PoP-SLAM: Point cloud Projection for SLAM

Seongmin Jung [1] **Krishi Attri** [2] **Josselin Marchand** [3] **Michele Lorenzo Paolicchi** [4]

Abstract

Dense visual simultaneous localization and mapping (SLAM) systems have significantly advanced with the integration of learning-based scene representations. These representations enhance mapping fidelity and robustness to noise but are computationally expensive, particularly in repetitive rendering processes. Neural point cloud-based systems, despite their adaptability and state-of-the-art performance, suffer from time-intensive nearest-neighbor searches. We propose a novel projection-first rendering strategy that significantly improves processing speed for tracking, enabling real-time performance. Additionally, we introduce a direct occlusion detection mechanism leveraging keyframe depth information for efficient and accurate occlusion handling without requiring volume rendering. Extensive evaluations on both synthetic (Replica) and real-world (TUM-RGBD) datasets demonstrate that PoP-SLAM significantly improves tracking speed—achieving around 4 frames per second—as well as maintaining superior tracking accuracy over Point-SLAM, our main baseline.

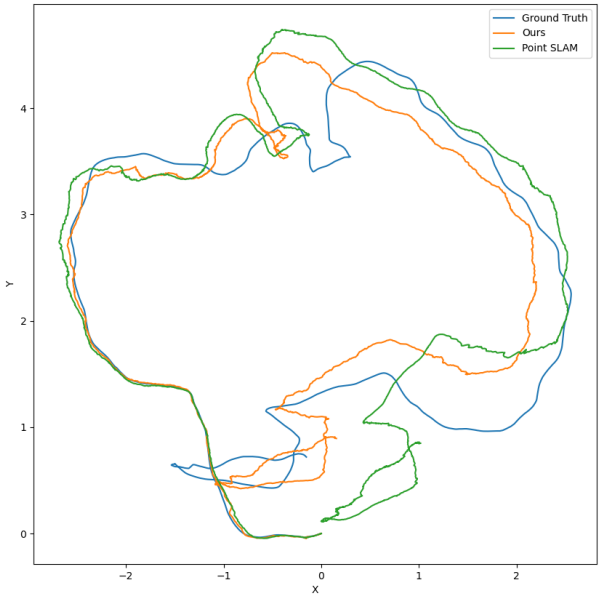

Figure 1. Estimated trajectory of PoP-SLAM and Point-SLAM in the `fr3_office` scene of TUM-RGBD dataset, and the ground truth trajectory are shown. PoP-SLAM gives a qualitatively better estimate of each frame than Point-SLAM.

1. Introduction

Dense visual simultaneous localization and mapping (SLAM) addresses the critical task of enabling an agent, such as a moving robot equipped with a camera, to simultaneously build a map of the surrounding environment while determining its location within that map. It is a well-established yet still rapidly evolving research area within computer vision and robotics. With the advent of neural radiation fields (NeRFs) (15), learning-based scene representations have gained significant attention due to their superior representational power and high rendering fidelity. SLAM systems that use these representations as maps are commonly referred to as *learning-based SLAM systems*. They adopt multi-layer perceptrons (MLPs) (25), feature grids (41), or neural points (21) as their primary map representations. These learning-based scene representations offer several advantages for mapping, such as improved noise and outlier handling (28), as well as improved hole filling (34) and inpainting capabilities for unobserved scene regions (41).

Learning-based scene representations have also significantly transformed tracking methods. Traditionally, tracking was mainly geometric, relying on feature extraction from images, feature matching, and the PnP algorithm (11) to calculate camera poses mathematically. With the advent of novel view synthesis in learning-based scene representations, optimization-based tracking has become the standard for learning-based SLAM systems (41; 21; 33). This ap-

[1]Interdisciplinary Program in Artificial Intelligence, Seoul National University, Seoul, Republic of Korea [2]Department of Mechanical Engineering, Seoul National University, Seoul, Republic of Korea [3]Cursus ISMIN (Ingénieur Systèmes Microélectronique et Informatique), École des Mines de Saint-Étienne, France [4]Department of Mathematics, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands. Correspondence to: Seongmin Jung <sm18570@snu.ac.kr>.

proach involves iteratively rendering color and depth at the estimated camera pose, computing the disparity between the rendered and ground-truth pixel values, and updating the camera pose through backpropagation. However, a fundamental limitation of optimization-based tracking is its high computational cost. Specifically, MLP- and feature grid-based systems require ray casting, and neural point-based approaches depend on nearest-neighbor searches due to the lack of spatial connectivity between points. These rendering processes are the most time-intensive components of the system, often making them unsuitable for real-time applications (4).

In this work, we propose PoP-SLAM, a neural point-based visual SLAM system that addresses these challenges by introducing an efficient projection-first rendering strategy. Building on Point-SLAM (21), one of the first neural point-based SLAM approaches, PoP-SLAM eliminates repetitive nearest-neighbor searches, enabling significant improvements in processing speed. Key contributions of this work include:

- A direct projection-based rendering method that reduces computational time, enabling real-time SLAM performance.
- A direct occlusion detection mechanism leveraging depth inputs from nearby keyframes for efficient occlusion handling without volume rendering.
- Improved tracking accuracy, narrowing the gap between traditional and learning-based SLAM systems.

