2025 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW) | 979-8-3315-1484-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/VRW66409.2025.00021

Mesh2SLAM in VR: A Fast Geometry-Based SLAM Framework for Rapid Prototyping in Virtual Reality Applications

Carlos A. Pinheiro de Sousa* University of Konstanz
Heiko Hamann† University of Konstanz
Oliver Deussen‡ University of Konstanz

ABSTRACT

SLAM is a foundational technique with broad applications in robotics and AR/VR. SLAM simulations evaluate new concepts, but testing on resource-constrained devices, such as VR *HMDs*, faces challenges: high computational cost and restricted sensor data access. This work proposes a sparse framework using mesh geometry projections as features, which improves efficiency and circumvents direct sensor data access, advancing SLAM research as we demonstrate in VR and through numerical evaluation.

1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a foundational method with broad applications in robotics and computer vision, critical for the operation of augmented and virtual reality (AR/VR) devices. SLAM algorithms rely on various sensors, such as LIDAR, cameras, and depth sensors, to build a map of the environment while simultaneously estimating the device's location. Visual SLAM (V-SLAM)[1] primarily uses images as input data. However, additional processing of the image stream is required to detect and match salient features between frames[2]. Accurate detection and matching of these features are vital for estimating camera poses between consecutive frames, known as *localization*.

Simulation plays a key role in efficiently testing SLAM concepts before real-world deployment [11]. It enables rapid prototyping with reduced noise, increased repeatability, and adjustable parameters for modeling environments, processes, and sensors. In SLAM simulation, as in real-world SLAM, image features are extracted from the generated images of the virtual environment. However, regardless of using modern AI methods [3, 6, 20, 21] or classic computer vision techniques [13, 22], feature-based methods introduce not only computational overhead [4, 19] but also inherent noise and ambiguities in feature matching [15, 16].

Another major challenge in SLAM research with direct use of Head-Mounted Displays (HMDs) is the restricted access to raw sensor data on commercial AR/VR hardware. While these devices often perform SLAM for real-time *localization*, this functionality is typically system-level and unavailable to users. Consequently, prototyping SLAM applications on HMDs remains limited and proprietary. Most manufacturers[3] restrict access to raw camera inputs, confining research to simulated sensors and environments.

To address these limitations, we propose an efficient and portable framework that performs monocular SLAM directly from *runtime* virtual environments, which circumvents the need for direct sensor access. Additionally, as an alternative to image-based features for SLAM, our method performs projections of mesh geometry components; we refer to it as *vertex features*. Our framework prioritizes efficiency over realism by leveraging the structural elements inherent in computer-generated environments, specifically *polygonal meshes*.

To our knowledge, this is the first real-time SLAM system to utilize polygonal mesh vertices as features in a virtual reality context that runs directly on HMD devices. It has three advantages:

- First, our approach eliminates feature association errors, enhances position estimation accuracy and boosts runtime efficiency, making it also a potential candidate for use as ground-truth in related applications.
- Second, this SLAM framework is capable of running *standalone* (without a tethered connection to a personal computer for heavy-lifting), directly as a user application on low-budget off-the-shelf HMDs, bypassing the need for real sensor input.
- Third, our framework allows the use of arbitrary 3D meshes without the need of textures.

As a result, our work expands research opportunities and prototyping for SLAM beyond VR, encompassing robotics and broader computer vision applications. It is particularly suited for SLAM research and prototyping in virtual simulation environments [5, 11].

After detailing the system and performance metrics, we showcase its application in a single-user VR interaction, mapping a one-to-one sparse representation of the virtual environment through user motion.

2 RELATED WORK

The use of image features for visual odometry dates back to NASA's 1980s Mars exploration programs [17]. Feature-based real-time visual odometry and reconstruction, or *localization and mapping*, became practical decades later with PTAM [10], which reduced computational costs by employing parallel threads for tracking and mapping.

Feature-based methods, often called indirect methods, rely on a front-end module to extract features from image sequences. While robust to photometric changes and large baselines, they demand high computational resources, challenging real-time performance requirements.

