tag{4}$$

and transform the means:

$$\mu = R\mu + t\,. \tag{5}$$

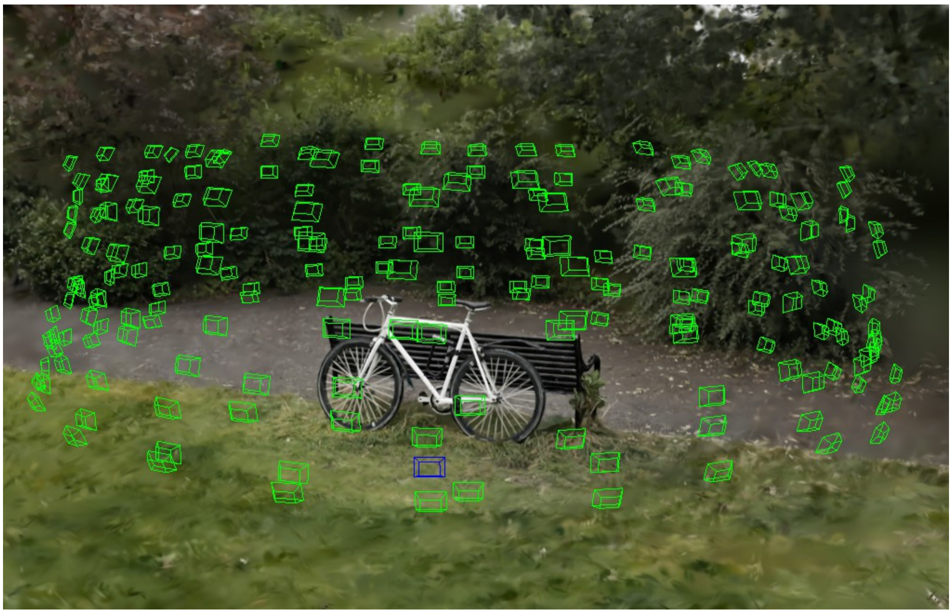

Fig. 4. Visualization of initial camera poses (in green) used during training and a new target pose that we must find (in blue) given its corresponding image.

2. We also need to apply transformation to the rotational component q_R of the covariance matrix Σ. We multiply two quaternions $q_i \cdot q_R$ to obtain rotational component in the camera space:

$$q_i = \{w_1, x_1, y_1, z_1\}$$
$$q_R = \{w_2, x_2, y_2, z_2\}$$

$$\hat{q} = \begin{pmatrix} w_1 * w_2 - x_1 * x_2 - y_1 * y_2 - z_1 * z_2 \\ w_1 * x_2 + x_1 * w_2 + y_1 * z_2 - z_1 * y_2 \\ w_1 * y_2 + y_1 * w_2 + z_1 * x_2 - x_1 * z_2 \\ w_1 * z_2 + z_1 * w_2 + x_1 * y_2 - y_1 * x_2 \end{pmatrix}. \quad (6)$$

3. Even, new Gaussian means $\hat{\mu}$ and $\hat{q}$ we pass them along with other parame ters to the differentiable rasterizer to obtain rendered image $\hat{I}$ and compute loss w.r.t. target image I_i :

$$L = \left\| \hat{I} - I_i \right\|. \quad (7)$$

Here we compute only L_1 loss, without SSIM loss term and compute gradi- ents w.r.t. camera rotation q_i and translation t_i.

4. Finally, we update q_i and t_i using Adam optimizer with learning rates $1e{-}3$ and $1e{-}2$ accordingly.

5. This procedure continues until convergence, which is when the distance be- tween previous and new camera positions is $< 1e{-}2$.

When the optimization procedure finishes, we can move on to the next unseen image. This can be trivially extended to track camera path for a moving camera, not just separate unordered images.

RESULTS

Here we showcase camera pose estimation results, show impact of different ren- dering resolutions on the convergence and display its limitations.

All experiments were done using Adam optimizer with $1e - 2$ learning rate for translational part t_i and $1e - 3$ for the rotational part q_i.

On Fig. 5 we can see examples paths that were taken by the camera during optimization procedure, for better contrast and clarity images were rendered in depth mode. The goal is given starting pose (red frustum) estimate the target pose (visualized with blue frustum) given its corresponding image. Optimization continues until convergence which typically takes around 100–200 optimization steps. Starting pose is obtained from the known camera poses that were used during training of Gaussian Splatting model with some noise ξ added to them.