2. Related Work

2.1. Dense Visual SLAM

Dense 3D visual SLAM has evolved significantly, with the foundational work of (1) introducing truncated signed distance functions (TSDFs). Following this paper are more advanced techniques such as KinectFusion (18). More scalable and memory-efficient techniques, such as voxel hashing (19) and octrees (5) were also developed.

Subsequent advancements addressed robustness and adaptability in SLAM pipelines. BundleFusion (2) handles long-term consistency in mapping. Tackling unreliable depth maps, RoutedFusion (14) introduced a learning-based fusion network for updating TSDFs, while NeuralFusion (12) and DI-Fusion (7) extended this concept by adopting implicit learning for scene representations, enhancing robustness to outliers. More recently, methods have bypassed the need for depth sensors entirely, achieving dense online reconstruction with RGB cameras alone (20).

Dense visual SLAM systems that utilize learning-based representations as maps can be broadly classified into *coupled* (13) and *decoupled* (4) approaches. *Coupled* methods utilize the mapped scene information for tracking, while *decoupled* methods adopt a standalone tracking system to estimate camera poses, subsequently reconstructing the scene based on these poses. Currently, *decoupled* methods have achieved better tracking accuracy and speed, but the tracking system relies on its own scene representations such as traditional point clouds, creating data redundancy. More importantly, their mapping and rendering performance is fundamentally constrained by the accuracy of the tracking system. In contrast, *coupled* methods enable mutual enhancement of tracking and mapping processes, thereby offering greater potential for improving both tracking accuracy and mapping quality.

After learning-based scene representation emerged, it also started to be adopted for SLAM systems. Coupled systems (41; 13; 21; 9; 33) perform tracking by optimizing camera poses by backpropagating errors between observed and rendered pixel colors and depths. Tracking and mapping can benefit from each other in this setting, however, repetitive volume rendering or nearest-neighbor search causes high time consumption.

On the other hand, decoupled systems (38; 4; 27; 31; 6) adopts off-the-shelf tracking pipelines such as (16). The tracking performance is stable, however, it requires independent map representation for the tracking system, and the mapping and rendering performance is fundamentally constrained by the accuracy of the tracking system. The gap in tracking accuracies between coupled and decoupled methods is reducing.

2.2. Learning-based scene representations

Learning-based dense 3D scene representations have been explored through various approaches, generally classified into grid-based, network-based, and point-based methods. Network-based methods use neural networks to provide a continuous representation of a scene through coordinate-MLPs. This method is particularly effective for capturing high-quality textures and complex details in a compact form (25). However, network-based methods lack scalability and make it difficult to update or modify the map once it is trained.

Grid-based representations, the most traditionally studied method, involve structuring the scene in fixed grids, often using dense grids (18), hierarchical octrees (5), or voxel hashing (19) to optimize memory use. These methods support efficient or straightforward neighborhood look-ups and subsequent context aggregations (41). However, since the grid resolution has to be chosen ahead of time in grid-based methods, this results in inefficient memory allocation, wasting resources in empty spaces while failing to capture finer details in complex regions (36).

Neural points (3) suggested a point cloud-based representation for neural fields. This approach avoids the memory inefficiency of grid-based representations by concentrating data storage on object surfaces, with minimal point allocation on free space. Point-NeRF (32) adopted these neural points as a 3D scene representation. However, the adaptability of point-based methods comes at the cost of rendering, due to the lack of an intrinsic connectivity structure among points. Point-SLAM utilizes nearest neighbor search, which is computationally intensive.

More recently, representations based on 3D Gaussian Splatting (3DGS) (10) have gained attention for their faster rendering speeds and explicit nature, while maintaining competitive representational power comparable to Neural Radiance Fields. Explicit parameters such as means and covariances are adjusted using a gradient-based rendering process guided by images captured from multiple viewpoints. However, when applied to SLAM map representations, the operation speed is not as fast as other state-of-the-art methods (4).

Among learning-based SLAM methods, neural point cloud-based systems (37; 13) demonstrate state-of-the-art performance and offer ease of map modification.

2.3. Image rendering

In neural point-based systems, nearest-neighbor search is used to render depth and color from the object's surface. Unlike feature grid-based methods that require ray casting, this approach samples fewer points along each ray, focusing on a range around the sensor's depth. Occupancies and colors for these points are decoded using pretrained Multi-Layer Perceptrons (MLPs) with Gaussian positional encodings (26) to enhance performance. To get a feature vector for each point, geometry, and color features are interpolated using neighboring points weighted by their inverse squared distance. The final depth and color are then determined as weighted averages of the decoded occupancies along each ray.

3. Approach

3.1. Neural Point Cloud

We start by defining a neural point cloud (NPC) similar to (32), namely, as a set of N neural points

$$P = \{(p_i, f_i) \mid i = 1, ..., N\}$$

where $p_i \in \mathbb{R}^3$ denotes the 3D position (x, y, z) in world coordinate system and $f_i \in \mathbb{R}^{32}$ a 32-dimensional color feature vector for the i-th point.

