gsplatam: Real-time Splat, Track, and Map with gsplat

Chahyon Ku[1], David Wang[2], Ruihan Chen[2], Yicheng Zou[2]

***Abstract*— Many prior works have proposed using 3D Gaussian splatting (3DGS) [1] as the map representation for simultaneous localization and mapping (SLAM). Among the earliest works, SplaTAM [2], or splat, track, and map, utilizes silhouette based rendering to identify unmapped regions and optimize the camera pose and the map with photometric and depth losses. While SplaTAM provides high quality reconstruction with a simple optimization process, it has limitations when used in online robotics settings. First, there is a lack of online 3D visualization for debugging and monitoring the state of the algorithm. Second, it is not optimized, with 2 to 0.5 frames per second throughput on a desktop GPU. Lastly, the 3D Gaussians can also be geometrically inaccurate, as looking at it from different angles can represent different surfaces. In this project, we improve upon SplaTAM on these 3 shortcomings by building an online interactive visualizer, optimizing the SLAM pipeline, and comparing 3DGS with more geometrically accurate 2D Gaussian splats [3]. We make available our open-source implementation at https://github.com/chahyon-ku/gsplatam along with a video of our presentation at https://youtu.be/T2NLEBzQ5w8.**

I. Introduction

Simultaneous localization and mapping (SLAM), the task of obtaining a model of the robot's surroundings while also keeping track of its state, is an essential capability for robotic systems to operate in unknown and unstructured environments. Dense RGBD SLAM utilizes recently popularized RGBD cameras to produce dense (pointcloud, sdf, or volumetric rendering) representations of the world.

Immediately after 3D Gaussian splatting [1] was introduced, many SLAM methods proposed using it as the underlying map representation for SLAM due to its explicit and efficient nature [4]. In order to adapt 3DGS to visual SLAM, these methods incorporate a combination of algorithms including sparse sampling of Gaussians [5], local optimization with keyframes [6], [7], integration of existing sparse SLAM methods with loop closure [8], or masking inactive regions [2], [9]. However, many such methods are still slow to run with <3 fps runtime, due to 3DGS's slow optimization process tailored to scene reconstruction tasks.

Therefore, we propose an extension of SplaTAM [2] to provide real-time performance and visualization. First, we provide an interactive visualizer that allows the users to inspect the rendered Gaussian splats as the SLAM algorithm executes without slowing down the training. Second, we optimize the rasterization process with the gsplat [10] library while keeping the underlying algorithm consistent with the original implementation, speeding up the SLAM pipeline by a factor of 4 and achieving 10 frames per second runtime. Lastly, we compare the performance of isometric/anisometric 3D/2D Gaussian splats [1], [3] in terms of photometric accuracy, geometric accuracy, and runtime.

*This work was not supported by any organization

[1]Chahyon Ku is with the Department of Robotics, University of Michigan chahyon@umich.edu

[2]David Wang, Ruihan Chen, and Yicheng Zou are with the Department of Electrical and Computer Engineering, University of Michigan davwan@umich.edu rhchen@umich.edu yichzou@umich.edu

II. Related Works

A. Dense RGBD Simultaneous Localization and Mapping

In this section, we briefly review various approaches to dense SLAM [11]. Dense RGBD SLAM is a core problem in robotics and computer vision, involving the task of simultaneously localizing a robot and building a dense 3D map of the surrounding environment using RGBD sensor data. This capability is essential for applications such as robotic navigation, augmented reality, grasping, and manipulation.

Traditional SLAM systems build explicit scene models—ranging from TSDF (Truncated Signed Distance Functions) [12] and voxel hashing [13] to Gaussian mixture models [14] and surfel-based surface elements [15] —each trading off memory, accuracy, and speed. Surfels, in particular, represent colored circular patches that can be updated in real time but are inherently discontinuous and prone to holes at depth edges; modern differentiable rasterizers enable gradient flow through these depth discontinuities [16] yet still demand careful regularization to close gaps and preserve surface integrity.

NeRF-based SLAM [17] builds on the power of neural implicit radiance fields. These methods were originally developed for novel view synthesis [18], dynamic scene modeling and generalization across scenes. The current methods seek to jointly optimise continuous scene geometry and camera poses.