CONVERGENCE AREA

We also explore how far the starting pose can be from the target pose until convergence fails. To do that, we shift starting pose from the target pose by XY δ values in $[-0.2, 0.2]$ range with 0.05 step. We then run 100 optimization steps for each δ and record if it has converged.

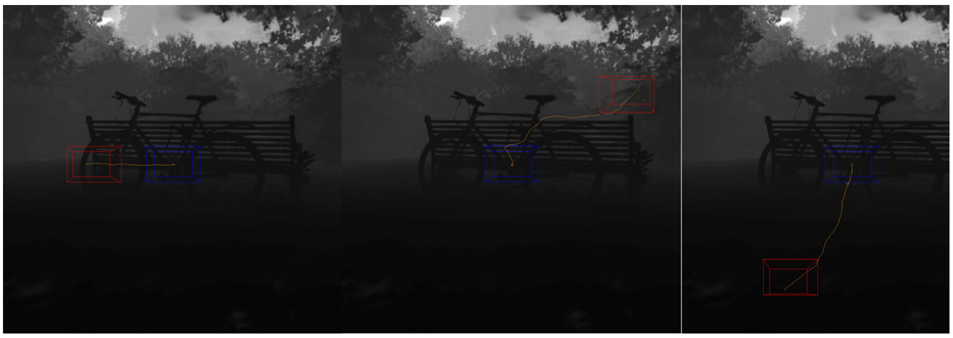

Fig. 5. Examples of camera paths taken during pose optimization. Blue frustum is the target pose, red — starting pose

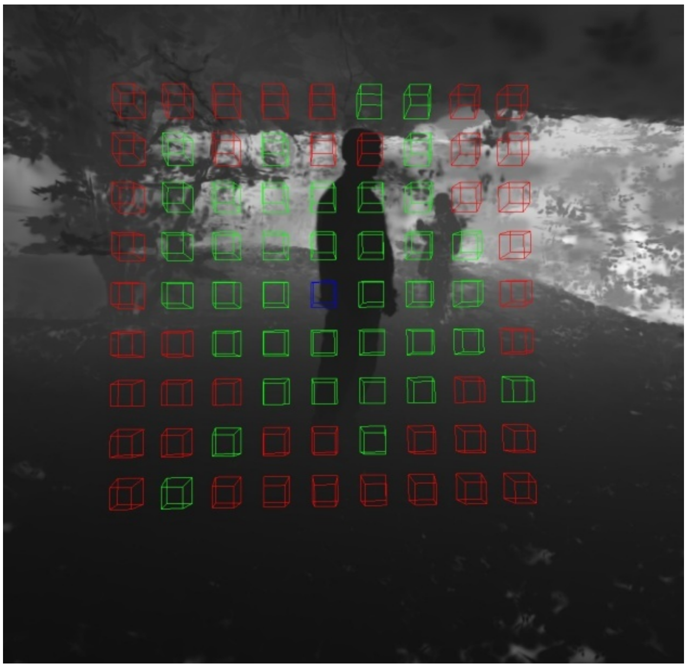

Fig. 6. Convergence area. Blue frustum — target pose, green frustums — starting poses that have managed to converge to the target pose within 100 optimization steps and red frustums — failed to do so

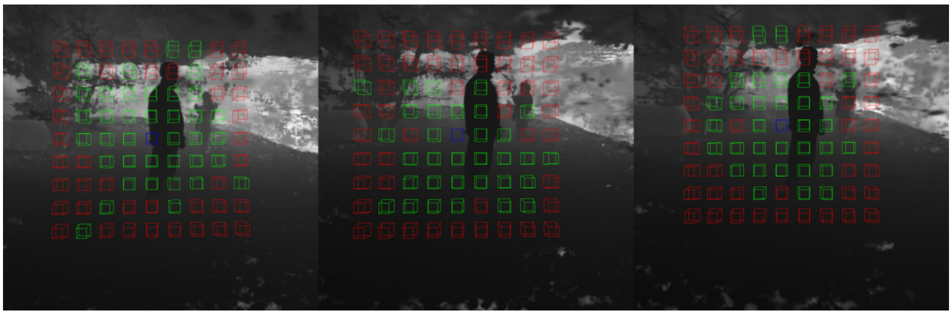

Fig. 7. Difference in the convergence area with changes in the rendering resolution.