3.1.1. Point Adding Strategy

During each mapping phase, given an estimated camera pose, we employ the following pixel sampling strategy. First, we sample X pixels uniformly across the image plane ensuring some even coverage of the scene. Then, we unproject the pixel coordinates to 3D space using the depth information to obtain their position in the camera coordinate system. Next, for each unprojected 3D point, we perform a nearest neighbor search within a fixed radius r in the existing neural point cloud. This step determines whether the point lies in a region of the scene already represented by existing points in the cloud. If no neighbors are found within the radius, we add a point at the measured depth from camera to the neural point cloud. This strategy ensures the neural point cloud naturally converges to a bounded set of points as no new scene parts are visited. This ensures that the neural point cloud remains compact and efficient. After a point is generated, the color feature is initialized with a normal distribution.

3.2. Projection-First Strategy and Rendering

Given a camera pose T_{cw}, we can render the NPC to get the RGBD pixel value of the specific image frame. We propose a direct projection-based rendering. We denote P_w as a $N \times 3$ matrix which contains N number of 3D coordinates for each point in the world coordinate system. Given a camera pose T_{cw}, we transform P_w into P_c in the camera coordinate system. Subsequently, we project P_c into P_p in the pixel coordinate system using the camera intrinsic matrix $\mathbf{K}$. This projection is done efficiently by vectorized calculation of GPUs.

Now we describe the process of rendering depth and color from a neural point in 3D space. Depth is easily acquired by simply taking the z value of P'_c, expressed as

$$\hat{D} = P_c(z).$$

For color, we directly utilize a point's 3D world coordinates and its feature vector, independent of the camera pose and projection. Specifically, we employ an MLP as a decoder, expressed as

$$\hat{C}_i = g_\xi(p_i, f_i),$$

to predict the color value $C_i \in \mathbb{R}^3$. The color decoder MLP, denoted by g_ξ, is parameterized by the trainable weights ξ. To enhance the network's ability to represent high-frequency details, the 3D point coordinates p_i are first processed through a positional encoding module (26). By assigning these per-point depth and color values to the corresponding pixel coordinates, we complete point-wise rendering.

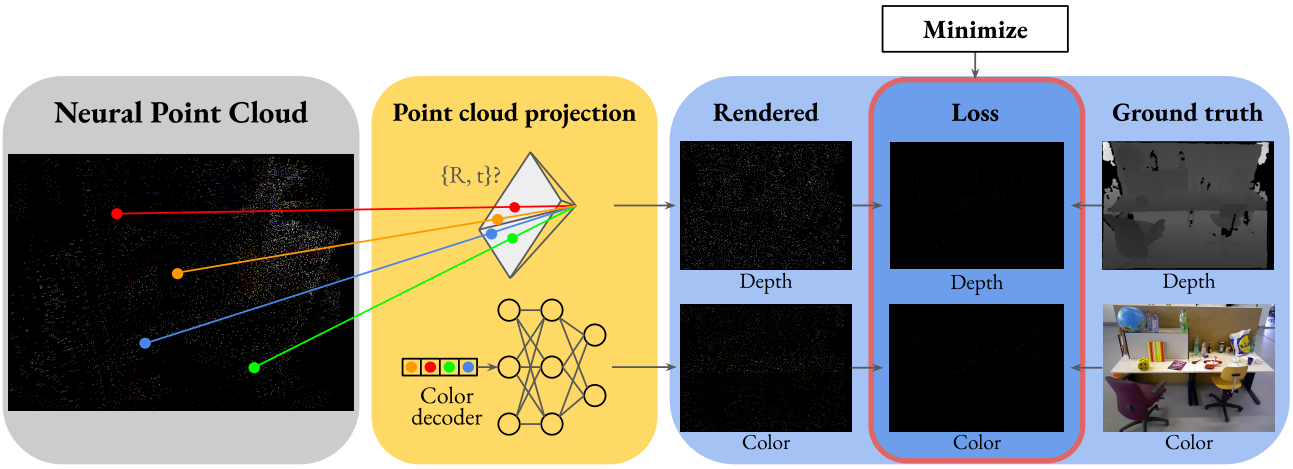

Figure 2. Overview of PoP-SLAM: Illustrates the projection-first rendering pipeline for efficient neural point cloud-based SLAM. The process begins with the neural point cloud, where 3D points are projected into the image plane using the camera pose and intrinsic parameters. The pipeline renders depth and color using point-wise interpolation and a color decoder network. The rendered outputs are compared against ground truth depth and RGB images to compute the loss, which is minimized for accurate tracking. This approach emphasizes computational efficiency of the pipeline.

