Improved Real-Time Dense ORB SLAM with GPU Implementation

[1,*]*Tsung-Wun Wang* (王琮文), [1]*Han-Pang Huang* (黃漢邦), [2]*Chiou-Shann Fuh* (傅楸善)

[1]Department of Mechanical Engineering,
[2]Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan,
[*]E-mail: r09522848@ntu.edu.tw hanpang@ntu.edu.tw fuh@csie.ntu.edu.tw

ABSTRACT

Simultaneous Localization And Mapping (SLAM) is critical in robotics, computer vision, and environmental reconstruction. This paper presents an improved GPU-accelerated Red, Green, Blue-Depth (RGB-D) module for real-time SLAM. Our approach utilizes parallel computing on GPUs to enhance depth measurement quality with a bilateral filter and minimize CPU-GPU data transfer overhead. Experimental results on the TUM RGB-D dataset demonstrate accurate camera pose estimation, dense surfel-based mapping, and three outputs: camera pose, sparse map (Atlas), and dense map (mesh). Our GPU implementation enables efficient deployment in resource-constrained environments, offering improved 3D reconstruction and real-time performance for various RGB-D SLAM applications. By leveraging the parallel computing capabilities of GPUs, our system contributes to the advancement of real-time RGB-D SLAM, providing new possibilities for robotics and autonomous systems.

Keywords: *RGB-D SLAM, 3D Reconstruction, GPU Acceleration, ORB-SLAM3, Surfel-Based Mapping*

1. INTRODUCTION

Simultaneous Localization And Mapping (SLAM) is an essential task in robotics, computer vision, and environmental reconstruction. The primary objective of SLAM is to estimate the robot's current position and construct a consistent map of its surrounding environment [1]. To achieve this goal, SLAM algorithms utilize measurements between sequential robot positions (odometry) and landmarks. However, performing SLAM in unknown environments lacking prior information can be challenging.

Various techniques have been developed to utilize cameras, which are widely used sensors in SLAM. These techniques include systems with a single camera [2], two cameras [3], and multiple cameras [4], and they are effective in both indoor and outdoor settings. However, the monocular system has the disadvantage of scale uncertainty [5]. By contrast, stereo-camera systems can solve the scale factor issue by comparing the same scene from two different viewpoints, allowing them to obtain 3D structural information [6]. The RGB-D camera, which directly captures depth information, is mainly suitable for indoor environments due to its sensitivity to external light, as it operates using infrared spectrum light [7].

To address the visual SLAM problem, several approaches have been suggested. The main methods are feature-based [8] and direct [2]. Feature-based methods involve the use of key points, which are stored in local submaps or key-frames, and then a graph optimization approach is applied. On the other hand, direct methods store all image pixels on local maps and use photometric losses to estimate measurement errors. These methods produce a dense or semi-dense model of the environment, making them more computationally intensive than feature-based methods [9]. State-of-the-art methods typically have front-end and back-end modules and are highly modular. Some of these methods can even solve the SLAM problem when dynamic objects are present.

ORB-SLAM3 proposed by Campos provides its RGB-D module that incorporates both color (RGB) and depth (D) information from a camera sensor [10]. Although this open-source library achieves some level of accuracy, sensor measurements still account for the results of localization and mapping. If the system produces poor localization performance, the dense surfel mapping is not as good as expected. Therefore, it is important to pre-process the depth measurement before visual odometry is estimated. ORB-SLAM3 only constructs a sparse map but a 3D dense map is more easily understandable and can be implemented in other applications. For example, dense mapping enables environmental perception for a robot.

In this paper, we propose an improved RGB-D module for SLAM with Graphic Processing Unit (GPU) acceleration to achieve real-time performance. Our system leverages the parallel computing capabilities of GPUs to improve the quality of depth measurement through the application of a bilateral filter on 2D depth images and reduce data transfer overheads between the Central Processing Unit (CPU) and GPU. Our experimental results demonstrate that our proposed system produces a dense surfel-based map that can be easily understood and implemented in other applications.

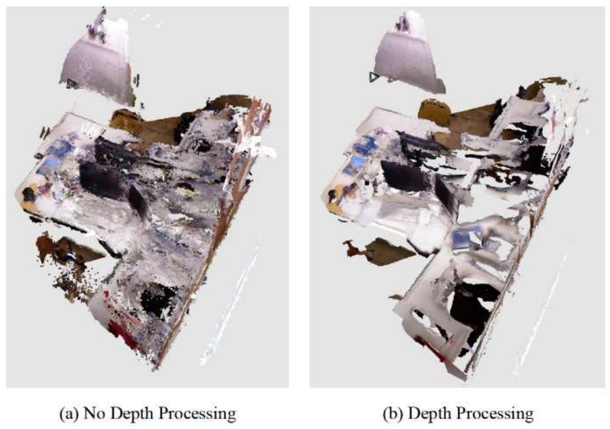

Fig. 1. 3D Reconstruction from TUM RGB-D fr1_desk Dataset during the first 6.07 seconds.

The remainder of this paper is organized as follows: Section 2 describes our proposed system in detail. Section 3 presents our experimental results on the TUM RGB-D dataset and discusses some applications of our project, including relocalization in the sparse map and pathfinding on 3D meshes, as well as challenges we faced such as high GPU utilization. Finally, Section 4 concludes the paper and discusses future work.