B. 3D Gaussian Splatting for Simultaneous Localization and Mapping

SplaTAM [2] represents the scene as a compact set of isotropic 3D Gaussian splats and, for every incoming RGBD frame, alternates three differentiable-rendering steps: (i) silhouette-masked camera tracking that refines the image and depth reconstruction of the RGBD frame; (ii) densification that adds new Gaussians to the map wherever the mask or depth residual reveals unseen geometry. and (iii) update the parameters of all the Gaussians in the scene by minimizing the RGB and depth errors over all images, while discarding low-utility splats. This silhouette-guided loop sustains real-time SLAM and underpins the method's

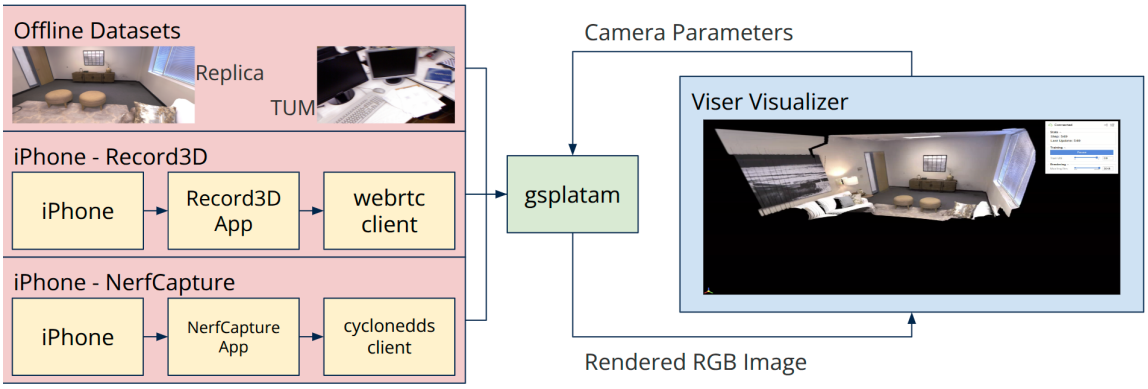

Fig. 1: An overview of our interactive visualization. RGBD images from offline datasets, the Record3D App, or the NerfCapture App (left) are fed into our gsplatam module to be visualized interactively as the SLAM pipeline executes (right)

state-of-the-art tracking accuracy and photorealistic reconstructions, giving it enough speed to optimize every pixel rather than sparse samples.

PhotoSLAM [8] keeps a single hyper-primitives map whose elements fuse sparse ORB keypoints with 3D Gaussian splats carrying pose, density and SH-color. Camera poses are estimated on-the-fly by motion-only and local bundle adjustment over these features, while new splats are triangulated and refined in a parallel geometry-mapping thread. A photorealistic thread renders each frame with 3D Gaussian splatting and optimizes all primitive parameters via an SSIM loss, aided by (i) geometry-based densification that seeds extra splats at inactive keypoints and (ii) Gaussian-pyramid learning that trains from coarse to fine image scales. A loop-closure module corrects drift by similarity transforms. Implemented in C++/CUDA, the system runs in real time on monocular, stereo or RGB-D cameras, even on embedded GPUs.

GS-SLAM [19] reformulates dense visual SLAM around a 3D Gaussian scene model that can be rasterized in real time. The system maintains a set of anisotropic Gaussians that are jointly optimized. For each keyframe, (i) projects the current map, identifies pixels with low accumulated opacity or large depth error, and inserts new Gaussians at the corresponding 3D positions, while suppressing outliers whose depths conflict with the sensor; and (ii) refines all Gaussian parameters by minimizing full-image RGB and depth residuals. Incoming frames are tracked with a two-stage optimizer: a low-resolution render supplies a robust pose seed, after which only depth-consistent Gaussians are re-rendered at full resolution to complete a gradient-based pose update. Periodic local bundle adjustment again couples recent keyframe poses with the Gaussian map.

III. Interactive Online Visualization

We extended the original Splatam iPhone demo to update in real time as RGB-D and pose data are received. We also developed an improved visualization pipeline to handle live updates and with additional controls. Compared to the original demo, we recorded a performance improvement of around 2.3x for tracking and 3.1x for mapping.