Algorithm 1 Point-Wise Rendering

Input: point P_w, camera pose T_{cw}, intrinsic matrix $\mathbf{K}$
$P_c = T_{cw} P_w$
$P_p = \mathbf{K} P_c$
for all k **in** $\{1, \ldots, K\}$ **do**
  $P_c^k = T_{cw}^k P_w$
  $P_p^k = \mathbf{K} P_c^k$
  $mask \leftarrow mask \;\&\; (|P_c^k(z) - D^k(P_p^k)| < \delta)$
end for
$P'_w \leftarrow P_w[mask]$
$P'_c \leftarrow P_c[mask]$
$P'_p \leftarrow P_p[mask]$
$\hat{D} = P_c(z)$
$\hat{C} = g_\xi(P'_w)$

3.2.1. Direct Occlusion Detection

In the process, we can naturally filter out points that do not fall into the image frame or are located behind the image frame. To filter out occluded points, we effectively utilize depth information of nearby keyframes at time $t = k \in \{1, \ldots, K\}$, of which we know the pose T_{cw}^k and where K denotes the total number of keyframes. Keyframes that have a significant overlap with the viewing frustum of the current frame are selected. We project P_w to each selected keyframe and choose points that are located near the measured depth of every selected keyframe, to get P'_p and its unprojected versions P'_w and P'_c. The margin δ between the ground truth and estimated depth is a hyperparameter. Algorithm 1 shows the overall tracking pipeline including point projection and occlusion detection.

3.3. Tracking

At time t, the camera pose T_{cw}^t is initialized using the constant velocity model, which predicts the pose incrementally based on the previous two time steps:

$$\Delta T_{cw}^{t-1} = \left(T_{cw}^{t-2}\right)^{-1} T_{cw}^{t-1},$$
$$T_{cw}^{t} = \Delta T_{cw}^{t-1} T_{cw}^{t-1}.$$

This initialization provides a reasonable initial estimate for the current pose based on the motion observed in preceding frames. Once the initial pose T_{cw}^t is initialized, the neural point cloud is rendered into the image frame, producing predicted depth $\hat{D}_i$ and color $\hat{C}_i$ for each valid point, as detailed in Section 3.2. To refine the pose, we iteratively minimize a multi-term loss function:

$$\mathcal{L} = \sum_{i=1}^{M_{tr}} \left(|D_i - \hat{D}_i| + \lambda |C_i - \hat{C}_i|\right),$$

where M_{tr} denotes the number of valid projected points P'_w, and λ is a constant hyperparameter that balances the contributions of depth and color errors. D_i and C_i are the ground truth depth and color values at pixel i, derived from the incoming RGB-D frame. The optimization of T_{cw}^t proceeds through gradient-based methods, leveraging the differentiable nature of the projection and rendering processes.

We employ bilinear interpolation for estimating D_i and C_i values at projected locations because projected points often fall at floating-point pixel coordinates rather than integer

coordinates. To estimate precise values at these positions, bilinear interpolation combines the contributions of the four nearest pixels, weighted by their relative distances to the projected point. This approach prevents rounding errors and ensures accurate depth and color predictions, essential for reliable pose optimization.

3.4. Mapping

During the mapping process, we minimize the mapping loss function to optimize the color feature vectors f_i for the points seen in the current image frame. The mapping loss $\mathcal{L}_{\text{map}}$ is defined as

$$\mathcal{L}_{\text{map}} = \sum_{k=1}^{K} \sum_{i=1}^{M} |C_i^k - \hat{C}_i^k|,$$

where M denotes the number of valid points P'_w, C_i^k denotes the k-th keyframe's ground truth color value for pixel i, and $\hat{C}_i^k$ is its predicted counterpart. The summation is performed over all pixels i in the current frame and over keyframes $k \in \{1, \ldots, K\}$, where K is the total number of keyframes. These are sampled based on their overlap with the current frame's viewing frustum.

In our approach, the optimization process is solely focused on refining the color feature vector, ensuring accurate color representation while we get depth values through direct projections.

Keyframes are incorporated into the optimization process to enhance robustness. These are sampled based on their overlap with the current frame's viewing frustum, and their pixel samples are included to regularize the mapping loss. This streamlined mapping pipeline improves speed and precision while focusing on color feature refinement.

4. Experiments

4.1. Experimental Setups

4.1.1. Implementation Details

Our implementation is designed to efficiently manage neural point cloud updates and leverage our novel projection-first rendering mechanism. Experiments are conducted on a machine with an Intel(R) Core(TM) i7-14700KF CPU and an NVIDIA GeForce RTX 4070 GPU.

We use the following hyperparameter settings for each datasets:

- **Learning rate:** 2×10^{-4}.
- **Depth difference margin** (δ)**:** 0.1.
- **Number of pixels sampled for mapping** (X)**:**
  - **TUM-RGBD:** 10000.
  - **Replica:** 5000.

4.1.2. Evaluation Metrics

We evaluate our method using the following metrics:

- **Tracking Accuracy:** Measured using Absolute Trajectory Error (ATE) RMSE (24) to quantify deviations in estimated camera poses.
- **Tracking Frames Per Second (FPS):** In order to measure computational efficiency of our pipeline.

4.1.3. Datasets

We perform experiments on two benchmark datasets to evaluate the effectiveness of our approach:

- **Replica Dataset** (23): A synthetic dataset featuring photorealistic indoor scenes.
- **TUM-RGBD Dataset** (24): Real-world indoor sequences with ground truth poses captured using an external motion capture system.