Our contributions in this paper are as follows:

(a) We propose an improved RGB-D module for SLAM with GPU acceleration to achieve real-time performance.
(b) We apply a bilateral filter on 2D depth arrays to improve the quality of depth measurement before visual odometry is estimated.
(c) We demonstrate that our proposed system produces a dense surfel-based map that can be easily understood and implemented in other applications.
(d) We provide experimental results showing improved accuracy in 3D reconstruction and camera pose estimation on the TUM RGB-D dataset.

2. THE PROPOSED METHODS

We consider only one input from RGB-D cameras. The depth image will be preprocessed first before entering the ORB-SLAM3 module. ORB-SLAM3 [10] is a visual SLAM that requires both RGB and depth images as inputs. After ORB-SLAM3 calculates the current camera pose, the remaining system will continue the reconstruct the dense 3D map. Our SLAM system generates both sparse and dense map, and the dense maps are constructed in two formats: point-cloud reconstruction and mesh reconstruction. The proposed method is illustrated in Figure 2.

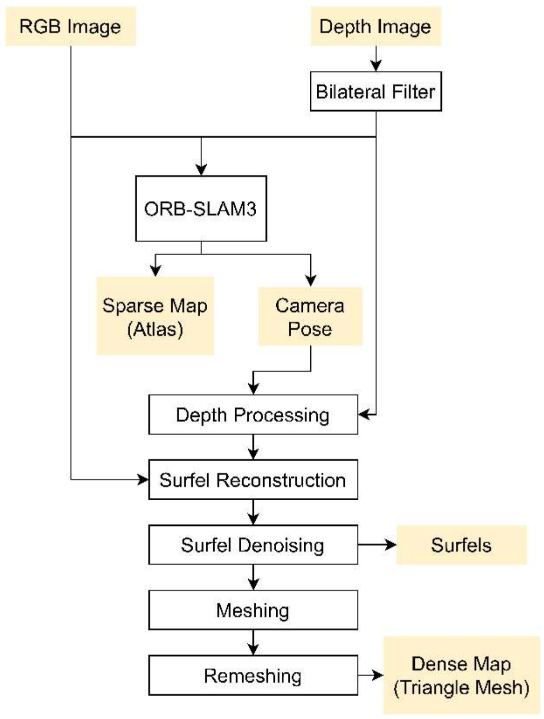

Fig. 2. Our proposed SLAM.

This section presents a novel SLAM algorithm that enhances the ORB-SLAM3 approach through three main improvements. Firstly, depth measurements are processed, and a first-stage processing (bilateral filter) is introduced to enhance the performance of ORB-SLAM3. Secondly, in addition to constructing a sparse map same as in ORB-SLAM3, a dense map is also constructed from surfels. Thirdly, a portion of the computation is offloaded onto the GPU to speed up the SLAM algorithm and achieve real-time performance.

2.1. ORB-SLAM3

ORB-SLAM3 [10] is an open-source library that is widely used for SLAM applications in various domains, including robotics, autonomous driving, and augmented reality. Currently, ORB-SLAM2 [11] and ORB-SLAM3 [10] are the most dependable techniques for trajectory estimation, as evidenced by their consistent performance in diverse environments, sensor setups, and even in the presence of dynamic objects [1]. This can be attributed to their high-quality implementation, use of fast and reliable visual features, and the robustness of their back-end algorithm. ORB-SLAM3 is an upgraded version of ORB-SLAM2 that includes several new features, such as multi-map fusion, improved-recall place recognition, and mid-term and long-term data association. One of its main advantages is its high accuracy in real-time localization and mapping scenarios, coupled with its ability to handle dynamic environments and large-scale scenes effectively. This is made possible by an efficient loop-closure detection method that enhances map accuracy.

However, it should be noted that ORB-SLAM3 may require additional computing resources when combined with other tasks, as it operates at maximum capacity in real-time on a standard computer. Furthermore, the accuracy of the RGB-D module in ORB-SLAM3 is dependent on sensor measurements. Despite these limitations, the affordability and increased popularity of GPUs provide a viable option for accelerating certain tasks.

2.2. Depth Processing

The depth measurement will be processed with GPU implementation similar to Schöps's method [12]. First, pixels with a depth larger than 3m, which are typically noisy, are discarded. A bilateral filter will also be implemented with a depth-dependent parameter σ_z (σ_{xy}= 3 pixels, σ_z = 0.05z). All the above-mentioned depth filtering methods are categorized as the first stage of depth processing, and the result will be downloaded from GPUs and converted into OpenCV MAT format for the input of our SLAM system.

Once the current camera pose has been determined using ORB-SLAM3, the second stage of depth processing can proceed. To eliminate outliers, each pixel is projected onto four previous frames, and only those that project to valid depth measurements within 2% of the projected depth in all frames are kept. Pixels that are within 2px of missing depth measurements are also removed to prevent foreground fattening. Normals are then calculated using finite differences, and pixels whose corresponding normals differ by more than 85 degrees from the direction to the camera are discarded.

2.3. Surfel-based Reconstruction

Our method for reconstructing meshes is similar to the approach used by Schöps [12], and it consists of four main components: surfel reconstruction, surfel denoising, meshing, and remeshing. While the surfel reconstruction and denoising components produce a surfel cloud, the meshing and remeshing components create a triangulation of the surfel cloud. This results in a mesh where the vertices represent the surfels and are updated at the same frame rate, while the topology is determined by the triangulation, which is updated at a slower rate.