On Fig. 6 blue frustum is the target pose, green frustums are the starting poses that successfully converged to the target pose within 100 steps and red ones are those that failed to do so.

We can see that the area within [−0.1, 0.1] range is almost entirely convergable which agrees with the second constraint that we've put on the task, specifically that the distance must be relatively small between the starting pose and the target pose. This approach is suitable for camera pose estimation from dense image captures such as video feed.

Next we observe the impact of resolution on the convergence area. Here the setup is equivalent to the previous one, the only difference is the rendering resolution and the resolution of the target image. We explore three cases: the original image resolution at 1080×1920 pixels and two downsized resolutions, 540×960 pixels (×2 smaller), 270×480 pixels (×4 smaller).

As can be seen in Fig. 7, left image (270×480 pixels) has biggest convergence area, while middle (540×960 pixels) and right (1080×1920 pixels) see small degradation. This is likely because image downsizing acts as a low-pass filter in this case, removing unnecessary information from the image that might prevent convergence.

PERFORMANCE

Rendering resolution and the amount of Gaussians are the two most important factors that impact performance. However, at smaller resolutions this approach can be used even in the real-time setting which is reinforced by the fact that smaller resolutions give better convergence.

Table below gives rendering times for 3 different resolutions, where the scene contains 1 million 3D Gaussians.

Results were obtained with AMD Ryzen 7 5800HS and NVIDIA GeForce RTX 3060 Mobile GPU. Implementation of the Gaussian Splatting algorithm and camera pose estimation is done in the Julia programming language [11] and supports multiple backends such as Nvidia CUDA, AMD ROCm and Apple Metal.

LIMITATIONS

While camera pose estimation work for short distances it is appealing to relax this constraint which presents an interesting research direction.

Additionally, for large scenes with many 3D Gaussians, rendering times become prohibitively slow since testing whether or not a Gaussian is visible involves transforming it to the camera space and checking if it projects on the image plane. To improve this situation, a hierarchical volume structures can be used to efficiently filter out invisible Gaussians. Particular interest presents Octree-GS [12] which provides consistent real-time rendering times with multiple levels of details.

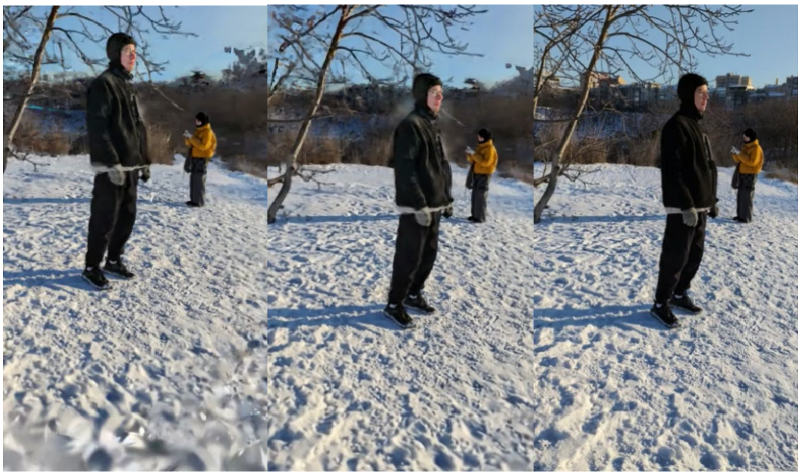

Fig. 8. : Left — image rendered at the starting pose, middle — image rendered at the converged pose, right — target image.

Table 1. Performance comparison of camera pose optimization step

Performance comparison for 1 optimization step			
Resolution	279×480	540×960	1080×1920
Time	30 ms	80 ms	240 ms

CONCLUSIONS

In conclusion, we present an algorithm for small-scale camera pose estimation using 3D Gaussian as an environment representation.

The choice for Gaussian Splatting as the environment representation is particularly appealing, since it allows for an end-to-end pose optimization. While fast rendering times allow for a wide applications from pose recovery given the trained model to real-time Simultaneous Localization and Mapping.

Further research can be conducted to improve convergence properties of the algorithm and to relax the ”small-scale” constraint.