Prominent SLAM frameworks, such as ORB-SLAM [18], aim to mitigate these ongoing challenges by leveraging efficient feature extraction methods like ORB [22]. ORB-SLAM's modular design has made it widely adopted, enabling extensions and fostering innovation in the SLAM field.

Research on SLAM specifically for virtual reality (VR) remains limited. Many existing approaches rely on simulation-based methods that either do not capture the user's perspective for mapping or simply use a device's built-in SLAM framework for localization. For instance, [1] introduced a SLAM testing platform in virtual environments—described as VR—yet did not employ physical head-mounted devices (HMDs). Similarly, [25] used SLAM on the HoloLens 2, but it was driven by the system's built-in SLAM, illustrating an application of SLAM rather than advancing new SLAM techniques. In another study, [7] developed visual–inertial SLAM

*e-mail: carlos.pinheiro-de-sousa@uni-konstanz.de
†e-mail:heiko.hamann@uni-konstanz.de
‡e-mail:oliver.deussen@uni-konstanz.de

[1]This work focuses on V-SLAM specifically, for simplicity, it will be referred to as SLAM throughout this document.
[2]Considering indirect, also known as feature-based V-SLAM methods.
[3]Exceptions include Microsoft Hololens2 and Varjo.

979-8-3315-1484-6/25/$31.00 ©2025 IEEE
DOI 10.1109/VRW66409.2025.00021

for hand controllers rather than from the rendered VR environment itself.

In contrast, our proposed Mesh2SLAM performs real-time SLAM by directly leveraging polygonal meshes from virtual environments and can operate entirely in VR, with the user as the primary mapping agent.

3 METHOD

3.1 Overview

We developed Mesh2SLAM drawing inspiration from the simplicity and dual parallel modules of PTAM (Parallel Tracking and Mapping). Additionally, we adapt the initialization, tracking, and mapping processes from ORB-SLAM2, with such modules tailored to integrate with our own feature processing method.

Given the growing importance of resource-constrained devices in SLAM research [2], our method is designed to be lightweight, portable and more independent of third-party libraries. Core component modules that process image features are replaced with our *vertex feature* processing approach, furthermore, particularly for visualization, it includes its own, optional, OpenGL ES visualization engine allowing greater portability.

As shown in Figure 1, Mesh2SLAM operates concurrently across the main application thread and two dedicated SLAM threads: *Tracking* and *Mapping*. Frames are captured from the scene geometry as vertex features and processed by the front-end *Tracking* module. The back-end *Mapping* module then performs posterior optimization, including windowed bundle adjustment [23], to minimize errors and achieve the best fit for frame poses and mapped points.

The multi-threaded design highlights the importance of maintaining high performance and independence from the main rendering thread in our system.

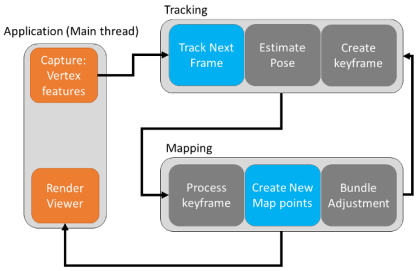

Figure 1: System overview, showing main threads; highlighted (blue) are related to vertex features processing, highlighted (orange) are executed in application main thread, arrows represent event triggers

Finally, the generated map is rendered back on the main thread, with frame poses and the point cloud map displayed in a first-person perspective, as shown in Figure 2.

The following sections detail the key processes specific to our method.

3.2 Frame Capture

For the typical operation, the mesh model is loaded into memory at start up. The sequence of vertex components is registered ensuring uniquely identified vertices by stored index.

At each iteration of the main rendering loop, the model's vertices are rendered or *captured* as vertex features and are input into the SLAM system. The capture process utilizes a custom *compute shader* [9] to project vertices into the virtual camera's image plane for each frame see Fig. 3.