2.3.1. Surfel Reconstruction

A surfel, denoted as s, consists of a position p_s, a normal n_s, an RGB color c_s, and a confidence score σ_s, and it belongs to the surfel cloud S. Each surfel is classified as either conflicting with, occluded by, or supported by the depth measurement it projects to. A surfel is considered to be in conflict with the measurement if it projects in front of the measurement uncertainty range of [(1-γ)z, (1+γ)z], where z is the depth and γ is assumed to be 0.05. A surfel is considered to be occluded by the measurement if it does not conflict with it and either projects behind the uncertainty depth range or its normal points away from the camera or differs significantly from the measurement normal. All other surfels are considered to be supported by the measurement.

If the projected surfel is classified as supported by the measurement, the surfel is updated by incorporating the depth value into its estimation of the surface position and normal, using an iterative optimization process that minimizes the distance between the projected surfel and the measurement. This iteration process computes the weighted average. For example, the iterative value of the position of a surfel s, denoted as p_s, is estimated by

$$p_s := \frac{\sigma_s p_s + w p_m}{\sigma_s + w}$$

where the weight w is the reciprocal of the set of all surfels supported by the measurement, σ_s is the confidence score of s, and p_m is the measured value of position of s. Since the measured values of position p_m, normal n_m, and color c_m are given, a supported surfel can be reconstructed by iteratively calculating the weighted values of position p_s, normal n_s, and color c_s.

2.3.2. Surfel Denoising

To avoid improper triangulation because of noisy surfels, regulation is introduced. Now for each surfel s, the indices N_s of four neighboring surfels and the position p_s calculated in Section 2.3.1 are considered, and the objective is to find the denoised position $\overline{p_s}$. After each iteration of surfel reconstruction, we run one iteration of gradient descent to minimize the cost C

To avoid improper triangulation because of noisy surfels, regulation is applied. For each surfel s, the indices N_s of four neighboring surfels and the position p_s calculated in Section 2.3.1 are taken into account, and the goal is to determine the denoised position $\overline{p_s}$. After each iteration of surfel reconstruction, one iteration of gradient descent is performed to minimize the cost C

$$C(S) = \sum_{s \in S} \|\overline{p_s} - p_s\|_2^2 + \frac{\alpha}{|N_s|} \sum_{n \in N_s} (n_s{}^T(\overline{p_s} - p_s))^2$$

where the smoothness weigh α is empirically set to 10.

In addition to regulation, a blending process is used to address surface discontinuities at the boundaries between observed and currently unobserved parts of the scene. To blend the boundary, Schöps et al. [12] proposed an algorithm, described in Alg. 1, that uses a method based on the depth image to detect observation boundaries by identifying pixels that have both a depth measurement and an associated supported surfel, as well as at least one neighboring pixel that lacks at least one of these properties. These pixels are chosen as seeds for the algorithm. The depth difference between the measurement and supported surfel at these pixels is then propagated to suitable neighboring pixels, which are located on the side of the boundary that needs to be corrected.

After each iteration, any pixel that receives a propagated depth difference is updated. A simulated change is made to the pixel's depth to reduce the depth difference, and the effect of this change decreases linearly as the distance from the observation boundary increases. This creates smooth transitions in depth that prevent the creation of discontinuous artifacts.

Algorithm 1 Observation boundary blending

```
1:  # $D(p)$ : depth of pixel $p$; is set to 0 if there is no depth
    measurement there
2:  # $\mathrm{SD}(p)$ : average supported surfel depth of pixel $p$; is
    set to 0 if there is no supported surfel at this pixel
3:  # $I_d, I_s, \delta_d, \delta_s$ : temporary image-sized buffers
4:  # $i_{\text{count}} = 10$ : iteration count
5:  procedure BLEND
6:    # Initialize $I_d$, $I_s$ to -1
7:    for all pixels $p$ do $I_d(p) := -1$; $I_s(p) := -1$
8:    # For all pixels with depth and surfel(s) ...
9:    for all pixels $p$ with $D(p) \neq 0$ and $\mathrm{SD}(p) \neq 0$ do
10:     d := $\mathrm{SD}(p) - D(p)$   # Surfel-vs-measurement delta
11:     for all pixels $q$ in $p$'s 8-neighborhood do
12:       # At boundary of measurement area?
13:       if $I_d(p) = -1$ and $D(q) = 0$ then
14:         # Start propagation (1st case)
15:         $\delta_d(p)$ := d; $I_d(p) := 0$; $D(p) := \mathrm{SD}(p)$
16:       # At boundary of surfel area?
17:       if $I_s(p) = -1$ and $\mathrm{SD}(q) = 0$ then
18:         # Start propagation (2nd case)
19:         $\delta_s(p)$ := d; $I_s(p) := 0$
20:   for $i \in [1, i_{\text{count}} - 1]$ do # Perform blending iterations
21:     # For all pixels with depth ...
22:     for all pixels $p$ with $D(p) \neq 0$ do
23:       # Propagate among pixels with surfel(s)?
24:       if $\mathrm{SD}(p) \neq 0$ and $I_d(p) = -1$ then
25:         UPDATE($i, p, \delta_d, I_d$)            # Update (1st case)
26:       # Propagate among pixels without surfels?
27:       if $\mathrm{SD}(p) = 0$ and $I_s(p) = -1$ then
28:         UPDATE($i, p, \delta_s, I_s$)            # Update (2nd case)
29: function UPDATE($i, p, \delta, I$)
30:   # Average deltas of previous iteration ...
31:   sum := 0; count := 0
32:   for all pixels $q$ in $p$'s 8-neighborhood do
33:     if $I(q) = i - 1$ then sum += $\delta(q)$; count += 1
34:   # Apply the (weighted) averaged delta?
35:   if count > 0 then
36:     $I(p) := i$; $\delta(p) := \frac{\text{sum}}{\text{count}}$; $D(p)$ += $(1 - \frac{i}{i_{\text{count}}})\frac{\text{sum}}{\text{count}}$
```