A. iPhone Applications

NeRFCapture [20]. We utilized the NeRFCapture app installed on an iPhone to stream RGB-D images (captured via the iPhone's LiDAR sensor) and ARKit poses to a computer via CycloneDDS [21]. Rather than the original approach of buffering until all frames are received, we dynamically update our Gaussian splat representation by tracking/mapping on each new frame as it is received, allowing for on the fly updates.

Record3D [22]**.** We also tried another available app for streaming RGBD images over WIFI, Record3D. Unlike NeRFCapture [20] which streams the full-resolution RGBD image along with the camera pose recognized by the ARKit backend, Record3D streams a much smaller-resolution RGBD image without camera poses using the webrtc [23] protocol commonly used for web conferencing. While this improves the number of frames that can be transmitted from around 1 fps to 15 fps, it also has greatly reduced resolution, with images around 500 by 500 pixels and depth resoultion of 180 steps over 3 meters.

Metric	Original	Ours)
Tracking Time / Iteration (ms)	43.33	18.63
Tracking Time / Frame (s)	2.88	1.19
Mapping Time / Iteration (ms)	46.89	15.26
Mapping Time / Frame (s)	2.81	0.92

TABLE I: NeRFCapture iPhone Demo Performance

B. Visualizers

Open3D [24]**.** We built a custom visualizer utilizing Open3D's `VisualizerWithKeyCallback` to showcase the SLAM process live as iPhone data is received. We implemented a dynamic update feature, which waits at every timestep for the latest splat checkpoints to be computed and saved by the iPhone demo process. Geometry is then updated via GaussianRasterizer and rendered in the Open3D

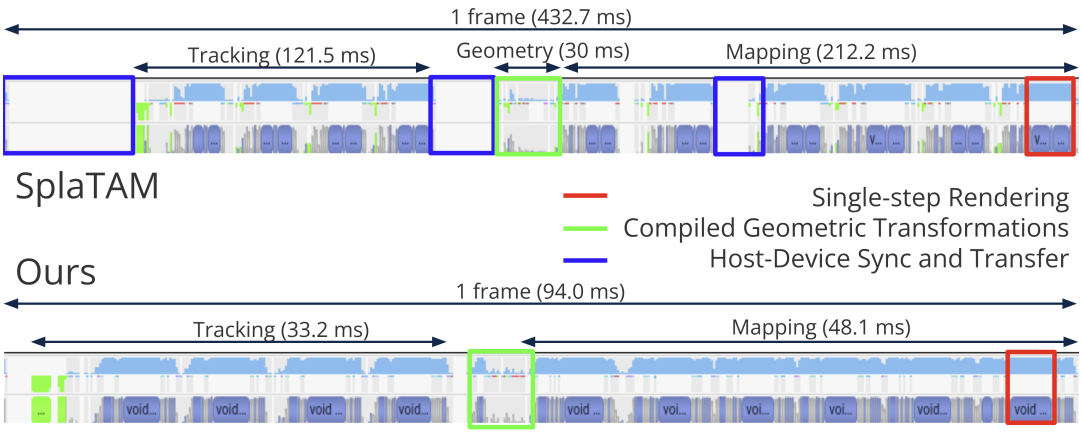

Fig. 2: Performance comparison between the original implementation (top) and ours (bottom).

event loop. Using inter-process communication, when the user closes the visualization, we inform the iPhone demo process to stop receiving additional data, and to save the final splats for future use. Additionally, our visualizer features 6 DoF controls implemented via custom key callbacks for translations, along with mouse controls to rotate and zoom the camera. We also designed a lightweight GUI overlay with Tkinter to display information to the user, such as the camera and translation controls.

Viser [25]**.** Viser is a standard library for visualizing neural radiance fields and gaussian splats. It is built to asynchronously update its views similar to how we implemented our Open3D visualizer. The biggest difference between the viser visualizer and the open3d visualizer is that the viser visualizer properly renders gaussians, yielding photorealistic results based on the rasterization equaution, instead of rendering the gaussian splats as a point cloud with fixed size and color.