The resulting vertex features are output as a list of features with their specific identifiers and image coordinates. The details of the computation are clarified:

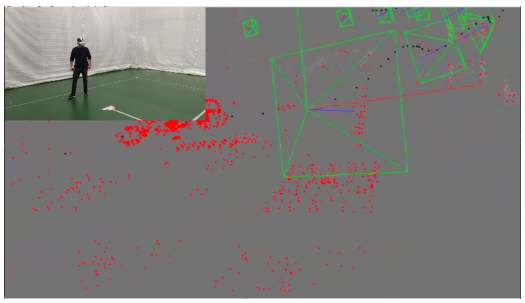

Figure 2: A screenshot live VR while running Mesh2SLAM.

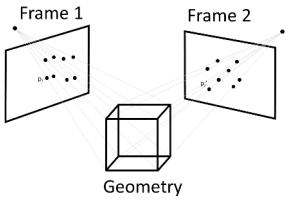

Figure 3: Direct geometric projection of vertices to the camera view in Mesh2SLAM.

Let $\mathbf{p}_i$ represent the position of the i-th vertex in world space:

$$\mathbf{p}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

The model vertices are projected into image space for every frame following traditional model-view-projection matrix operations (1), where, P is the perspective projection, V is the camera view matrix and M is the model matrix and $\mathbf{p}_i$ now represented in *homogenous coordinates*

$$\mathbf{v}_{\text{clip}} = \mathbf{P} \cdot \mathbf{V} \cdot \mathbf{M} \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \quad (1) \qquad \mathbf{v}_{\text{view}} = \mathbf{V} \cdot \mathbf{M} \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \quad (2)$$

Complementarily, z components, representing depth with respect to the camera, are kept in view space (2) in order to assert depth thresholds; Depth can be used to include only vertexes that are within acceptable range.

The values for clip and view can now be used to compute the normalized screen coordinates (ndc) (3a),(3b) and depth (3c):

$$x_{\text{ndc}} = \mathbf{v}_{\text{clip}}.x \cdot \mathbf{v}_{\text{clip}}.w^{-1}, \quad (3a)$$

$$y_{\text{ndc}} = \mathbf{v}_{\text{clip}}.y \cdot \mathbf{v}_{\text{clip}}.w^{-1}, \quad (3b)$$

$$z = -\mathbf{v}_{\text{view}}.z. \quad (3c)$$

And finally screen coordinates u, v are obtained with w and h as width and height of screen resolution:

$$u = (x_{\text{ndc}} \times 0.5 + 0.5) \times w, \quad v = (-y_{\text{ndc}} \times 0.5 + 0.5) \times h. \quad (4)$$

As a result, a *vertex feature* aggregates the screen coordinates along with the descriptor, the vertex's unique identifier i:

$$\mathbf{v}_i = \begin{pmatrix} u_i \\ v_i \\ i \end{pmatrix}$$

Next we clarify how our approach provides advantages for feature processing.

3.3 Feature Proccesing with vertex features

In feature-based SLAM processing, the association and matching of feature correspondences between frame pairs are computed by calculating the *similarity* between the feature descriptors. Our approach streamlines this process by leveraging unique IDs as descriptors, i.e, a simple integer comparison, allowing for more efficient and errorless feature association.

Traditional image-feature methods, suffer from degraded feature matching under large parallax or substantial baselines between camera poses. This is often mitigated by limiting the average normal divergence angle between frames that observe the same feature [18]. Our approach eliminates this constraint since regardless of angle, *vertex features* are viewpoint-invariant. Similarly, traditional methods require scale matching to ensure consistent feature measurement across frames. With our approach, this is unnecessary, as scale invariance is inherently enforced by depth threshold limits.

Figure 4 illustrates a comparison between traditional image features and our approach *vertex features*

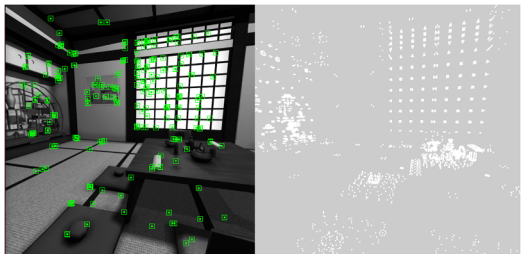

Figure 4: Left: A display of image-based features. Right: Our approach vertex features.