2.3.3. Meshing

The meshing algorithm works by triangulating the surfels in their local neighborhoods to gradually expand the mesh, with each surfel transitioning between the states of free, front, or completed. A surfel is considered free if it has no triangles connected to it, and all new surfels are initially designated as free. A surfel is considered to be in a front state if it is on the current boundary of the mesh, and the algorithm iterates over all new surfels and all free and front surfels that have moved since the last triangulation iteration.

The search radius of each surfel *s* is defined to find all neighboring surfels as candidates for triangulation. If s is on the mesh boundary and its boundary neighbors are further away than the search radius, the search radius is extended to include those neighbors, up to twice the search radius. If this limit is exceeded, triangulation for this surfel is aborted.

All neighboring surfels are projected onto the tangent plane defined by the normal of the surfel. Neighboring surfels that are not visible from *s* in this 2D projection or whose normal differs significantly from *s* are discarded. If *s* is in a free state, an initial triangle is created with s as one of its vertices [14]. The remaining visible neighbors are sorted based on their angle to s in the 2D projection. While the spaces between adjacent neighbors are classified as either "gap" or "narrow" based on whether the angle between them is considered too large or too small for triangulation, respectively, neighbors that would form "narrow" triangles are discarded to avoid degenerate triangles [22]. Finally, new triangles are added to fill the space between *s* and the remaining neighbors, excluding gaps. After a meshing iteration is completed, the triangle indices of the mesh are updated to reflect the new triangulation.

2.3.4. Remeshing

In addition to the meshing algorithm discussed in Section 2.3.3, a remeshing approach is proposed to efficiently update outdated parts of a mesh due to the movement of existing surfels or the creation of new ones. Invalid triangles, which cannot be generated by the current surfel cloud, are identified and locally replaced using the meshing algorithm. Valid triangles are those that meet certain criteria derived from the algorithm presented in Section 2.3.3. Specifically, (1) a valid triangle must have at least one vertex surfel that includes all triangle vertices within its neighbor search radius and has a similar normal, (2) the normal of the triangle must be within 90 degrees of the normal of the surfel, and (3) there should be no other neighboring surfel that projects into the triangle or intersects with another triangle in the tangent plane of the surfel. In the remeshing algorithm, invalid triangles are deleted and all other triangles connected to surfels within the neighbor search radii of their corner surfels are also removed, and these affected surfels are then scheduled to fill the holes created by the remeshing step. The remaining mesh can be used as the initial state for meshing and new triangles are added at mesh boundaries such as holes.

2.4. GPU Implementation

In order to accomplish all tasks and attain immediate execution, certain operations are relocated to graphics processing units (GPUs). Within our SLAM system, depth processing, such as bilateral filtering, as well as surfel reconstruction and surfel denoising, will be executed on GPU resources to sustain real-time operations. However, the primary aspect of ORB-SLAM3 will be carried out on central processing units (CPUs). Therefore, the data transfer between GPUs and CPUs should be designed to enable both libraries OpenCV and libvis to process and visualize the images on CPUs, respectively. Additionally, the meshing and remeshing procedures will be executed in an asynchronous manner on CPUs.

3. EXPERIMENTAL RESULTS

This section evaluates our proposed SLAM method. Two sets of experiments are conducted: one on TUM datasets and the other in a real environment. All experiments were conducted on a personal computer with an Intel i7 processor, a 16 GB RAM, and an Nvidia GPU RTX3050.

3.1. Evaluation on TUM RGB-D dataset

In order to conduct a further evaluation of our system, the TUM RGB-D dataset [15] was utilized as a benchmark for assessing the accuracy of visual SLAM systems. The TUM benchmark provides multiple real-world scenarios and corresponding ground truth obtained by an exterior motion capture system. Each sequence consists of RGB and depth images (640x480) collected by a Kinect RGB-D camera at a frame rate of 30 Hz and ground-truth camera poses. We run our proposed SLAM method on the TUM RGB-D datasets to estimate the camera poses while constructing sparse and dense maps. For example, the process of running on ‘fr1_desk2’ can be visualized at https://youtu.be/EfcWLF9g2Rc.