4.1.4. Baseline Methods

We compare our approach against several state-of-the-art SLAM systems, focusing on methods that employ learning-based scene representations. Our primary baseline is Point-SLAM (21), a neural point cloud-based SLAM system that uses nearest neighbor search for rendering. We also evaluate against NICE-SLAM (41), which leverages feature grids for dense reconstruction, and GLORIE-SLAM (37), a coupled SLAM system known for its competitive tracking accuracy. Loopy-SLAM (13), designed for loop closure optimization and accurate mapping, is another key baseline.

4.2. Tracking

We evaluate the tracking performance of our method on the TUM-RGBD and Replica datasets. Results are summarized in Tables 2 and 3. On TUM-RGBD, our approach achieves the best tracking accuracy among all evaluated methods, with an average ATE RMSE of 0.75 cm, significantly outperforming prior state-of-the-art methods. This highlights the robustness of our projection-first methodology in real-world environments.

On the Replica dataset, our method demonstrates competitive performance, achieving an average ATE RMSE of 0.71 cm. Although we do not outperform methods that utilize loop closure, such as Loopy-SLAM (13), our results remain consistent and robust across all evaluated scenes.

Our projection-first methodology directly utilizes interpolated depth and color values, enabling precise pose estimation while avoiding the computational overhead of nearest-neighbor search. Depth and color values are efficiently rendered using bilinear interpolation, ensuring smooth and accurate results even in challenging scenarios.

Method	TUM-RGBD (24)						Replica (23)				
	fr1_desk	fr1_desk2	fr1_room	fr2_xyz	fr3_office	AVG	office0	office1	office2	office3	AVG
Point-SLAM (21)	1.31	0.84	1.39	1.66	1.61	1.36	2.81	2.89	2.58	2.49	2.69
PoP-SLAM (Ours)	3.76	3.45	3.53	5.85	4.00	4.12	3.37	4.22	3.42	3.17	3.55

Table 1. **Tracking Frames Per Second (FPS) comparison on TUM-RGBD (24) and Replica (23).** Dataset-specific averages (TUM AVG and Replica AVG) are included in the table. Results are measured on the same PC, with specifications detailed in Section 4.1. Better values are highlighted as best.

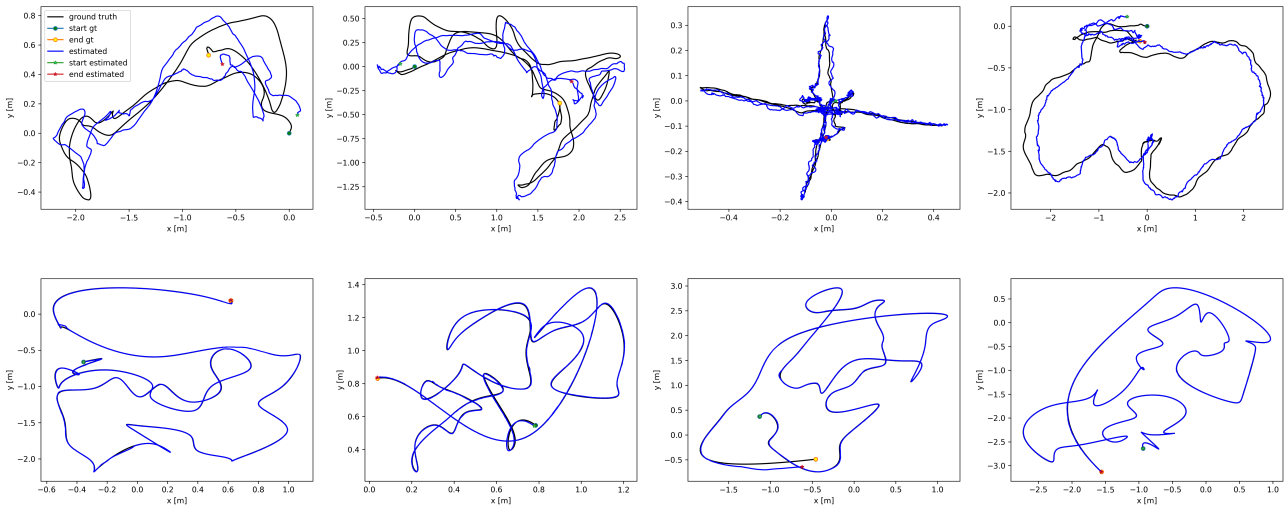

Figure 3. Qualitative evaluation of our system's trajectory estimation across diverse indoor scenes from the TUM-RGBD and Replica datasets. Each subfigure compares the ground truth trajectory (black) against the estimated trajectory from our system (blue), with markers indicating the start (green) and end (orange) points for ground truth, as well as the start (cyan) and end (red) points for the estimated trajectory. The scenes include (top row, left to right): TUM-RGBD fr1-desk, TUM-RGBD fr1-desk2, TUM-RGBD fr1-room, TUM-RGBD fr2-xyz; and (bottom row, left to right): TUM-RGBD fr3-office, Replica office0, Replica office1, and Replica office2. These results demonstrate the effectiveness of our approach in both real-world and synthetic environments, achieving robust tracking performance across varied and challenging conditions.