3.4 SLAM Loop

The system functionality follows that of ORB-SLAM2 and can be broken down to: initialization then parallel tracking and mapping. At initialization, the system attempts to create an initial map. Then once a map is initialized, Tracking and Mapping process the sequential frames containing *vertex features* data in parallel. For tracking, camera pose estimations are computed with association of features with previous frame. For mapping, new map points are created by triangulation, provided that, among other conditions, enough unmapped features of previous frame match with the new frame's features observations.

3.5 Viewer

The visualization component is managed by our custom *Viewer* module, which runs on the main application thread. It is updated in response to new map update events. The *Viewer* rendering itself follows the frame rate of the main application, independently of SLAM performance. The result is a generated sparse point cloud map that accurately overlays the original mesh geometry as the user scans the virtual environment as seen in 4.

For this application, *Viewer* has been integrated with *OpenXR* and will be described next.

3.6 OpenXR Integration

For proper application in virtual reality, the camera pose or reference used for geometry projection should match that of the user's perspective or viewpoint. In this monocular setup, the reference is centered in the device's View Space, (Fig. 5) which serves as a frame of reference as defined by the OpenXR API.

Figure 5: *View Space*, used as reference for vertex feature extraction (obtained from [8]).

4 Experiments and Results

In this section, we demonstrate the efficiency and effectiveness of our method under several important metrics: 1) Efficiency of Vertex Feature Extraction, 2) System Performance and 3) Virtual Reality Practical Evaluation

4.1 Efficiency of Vertex Feature Extraction

We conducted experiments using polygonal meshes ranging from 500 to 2 million vertices on both a mid-range notebook (Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, Intel® Iris Plus Graphics G7, 16 GB of RAM) and a low-budget (mid-range) HMD, the Oculus Quest 2 [14].

The performance metric used is the average time in *milliseconds* from the dispatch of the compute shader program to the output of the compute shader of resulting vertex feature components into vector storage, additionally, since this process runs on the main application thread, i.e. rendering loop, the nominal application framerate is also measured in frames-per-second (FPS). The results are presented in Table 1 below:

Vertex Count	PC (ms)/ FPS	Quest 2 (ms) / FPS
600	12 /75	0.006 /72
60 000	12 /75	2 /72
240 000	12 /75	8 /72
480 000	12 /75	14 /58
2 000 000	12 /75	60 /14

Table 1: Median time (in ms) for 1000 runs for different counts of mesh vertex feature computation on a personal computer (PC) and Quest 2.

While the personal computer maintained stable performance across all vertex counts, the Meta Quest 2 exhibited performance degradation beyond 400,000 vertices, as shown by the drop in frame rate. We believe this correlates with Quest's performance budget highlighted in [14], also because no frustum culling is performed on our side for these tests. Nonetheless, the performance of vertex feature extraction method far surpasses that of traditional image-based feature extraction, which typically does not exceed a few thousand features extracted per image.

4.2 System Performance

4.2.1 Experiment Setup

Virtual environment: Next we present an evaluation of localization accuracry of Mesh2SLAM in comparison to the baseline ORB-SLAM2. The overall tracking behaviour with varying camera image input frame-rates, 15, 30, 60 and 75 and accuracy are tested on a PC running computer graphics-generated environment specifically designed for both methods.

For image feature-based method, the scene is pre-rendered as an image sequence at resolution of 1024 x 1024 pixels and additional lighting and textures are used aiming for a realistic look (see Fig. 6). Concurrently, the environment consists of a scene composed by

polygonal meshes used as source for the vertex feature extraction for our method.

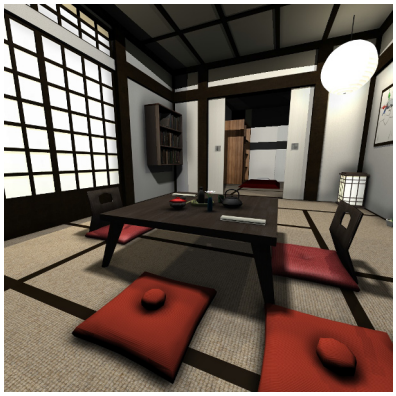

Figure 6: Pre-rendered scene, used for image feature extraction.