To evaluate the accuracy of our SLAM system, we use an open-source measurement tool called evo , which is available at github.com/MichaelGrupp/evo and accessed on 12 May 2023. We use the Absolute Pose Error (APE) metric to assess the overall difference between the algorithm-generated trajectory and the true trajectory. Specifically, we first run our SLAM algorithm and then evaluate the output camera trajectory using the evo tool, as shown in Figure 3. We use two evaluation indices: mean and Root Mean Square Error (RMSE). We repeat this evaluation procedure ten times and calculate the average of these two indices.

Table 1 presents a comparison of the average evaluation results between ORB-SLAM3 and our proposed method when applied to the TUM RGB-D datasets. Our proposed method demonstrates superior performance over ORB-SLAM3 by implementing a depth information filtering technique. Specifically, in the 'fr1_desk2' sequence, our proposed method exhibits an improvement of approximately 3.70% in terms of the APE RMSE value compared to ORB-SLAM3. To further evaluate our method, we employed the 'fr2_large_loop' dataset, which emphasizes the significance of loop correction in a large-scale environment. In this context, the accumulated error becomes prominent. As shown in Table 1, our proposed SLAM method achieves a notable 39.0% reduction in APE RMSE compared to ORB-SLAM3. It should be noted that while ORB-SLAM3 incorporates loop correction through pose-graph optimization, it still encounters challenges in loop detection, resulting in five failed tests and higher APE RMSE values exceeding 0.3. Conversely, as depicted in Figure 9, our proposed SLAM method effectively detects loops, leading to a significant reduction in accumulated error. When both ORB-SLAM3 and our proposed method successfully detect a loop, our approach yields a 1.0% improvement in APE RMSE. Additionally, during the 'fr1_floor' sequence, both SLAM systems eventually lose tracking. However, our proposed method demonstrates the ability to extend the tracking status, thereby contributing to the improvement of RSME in this dataset, primarily through mitigating tracking loss.

3.2. 3D Reconstruction

In this section, we present the results of our surfel-based mapping approach. We compare the sparse map constructed by ORB-SLAM3 with the dense mesh map reconstructed by our proposed method. Figure 4 shows an example of a sparse map from ORB-SLAM3, while Figure 5 shows the corresponding dense mesh map from our surfel-based mapping approach.

As can be seen from these figures, our surfel-based mapping approach produces a dense and detailed representation of the environment that can be easily understood and implemented in other applications. In contrast, the sparse map from ORB-SLAM3 provides only a rough approximation of the environment and lacks the level of detail provided by our dense mesh map.

Table 1. Accuracy of APE on TUM dataset.

TUM Sequence	ORB-SLAM3		Ours	
	Mean ↓	RMSE ↓	Mean ↓	RMSE ↓
fr1_desk2	0.022605	0.025841	**0.021730**	**0.024884**
fr1_rpy	0.017606	0.021268	**0.017560**	**0.021154**
fr2_xyz	0.014082	0.016157	**0.013793**	**0.015585**
fr1_floor	0.561724	0.692359	**0.501201**	**0.590607**
fr2_large_loop	0.250031	0.269637	**0.147816**	**0.164603**

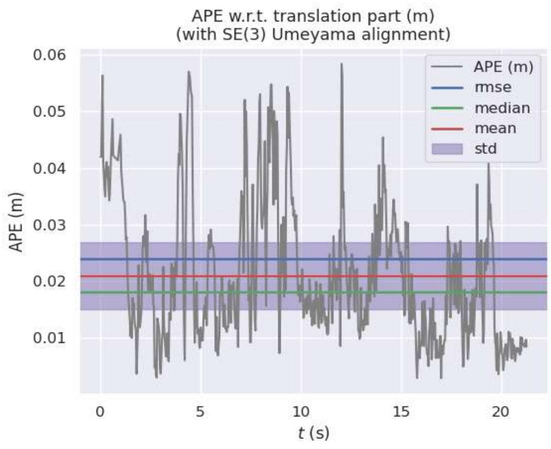

Fig. 3. APE of our proposed method from fr1_desk2.

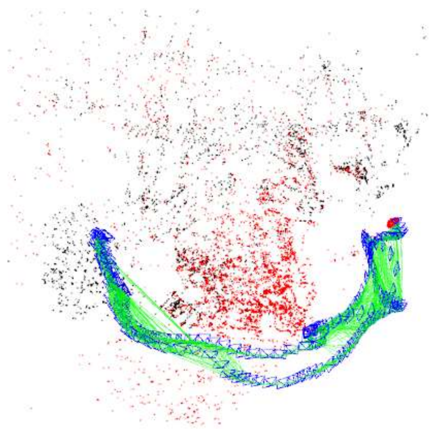

Fig. 4. The sparse map (Atlas) of TUM fr1_desk2 dataset.

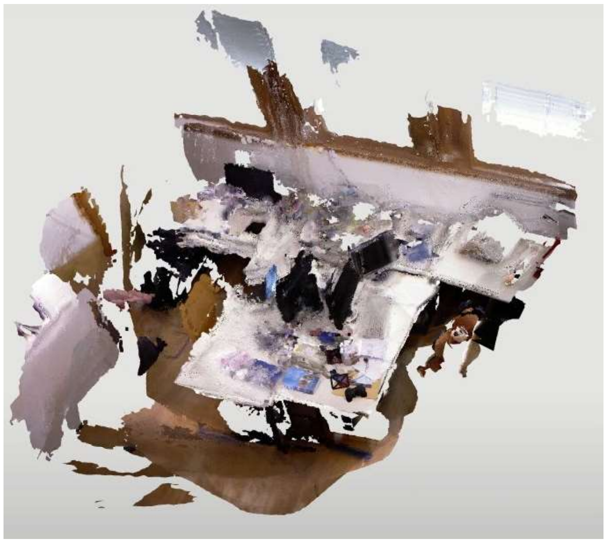
Fig. 5. The surfel-based map of TUM fr1_desk2 dataset.