Method	LC	fr1_desk	fr1_desk2	fr1_room	fr2_xyz	fr3_office	AVG
Neural Implicit Fields							
NICE-SLAM (41)	✗	4.26	4.99	34.49	6.19	3.87	10.76
Point-SLAM (21)	✗	4.34	4.54	30.92	1.31	3.48	8.92
ESLAM (8)	✗	2.47	3.69	29.73	1.11	2.42	7.89
GO-SLAM (38)	✔	1.50	N/A	4.64	0.60	1.30	N/A
Loopy-SLAM (13)	✔	3.79	3.38	7.03	1.62	3.41	3.85
PoP-SLAM (Ours)	✗	2.35	2.23	2.40	1.49	3.41	6.52
3D Gaussian Splatting							
SplaTAM (9)	✗	3.35	6.54	11.13	1.24	5.16	5.48
Gaussian-SLAM (36)	✗	2.73	6.03	14.92	1.39	5.31	6.08
LoopSplat (40)	✔	2.08	3.54	6.24	1.58	3.22	3.33
Classical							
BAD-SLAM (22)	✔	1.7	N/A	N/A	1.1	1.7	N/A
Kintinuous (30)	✔	3.7	7.1	7.5	2.9	3.0	4.84
ORB-SLAM2 (17)	✔	1.6	2.2	4.7	0.4	1.0	1.98
ElasticFusion (29)	✔	2.53	6.83	21.49	1.17	2.52	6.91
BundleFusion (2)	✔	1.6	N/A	N/A	1.1	2.2	N/A

Table 2. **Tracking Performance on TUM-RGBD (24)** (ATE RMSE ↓ [cm]). LC indicates loop closure. The best results are highlighted as first, second, and third. PoP-SLAM performs the best.

Method	LC	Rm 0	Rm 1	Rm 2	Off 0	Off 1	Off 2	Off 3	AVG
Neural Implicit Fields									
NICE-SLAM (41)	✗	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.06
ESLAM (8)	✗	0.71	0.70	0.52	0.57	0.55	0.58	0.72	0.63
Point-SLAM (21)	✗	0.61	0.41	0.37	0.38	0.48	0.54	0.69	0.52
GO-SLAM (38)	✔	0.34	0.29	0.29	0.32	0.30	0.39	0.39	0.35
Loopy-SLAM (13)	✔	0.24	0.24	0.28	0.26	0.40	0.29	0.22	0.29
PoP-SLAM (Ours)	✗	0.19	N/A	0.47	0.53	0.45	0.35	0.30	0.38
3D Gaussian Splatting									
SplaTAM (9)	✗	0.31	0.40	0.29	0.47	0.27	0.29	0.32	0.38
Gaussian-SLAM (36)	✗	0.29	0.29	0.22	0.37	0.23	0.41	0.30	0.31
LoopSplat (40)	✔	0.28	0.22	0.17	0.22	0.16	0.49	0.20	0.26

Table 3. **Tracking Performance on Replica (23)** (ATE RMSE ↓ [cm]). PoP-SLAM outperforms Point-SLAM on average and remains competitive with other methods.

By leveraging efficient rendering and accurate pose estimation, our approach demonstrates strong performance in both synthetic (Replica) and real-world (TUM-RGBD) environments. These results validate the effectiveness of our pipeline in achieving state-of-the-art tracking accuracy.

Loop closure, used by methods like Loopy-SLAM, enhances tracking accuracy by identifying previously visited locations and correcting pose drift. It jointly refines camera poses and observed point locations, reducing local perturbations and cumulative errors, especially in large or repetitive environments. Incorporating loop closure into our system could further improve both tracking accuracy and map quality.

4.3. Time Consumption

Table 1 presents the comparison of Tracking Frames Per Second (FPS) between PoP-SLAM and Point-SLAM, our primary baseline. Our method demonstrates significant improvements in computational speed across all scenes.

The computational efficiency of our method primarily benefits from the vectorized calculations enabled by GPUs. While our system processes approximately 15,000 neural points per frame, fewer than 500 points ($\sim 3.3\%$) are typically pruned. This small ratio does not significantly impact overall time consumption, as matrix multiplication involving $\sim$15,000 rows is efficiently handled by GPUs. Testing Point-SLAM with and without pruning points showed no meaningful statistic in processing time. This is because pruning points can directly affect tracking accuracy, making subsequent frame tracking more unstable and causing variations in processing time.

Thus, the efficiency gains of our method stem from algorithmic optimizations and effective GPU utilization, even when handling significantly larger point sets compared to Point-SLAM.

5. Conclusion

We present PoP-SLAM, a dense visual SLAM system that leverages neural point clouds as the sole scene representation, addressing critical limitations of existing learning-based SLAM systems. By introducing a novel projection-first rendering strategy, PoP-SLAM eliminates the need for computationally expensive ray casting or nearest-neighbor searches, achieving significantly faster processing times. This enables real-time performance while maintaining tracking accuracy. Furthermore, the system incorporates a direct occlusion detection mechanism that effectively leverages keyframe depth information, enabling accurate occlusion handling without relying on volume rendering.