Camera setup: For the evaluation, a first-person style camera motion is recorded and is re-used (show in figure 7). The camera projection uses configuration with a field-of-view of 90 degrees and no lens-distortion.

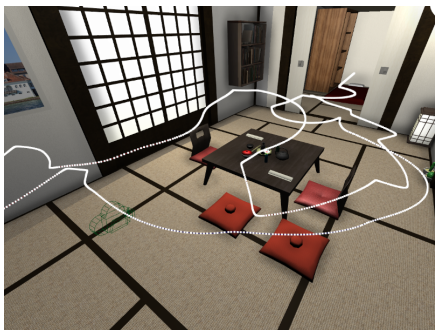

Figure 7: Prerecorded camera trajectory/poses which is re-used for evaluation.

ORB-SLAM2 Performance: ORB-SLAM2 has been altered for asynchronous input frame-rate i.e. with possible frame skipping if processing time takes longer than expected for next sequential frame. Such adaptation assures a fair comparison with our method which processes input frames in asynchronous manner as well.

With the original source image resolution of 1024 by 1024 pixels, ORB-SLAM2 could only operate at an average of 15 FPS with eventual loss of tracking. At higher frame-rates, 30, 60 and 75 was only possible by halving of image resolution and led to recurring loss of tracking with abrupt movements. Further reduction in input image resolution deemed unfeasible due to poor system performance.

Mesh2SLAM Performance: Our method functions effectively across all tested frame rates: 30, 60, and 75 FPS (The test was conducted on a PC where the refresh rate was capped at 75 Hz due to hardware limitations).

In table 2 we briefly present the results which reflect the degradation of localization with increasing frame-rates. The evaluation of SLAM performance is typically measured by comparing estimated camera pose trajectories to ground truth [12, 24]. These values present the Absolute Trajectory Error, RMSE. As expected, Mesh2SLAM has very high accuracy result, standing around an order of magnitude lower error than the baseline method for higher frame rate. A visual inspection of the tracking quality through the trajectory of camera sequence is presented next, with figure in Appendix A.

Frame Rate	Mesh2SLAM (RMSE)	ORB-SLAM2 (RMSE)
30	0.037 ± 0.026	0.124 ± 0.065
60	0.065 ± 0.039	0.147 ± 0.060
75	0.095 ± 0.060	2.011 ± 1.033

Table 2: Comparison of performance for systems Mesh2SLAM and ORB-SLAM2 at different frame rates with associated errors.

4.2.2 Absolute Trajectory Error Visual Inspection

As shown in Figure 8 (see Appendix A) and summarized in the table above, the localization accuracy of our method significantly outperforms ORB-SLAM2 when compared to the ground truth. We include frame rates of 30 FPS to match ORB-SLAM2's ideal rate and 75 FPS as typically desired for VR. Notably, at the end of the sequence at 75 FPS, ORB-SLAM2 loses tracking entirely, indicated by missing aligned red line segments in the figure.

4.3 Virtual Reality Practical Evaluation

For a practical functionality with a real-time interactive demonstration we evaluate the application running as *standalone* on a Meta Quest 2. It starts with the user at scene origin in the computer-generated environment. As SLAM performs in real-time, the mapping provides the reconstruction of the scene as sparse set of red points, which correspond to the mapped vertexes of the virtual environment and the corresponding user poses at each instant. Subsequently as the user moves and interactively looks around, the scene is mapped accordingly. Finally, on a qualitative description, this demo runs at 72 FPS with no noticeable jitter or freezing; vertex feature extractions have no practical overhead and the SLAM threads run independently from the main application rendering thread.

5 Scope and Constraints

Mesh2SLAM is limited to virtual environments, which restricts its applicability to real-world scenarios. Additionally, unbalanced spatial density or clustering of mesh vertices can impact tracking performance. Although the scene is currently treated as a single mesh, future extensions could handle large-scale environments with multiple meshes, which may be achieved through other optimization, sorting, or culling mechanisms. While our evaluation focused on the Meta Quest 2, we expect testing on other HMDs will likely yield similar results and provide a more comprehensive assessment.