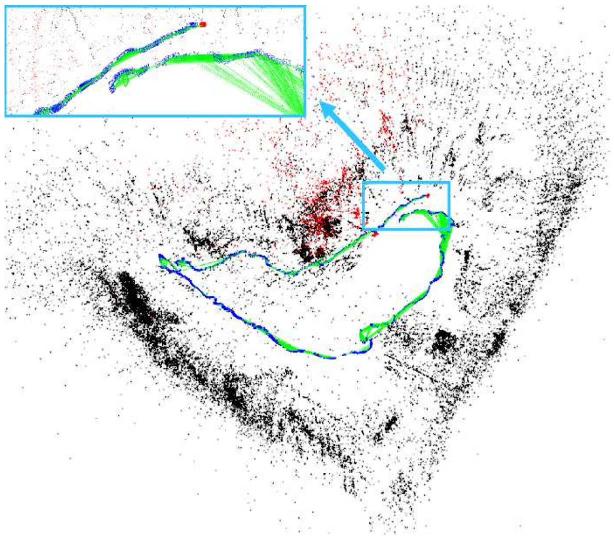
Fig. 8. No loop is detected in the sparse map (Atlas) constructed by ORB-SLAM3 when running TUM fr2_large_loop dataset.

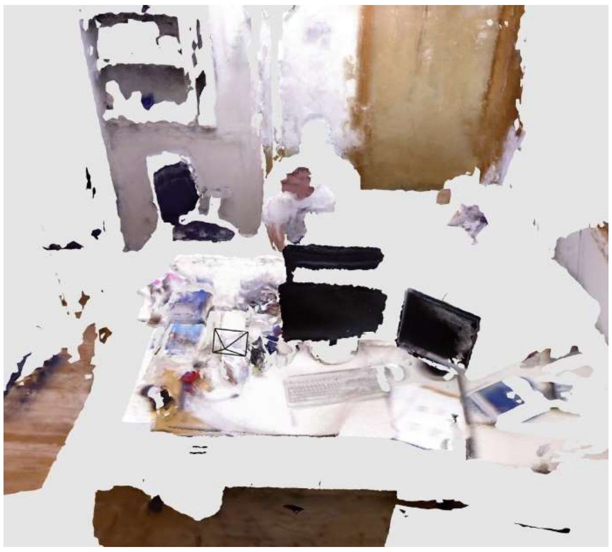
Fig. 6. The surfel-based map of TUM fr1_rpy dataset.

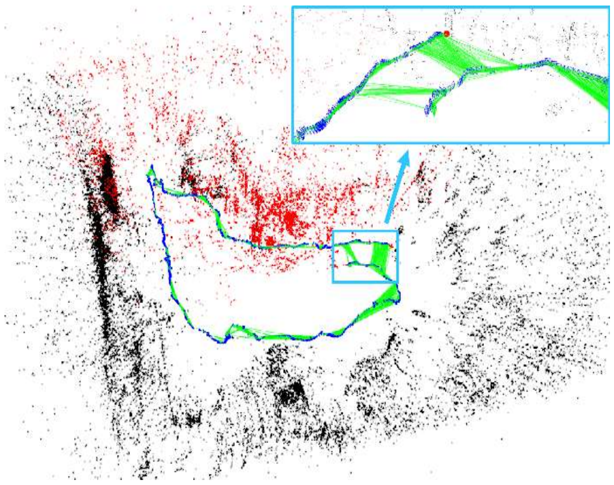
Fig. 9. The sparse map (Atlas) constructed by our proposed SLAM shows loop closing when running TUM fr2_large_loop dataset.

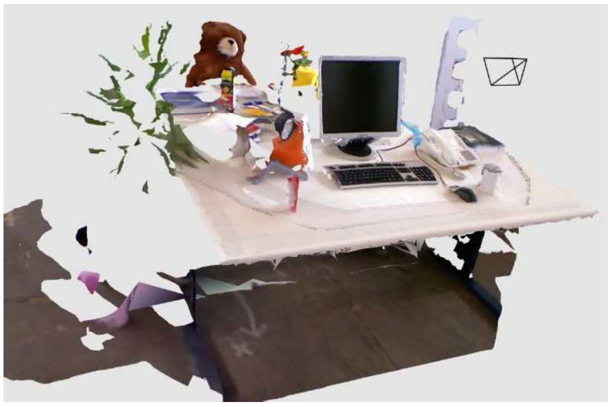
Fig. 7. The surfel-based map of TUM fr2_xyz dataset.