Our extensive evaluations on both synthetic (Replica) and real-world (TUM-RGBD) datasets demonstrate that PoP-SLAM achieves substantial improvements in rendering speed, supporting around 4 frames per second. The direct projection-first strategy not only accelerates the SLAM pipeline but also reduces redundancy in rendering and tracking processes. This achievement underscores its potential as an efficient framework for dense 3D mapping and tracking, with broad applicability in real-world scenarios such as robotics and augmented reality.

We built our system on Point-SLAM and observed improvements in both tracking accuracy and processing time. Our method is versatile and can be seamlessly integrated into state-of-the-art systems like Loopy-SLAM for a performance boost. Its compatibility with existing point cloud and pose optimization frameworks makes it an ideal enhancement, offering improved tracking precision and reduced processing time.

5.1. Limitation and Future Works

While PoP-SLAM demonstrates competitive performance, certain limitations and areas for future exploration remain. First, the projection-first rendering approach is currently applied only to the tracking phase. Extending this strategy to the mapping phase could yield additional performance improvements, particularly in terms of mapping speed. Second, our method does not yet support full-image rendering for 3D reconstructions, which is critical for novel view synthesis and visualization tasks. Future work could explore integrating techniques like masked autoencoders to enable efficient full-image rendering (35; 39). Third, implementing loop closure in our system would significantly enhance tracking accuracy by reducing cumulative drift during extended sequences. Since our system is point cloud-based, loop closure can be easily incorporated using existing point cloud alignment and optimization techniques. This would improve the stability and consistency of both tracking and mapping, particularly in large-scale or repetitive environments. Lastly, as the number of neural points increases, processing time naturally grows. Implementing strategies such as local maps to constrain the number of points being projected at any given time could help maintain real-time performance for larger and more complex scenes. Addressing these limitations will not only enhance the capabilities of PoP-SLAM but also contribute to advancing the broader field of dense visual SLAM systems.

References

[1] CURLESS, B., AND LEVOY, M. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (1996), pp. 303–312.

[2] DAI, A., NIESSNER, M., ZOLLHÖFER, M., IZADI, S., AND THEOBALT, C. BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Re-integration. In *ACM Transactions on Graphics (SIGGRAPH)* (2017), vol. 36, ACM, pp. 76:1–76:18.

[3] FENG, W., LI, J., CAI, H., LUO, X., AND ZHANG, J. Neural Points: Point Cloud Representation with Neural Fields for Arbitrary Upsampling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022).

[4] HA, S., YEON, J., AND YU, H. RGBD GS-ICP SLAM. *arXiv preprint arXiv:2403.12550* (2024).

[5] HORNUNG, A., WURM, K. M., BENNEWITZ, M., STACHNISS, C., AND BURGARD, W. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots 34*, 3 (2013), 189–206.

[6] HUANG, H., LI, L., CHENG, H., AND YEUNG, S.-K. Photo-SLAM: Real-Time Simultaneous Localization and Photorealistic Mapping for Monocular, Stereo, and RGB-D Cameras. *arXiv preprint arXiv:2311.16728* (2023).

[7] HUANG, H., SHI, J., MO, Z., HU, Z., XIA, S., BAO, H., AND CUI, Z. DI-Fusion: Online Implicit 3D Reconstruction with Deep Priors. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2021), IEEE, pp. 8446–8455.

[8] JOHARI, M. M., CARTA, C., AND FLEURET, F. ESLAM: Efficient Dense SLAM System Based on Hybrid Representation of Signed Distance Fields, 2023.

[9] KEETHA, N., KARHADE, J., JATAVALLABHULA, K. M., YANG, G., SCHERER, S., RAMANAN, D., AND LUITEN, J. Splatam: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM. *arXiv preprint arXiv:2312.02126* (2023).

[10] KERBL, B., KOPANAS, G., LEIMKÜHLER, T., AND DRETTAKIS, G. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics 42*, 4 (July 2023).

[11] LEPETIT, V., MORENO-NOGUER, F., AND FUA, P. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision 81* (02 2009).

[12] LI, X., TULSIANI, S., KAR, A., HERTZMANN, A., AND FIDLER, S. NeuralFusion: Learning Depth Fusion from Visible Surfaces. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2019), IEEE, pp. 7055–7065.

[13] LISO, L., SANDSTRÖM, E., YUGAY, V., VAN GOOL, L., AND OSWALD, M. R. Loopy-SLAM: Dense Neural SLAM with Loop Closures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2024), pp. 20363–20373.

[14] LIU, Y., DAI, Y., AND LI, H. RoutedFusion: Learning Real-Time Depth Map Fusion. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2020), Springer, pp. 417–433.

[15] MILDENHALL, B., SRINIVASAN, P. P., TANCIK, M., BARRON, J. T., RAMAMOORTHI, R., AND NG, R. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV* (2020).

[16] MUR-ARTAL, R., MONTIEL, J., AND TARDÓS, J. D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics 31*, 5 (2015), 1147–1163.

[17] MUR-ARTAL, R., AND TARDÓS, J. D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics 33*, 5 (2017), 1255–1262.