6 Conclusion

In this work, we presented Mesh2SLAM, a fast and efficient geometry-based SLAM framework for prototyping that operates on HMDs in virtual reality. Its main novelty lies in utilizing mesh geometry components, *vertex features*, and leveraging GPU acceleration to enhance position estimation accuracy and drastically reduce computational overhead. To our knowledge, it is the first real-time SLAM method running in a virtual environment as a *standalone* application, without tethering, on low-budget, off-the-shelf HMDs.

As a SLAM simulation, Mesh2SLAM maintains SLAM functionality while bypassing issues in image-based methods that hinder prototyping. In the context of XR development, its lightweight sparse map representation is well-suited for localization-focused tasks, including tracking user movement, supporting spatial interactions, validating pose estimation methods, and benchmarking other localization algorithms. Additionally, future considerations include extending it to collaborative localization or multi-agent SLAM. These capabilities make it a valuable tool for efficiently testing and refining concepts, driving advancements in XR research.

A Additional Figures

Below we present additional illustrations of the Absolute Trajectory Error (ATE) for ORB-SLAM2 and Mesh2SLAM at different frame rates. This figure highlights the trajectory alignment and differences between the estimated and ground-truth camera poses.

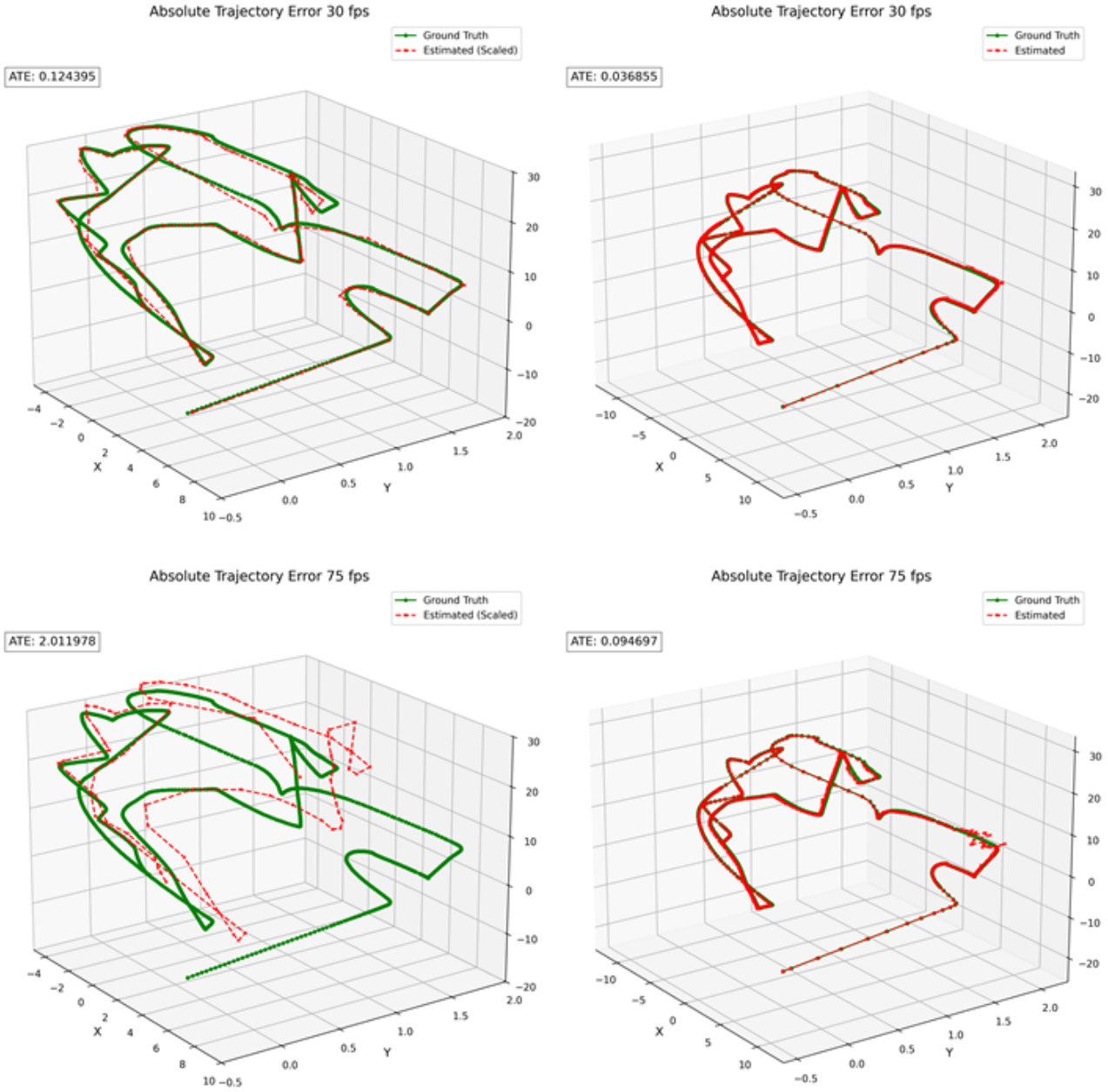

Figure 8: ATE comparison at 30 FPS and 75 FPS for ORB-SLAM2 (left) and Mesh2SLAM (right).

REFERENCES

[1] A. Bettens, B. Morrell, M. Coen, N. Mchenry, X. Wu, P. Gibbens, and G. Chamitoff. UnrealNavigation: Simulation Software for testing SLAM in Virtual Reality. In *AIAA Scitech 2020 Forum*, 2020. doi: 10.2514/6.2020-1343

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[3] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description, 2018.

[4] A. Dhakal, X. Ran, Y. Wang, J. Chen, and K. K. Ramakrishnan. SLAM-share: Visual Simultaneous Localization and Mapping for Real-Time Multi-User Augmented Reality. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, pp. 293–306. Association for Computing Machinery, New York, NY, USA, 2022. doi: 10.1145/3555050.3569142

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator, 2017.