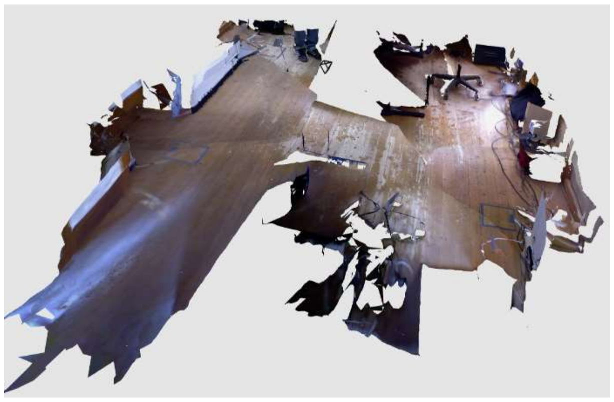
Fig. 10. The surfel-based map of TUM fr1_floor dataset.

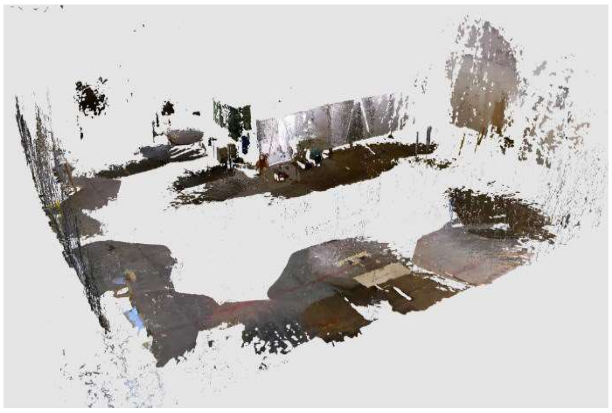
Fig. 11. The surfel-based map of TUM fr2_large_loop dataset.

3.3. Robot SLAM

In our experiment, we employed an Intel RealSense Depth Camera D435i, securely affixed to a wheeled robot, to conduct our investigation. By harnessing the camera's RGB-D capabilities, we successfully reconstructed an indoor environment utilizing our surfel-based mapping method. This approach facilitated the simultaneous capture of color and depth information, allowing for a comprehensive and meticulous representation of the environment. To illustrate the efficacy of our method, Figure 12 showcases a striking example of the 3D reconstruction achieved within a corridor of a building, with adjoining rooms perceptible on the right side.

3.4. Applications

In practical robot applications, relocalization and pathfinding are essential tasks. Firstly, we applied our proposed SLAM method to relocalization using the sparse map (Atlas) constructed by ORB-SLAM3. The Atlas map, which can be saved and reused, allows the robot to quickly determine its position within a previously mapped environment. This capability is crucial for tasks requiring the robot to resume operation seamlessly, leading to enhanced real-time performance.

Additionally, we explored the utilization of our proposed SLAM method for pathfinding on meshes, specifically navigation meshes (NavMeshes), which are widely used in state-of-the-art pathfinding research. By implementing the A* pathfinding algorithm on the NavMesh constructed by our SLAM method, we enabled efficient navigation within the mapped environment. Figure 12 showcases an example of the shortest path between a starting point and destination on the NavMesh, leveraging an indoor environmental model in mesh format. This capability empowers the robot to plan and follow optimal paths between different locations within the environment, further enhancing its autonomy and utility.

3.5. Discussions

One of the challenges we faced in our project is high GPU usage. Due to the large amount of data processing required for dense SLAM and mesh reconstruction, our system places a high demand on GPU resources. This can limit the size of the environmental model that can be built for a long journey of a robot. To address this challenge, we explore several potential solutions, including optimizing GPU usage through careful scheduling and pipelining of CPU and GPU work, reducing the size of the environmental model through downsampling or filtering, using more efficient algorithms for dense SLAM and mesh reconstruction, and upgrading our hardware to a more powerful GPU with more memory and processing power.

In addition, it is important to acknowledge that certain sequences within the TUM dataset feature dynamic objects, which introduce motion and alterations in the scene. If a SLAM algorithm fails to account for these dynamic objects, it can lead to an increase in drift, pose errors, and inaccurate mapping. A specific instance illustrating this issue is observed in the 'fr3_sitting_static' sequence, as depicted in Figure 13 (a) and (b). Although the individuals in the scene are seated, the slight movement of their hands results in the presence of multiple arm representations within the 3D reconstruction. As part of our future work, we plan to incorporate a meshing process that exclusively considers static objects, thereby excluding dynamic objects from the final results of the 3D reconstruction. Our ultimate objective is to remove the influence of dynamic objects in the mapping process.

4. CONCLUSION

In this paper, we presented an improved RGB-D module for ORB-SLAM3 with GPU acceleration to enhance the quality of depth measurement. Our experimental results demonstrate that our proposed system produces a dense surfel-based map that scales well with GPU-accelerated depth processing while maintaining comparable or improved accuracy in 3D reconstruction and camera pose estimation. By leveraging the parallel computing capabilities of GPUs and implementing strategies for data transfer and memory management, we were able to minimize the overhead of data movement between the CPU and GPU, making our method more efficient for real-time RGB-D SLAM in resource-constrained environments.

Our contributions in this paper include improving the quality of depth measurement through the application of a bilateral filter on 2D depth images and enabling dense SLAM on GPU-enabled systems. While ORB-SLAM3 provides a good level of accuracy, it still relies heavily on sensor measurements, and pre-processing of depth measurement is important for accurate localization and mapping. In addition, ORB-SLAM3 only constructs a sparse map, but a 3D dense map is more easily understandable and can be implemented in other applications such as environmental perception for a robot. Our proposed method addresses these limitations and opens up new possibilities for real-time RGB-D SLAM in resource-constrained environments such as robots and autonomous vehicles.