REFERENCES

1. Jeff Bezanson. Julia: A Fast Dynamic Language for Technical Computing, 2012, arXiv: 1209.5145 [cs.PL].
2. Bernhard Kerbl. 3D Gaussian Splatting for Real-Time Radiance Field Rendering, 2023, arXiv: 2308.04079 [cs.GR].
3. Kai Li Lim, Thomas Braunl. A Review of Visual Odometry Methods and Its Applications for Autonomous Driving, 2020, arXiv: 2009. 09193 [cs.CV].
4. Tony Lindeberg. *Scale Invariant Feature Transform*. Vol. 7, May 2012, doi: 10.4249/scholarpedia.10491.
5. Ben Mildenhall. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, 2020, arXiv: 2003.08934 [cs.CV].
6. Thomas Muller. *Instant neural graphics primitives with a multires- olution hash encoding.* ACM Transactions on Graphics 41.4 (July 2022), pp. 1–15. issn: 1557-7368. doi: 10. 1145 / 3528223. 3530127. url: http://dx.doi.org/10.1145/3528223.3530127.
7. Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: IEEE Transactions on

Robotics 31.5 (Oct. 2015), pp. 1147–1163. issn: 1941–0468. doi: 10.1109/ tro.2015.2463671. url: http:// dx.doi.org/10.1109/TRO. 2015.2463671.

8. Jim Nilsson, Tomas Akenine-Moller. Understanding SSIM, 2020, arXiv: 2006.13846 [eess.IV].
9. Shashi Poddar, Rahul Kottath, Vinod Karar. Evolution of Visual Odometry Techniques, 2018, arXiv: 1804.11142 [cs.CV].
10. Kerui Ren. Octree-GS: Towards Consistent Real-time Rendering with LOD-Structured 3D Gaussians, 2024, arXiv: 2403.17898 [cs.CV].
11. Johannes Lutz Schonberger, Jan-Michael Frahm. *Structure-from-Motion Revisited. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
12. Rich Sutton. The Bitter Lesson, 2019. URL: http://www.incompleteideas.net/ IncIdeas/BitterLesson.html.

Received 29.03.2024

Смірнов А.О., аспірант
https://orcid.org/0009-0002-6509-4135, e-mail: tonysmn97@gmail.com

Міжнародний науково-навчальний центр інформаційних технологій
та систем НАН України та МОН України,
просп. Акад. Глушкова, Київ, 40, 03187, Україна

ОЦІНКА ПОЛОЖЕННЯ КАМЕРИ З ВИКОРИСТАННЯМ ПОЛЕЙ ВИПРОМІНЮВАННЯ НА ОСНОВІ 3D ГАУСИАН

Вступ. *Точна оцінка положення камери є ключовим етапом багатьох алгоритмів, які мають широке застосування як в робототехніці так і в віртуальній (VR) та розширеній реальності (AR).*

Процес визначення положення агента з використанням даних спостереження називається одометрією.

Ця робота фокусується на візуальній одометрії, яка використовує тільки вхідні зображення з камери як вхідні дані.

Метою *цієї роботи є демонстрація алгоритму для оцінки положення камери в маленьких масштабах використовуючи 3D Гаусиани для представлення середовища.*

Нейронні об'ємні представлення навколишнього середовища набули широкого застосування як метод для подання середовища.

Методи. *Ця робота спирається на алгоритм Gaussian Splatting для високоякісного об'ємного подання середовища.*

Результати. *Для навченої моделі 3D Гаусиан та цільового зображення, яке не було в наборі даних, який застосовувався для тренування, оцінка положення камери відбувається з використанням методів диференційованого рендерингу та методів градієнтної оптимізації.*

Градієнти відносно параметрів положення камери обчислюються методом зворотнього поширення (backpropagation) з використанням штрафної функції, яка визначена в просторі зображень.

Вибір метода Gaussian Splatting для подання середовища є особливо привабливим, адже він уможливлює наскрізну (end-to-end) оптимізацію положення камери, прибирає декілька етапів, присутніх в більш класичних алгоритмах.

Диференційована растеризація як процес формування зображення забезпечує роботу в режимі реального часу, що сприяє широкому застосуванню цього методу.

Висновки. *Запропонований наскрізний (end-to-end) метод значно спрощує оцінювання положення камери, уникаючи накопичення похибок, наявних в алгоритмах з декількома етапами, та забезпечує високу якість оцінки положення камери.*

Ключові слова: *одометрія, оцінка положення камери, диференційований рендеринг, нейронні поля випромінювання.*