[6] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler. D2-net: A trainable cnn for joint detection and description of local features, 2019.

[7] X. Jiang, L. Zhu, J. Liu, and A. Song. A slam-based 6dof controller with smooth auto-calibration for virtual reality. *Vis. Comput.*, 39(9):3873–3886, June 2022. doi: 10.1007/s00371-022-02530-1

[8] Khronos Group. *OpenXR 1.0 Specification*, July 2019.

[9] Khronos Group. *Compute Shader - OpenGL Wiki*, September 2024.

[10] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10. IEEE, 2007. doi: 10.1109/ISMAR.2007.4538852

[11] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154. IEEE, 2004. doi: 10.1109/IROS.2004.1389727

[12] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27:387–407, 2009.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. doi: 10.1023/B:VISI.0000029664.99615.94

[14] Meta. Unity performance guidelines, n.d. Accessed: 2024-09-17.

[15] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, Nov 2005. doi: 10.1109/TPAMI.2005.188

[16] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2):43–72, Nov 2005. doi: 10.1007/s11263-005-3848-x

[17] H. P. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Stanford University, 1980.

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671

[19] F. Muzzini, N. Capodieci, R. Cavicchioli, and B. Rouxel. Brief announcement: Optimized gpu-accelerated feature extraction for orb-slam systems. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pp. 299–302. Association for Computing Machinery, New York, NY, USA, 2023. doi: 10.1145/3558481.3591310

[20] G. Potje, F. Cadar, A. Araujo, R. Martins, and E. R. Nascimento. Xfeat: Accelerated features for lightweight image matching, 2024.

[21] J. Revaud, P. Weinzaepfel, C. D. Souza, N. Pion, G. Csurka, Y. Cabon, and M. Humenberger. R2d2: Repeatable and reliable detector and descriptor, 2019.

[22] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. pp. 2564–2571, 11 2011. doi: 10.1109/ICCV.2011.6126544

[23] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92, 2011. doi: 10.1109/MRA.2011.943233

[24] Z. Zhang and D. Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7244–7251, 2018. doi: 10.1109/IROS.2018.8593941

[25] M. Zins, G. Simon, and M.-O. Berger. Oa-slam: Leveraging objects for camera relocalization in visual slam, 2022.