Overall, our GPU implementation of dense SLAM provides a significant improvement in efficiency and accuracy in 3D reconstruction and camera pose estimation. Future work includes extending our proposed method to handle dynamic objects and incorporating other sensors for even more accurate and robust mapping and localization.

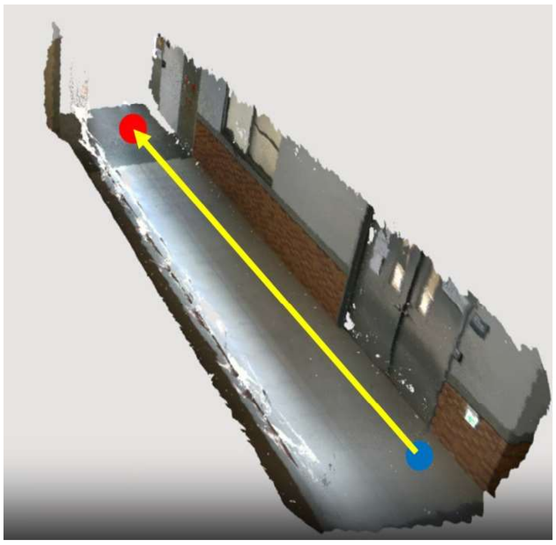
Fig. 12. 3D Pathfinding on Meshes.

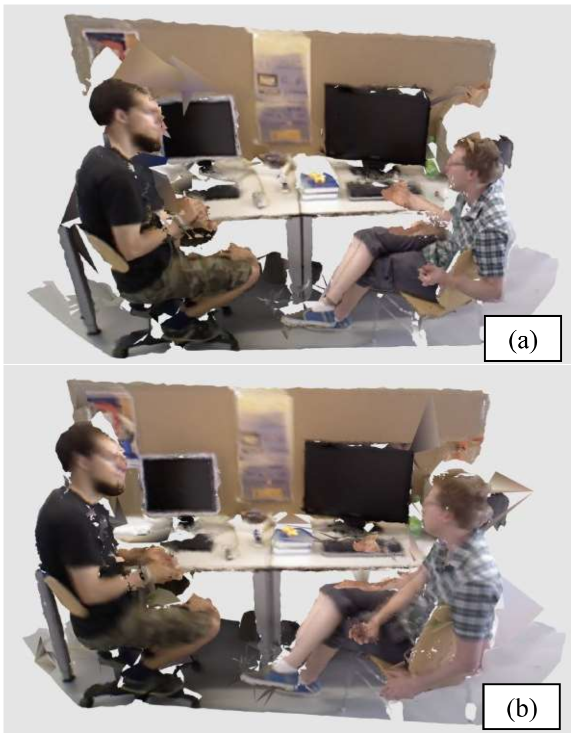

Fig. 13. The surfel-based map of TUM fr3_sitting_static dataset at different time stamps.

REFERENCES

[1] Sharafutdinov, Dinar, et al. "Comparison of modern open-source visual SLAM approaches." Journal of Intelligent & Robotic Systems 107.3 (2023): 43.
[2] Engel, Jakob, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM." Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13. Springer International Publishing, 2014.
[3] Engel, Jakob, Jörg Stückler, and Daniel Cremers. "Large-scale direct SLAM with stereo cameras." 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2015.
[4] Harmat, Adam, Inna Sharf, and Michael Trentini. "Parallel tracking and mapping with multiple cameras on an unmanned aerial vehicle." Intelligent Robotics and Applications: 5th International Conference, ICIRA 2012, Montreal, QC, Canada, October 3-5, 2012, Proceedings, Part I 5. Springer Berlin Heidelberg, 2012.
[5] Kitt, Bernd Manfred, et al. "Monocular visual odometry using a planar road model to solve scale ambiguity." (2011).
[6] Zekavat, Reza, and R. Michael Buehrer. Handbook of position location: Theory, practice and advances. Vol. 27. John Wiley & Sons, 2011.
[7] Chong, T. J., et al. "Sensor technologies and simultaneous localization and mapping (SLAM)." Procedia Computer Science 76 (2015): 174-179.
[8] Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." IEEE transactions on robotics 31.5 (2015): 1147-1163.
[9] Zheng, Shuran, et al. "Simultaneous Localization and Mapping (SLAM) for Autonomous Driving: Concept and Analysis." Remote Sensing 15.4 (2023): 1156.
[10] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874-1890, 2021.
[11] Mur-Artal, Raul, and Juan D. Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras." IEEE transactions on robotics 33.5 (2017): 1255-1262.
[12] T. Schöps, T. Sattler and M. Pollefeys, "SurfelMeshing: Online Surfel-Based Mesh Reconstruction," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 10, pp. 2494-2507, 2020.
[13] M. Gopi and S. Krishnan, "A fast and efficient projection-based approach for surface reconstruction," Proceedings. XV Brazilian Symposium on Computer Graphics and Image Processing, Fortaleza, Brazil, 2002, pp. 179-186.
[14] Z. C. Marton, R. B. Rusu and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 2009, pp. 3218-3223.
[15] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 2012, pp. 573-580.