[18] NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *IEEE International Symposium on Mixed and Augmented Reality* (2011), pp. 127–136.

[19] NIESSNER, M., ZOLLHÖFER, M., IZADI, S., AND STAMMINGER, M. Real-Time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 169.

[20] PRISACARIU, V. A., KAEHLER, O., MURRAY, D., AND REID, I. MonoFusion: Real-Time 3D Reconstruction of Static Scenes Using a Single Monocular Camera. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2017), IEEE, pp. 2503–2510.

[21] SANDSTRÖM, E., LI, Y., GOOL, L. V., AND OSWALD, M. R. Point-SLAM: Dense Neural Point Cloud-Based SLAM. *arXiv preprint* (2023).

[22] SCHOPS, T., SATTLER, T., AND POLLEFEYS, M. BAD SLAM: Bundle Adjusted Direct RGB-D SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[23] STRAUB, J., PASCHALIDOU, D., VALENTIN, J., GAIDON, A., GEIGER, A., AND DAI, A. Replica: A Dataset of Photorealistic 3D Indoor Scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2019).

[24] STURM, J., ENGELHARD, N., ENDRES, F., BURGARD, W., AND CREMERS, D. Benchmark for RGB-D SLAM Evaluation. *Robotics and Autonomous Systems 62*, 2 (2012), 966–975.

[25] SUCAR, E., LIU, S., ORTIZ, J., AND DAVISON, A. iMAP: Implicit Mapping and Positioning in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)* (2021).

[26] TANCIK, M., SRINIVASAN, P. P., MILDENHALL, B., FRIDOVICH-KEIL, S., RAGHAVAN, N., SINGHAL, U., RAMAMOORTHI, R., BARRON, J. T., AND NG, R. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2020), NIPS '20, Curran Associates Inc.

[27] TEIGEN, A. L., PARK, Y., STAHL, A., AND MESTER, R. RGB-D Mapping and Tracking in a Plenoxel Radiance Field. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2024), IEEE, pp. 3342–3351.

[28] WEDER, S., SCHÖNBERGER, J. L., POLLEFEYS, M., AND OSWALD, M. R. NeuralFusion: Online Depth Fusion in Latent Space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), IEEE, pp. 3162–3172.

[29] WHELAN, T., LEUTENEGGER, S., SALAS-MORENO, R. F., GLOCKER, B., AND DAVISON, A. J. ElasticFusion: Dense SLAM Without A Pose Graph. In *Robotics: Science and Systems* (2015).

[30] WHELAN, T., MCDONALD, J., KAESS, M., FALLON, M., JOHANNSSON, H., AND LEONARD, J. Kintinuous: Spatially Extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras* (2012).

[31] WU, X., LIU, Z., TIAN, Y., LIU, Z., AND CHEN, W. KN-SLAM: Keypoints and Neural Implicit Encoding SLAM. *IEEE Transactions on Instrumentation and Measurement 73* (2024), 1–12.

[32] XU, Q., XU, Z., PHILIP, J., BI, S., SHU, Z., SUNKAVALLI, K., AND NEUMANN, U. Point-NeRF: Point-Based Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 5438–5448.

[33] YAN, C., QU, D., WANG, D., XU, D., WANG, Z., ZHAO, B., AND LI, X. GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting. *arXiv preprint arXiv:2311.11700* (2023).

[34] YANG, X., LI, H., ZHAI, H., MING, Y., LIU, Y., AND ZHANG, G. Vox-Fusion: Dense Tracking and Mapping with Voxel-Based Neural Implicit Representation. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* (2022), IEEE, pp. 499–507.

[35] YIN, M., SUN, L., AND LI, Q. Novel view synthesis on unpaired data by conditional deformable variational auto-encoder. In *European Conference on Computer Vision* (2020), Springer, pp. 87–103.

[36] YUGAY, V., LI, Y., GEVERS, T., AND OSWALD, M. R. Gaussian-SLAM: Photo-Realistic Dense SLAM with Gaussian Splatting, 2024.

[37] ZHANG, G., SANDSTRÖM, E., ZHANG, Y., PATEL, M., GOOL, L. V., AND OSWALD, M. R. GlORIE-SLAM: Globally Optimized RGB-Only Implicit Encoding Point Cloud SLAM, 2024.

[38] ZHANG, Y., TOSI, F., MATTOCCIA, S., AND POGGI, M. GO-SLAM: Global Optimization for Consistent 3D Instant Reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2023), IEEE, pp. 3727–3737.

[39] ZHENG, C., AND VEDALDI, A. Free3d: Consistent novel view synthesis without 3d representation. *arXiv* (2023).

[40] ZHU, L., LI, Y., SANDSTRÖM, E., HUANG, S., SCHINDLER, K., AND ARMENI, I. LoopSplat: Loop Closure by Registering 3D Gaussian Splats, 2024.

[41] ZHU, Z., PENG, S., LARSSON, V., XU, W., BAO, H., CUI, Z., OSWALD, M. R., AND POLLEFEYS, M. NICE-SLAM: Neural Implicit Scalable Encoding for SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), IEEE, pp. 12786–12796.