IV. Performance Optimization

We make 3 key improvements to the original SplaTAM pipeline, reducing runtime from 430 milliseconds per frame to 95 milliseconds per frame. First, we optimize the rasterization of Gaussians by using the single-step rendering CUDA kernel from the gsplat [10] library, which jointly renders RGB, depth, and the silhouette, bringing down the runtime of tracking and mapping by a factor of 4. Second, we make use of Torch's just-in-time compiler to optimize geometric functions in keyframe selection and densification. Finally, we make all costly device-to-host and host-to-device memory copies asynchronous, removing gaps and increasing GPU utilization.

A. Single-step Rendering

The majority of SplaTAM's compute time is spent on the forward and backward passes for the CUDA kernel that rasterizes Gaussians. SplaTAM implements this using 2 rasterization passes, where the first pass renders the RGB image and the second pass renders the depth and silhouette, due to the limitation of the original 3DGS implementation having a fixed number of channels. On the other hand, the rasterization kernel from the gsplat [10] library features a dynamic number of channels with built in alpha channel (silhouette) and expected depth rendering, simplifying the rendering into just a single rasterization pass. Furthermore, as gsplat offers gradient backpropagation to camera parameters, the camera pose can be directly optimized in an optimized CUDA kernel instead of manually multiplying the transformation matrix to the Gaussians in pytorch.

B. Compiled Geometric Transformations

The addition of new Gaussians based on silhouette, as well as selection of keyframes for mapping, requires various geometric processing such as back-projecting depth images into a point cloud and checking for overlap between multiple camera views. We use specialized CUDA kernels provided in the gsplat [10] library (*fully_fused_projection*), as well as just-in-time compilation provided by the pytorch [26] library to reduce the runtime of keyframe selection from 30 ms to 5 ms.

C. Host-Device Sync and Transfer

A critical component of performant GPU code that is often overlooked is the overhead from memory transfer between the RAM and the GPU's memory. In SplaTAM's case, data was synchronously read from disk and loaded onto the GPU, which resulted in 10s of milliseconds of gaps where the GPU was not computing anything. By making this pipeline asynchronous for both offline datasets and iPhone demos, we were able to maximize GPU utilization and improve our runtime.

V. Experiments

We conduct experiments on 2 **datasets**. First, *Replica* [27] is a dataset of 18 photo-realistic scene reconstructions. Following standard practice from NICE-SLAM [16] and SplaTAM [2], we use 1200 by 680 RGBD images and camera poses synthetically generated by the authors of iMAP [28]. We conduct all our experiments on the Room-0 scene consisting of 2000 frames of smooth and random camera trajectory covering a single room.

Setting	Methods	Track Time (s/frame) ↓	Map Time (s/frame) ↓	Total Time (s/frame) ↓	Num Gaussians ↓	ATE RMSE (cm) ↓	PSNR (db) ↑	Depth L1 (cm) ↓
base	reported	1.00	1.44	-	-	0.27	32.81	0.49
	reproduced	2.74	4.94	7.85	5,085,417	0.32	32.48	0.51
	ours	0.32	0.59	0.93	973,059	0.05	35.78	0.25
small	reported	0.19	0.33	-	-	0.39	-	-
	reproduced	0.27	0.45	0.84	931,214	0.52	29.29	0.83
	ours	0.08	0.14	0.24	1,101,708	0.28	30.44	0.55
tiny	reported	-	-	-	-	-	-	-
	reproduced	0.10	0.17	0.39	880,241	6.40	22.97	4.31
	ours	0.03	0.05	0.10	954,972	0.26	22.92	3.28

TABLE II: Replica Room 0

Second, we also evaluate on TUM RGB-D [29], a dataset consisting of 39 indoor sequences of posed RGBD images (640 by 480, 30 Hz) recorded with a Microsoft Kinect. As this dataset contains real noisy images collected from a naturally jittery camera trajectory, it better resembles a robotics use case compared to the synthetic Replica dataset. We conduct all our experiments on the desk scene consisting of 592 frames.

We compare the performance of our work over 3 **settings**, which vary in the number of optimization steps for tracking and mapping. The *base* and *small* settings are as defined in the original paper, with 40/60 and 10/15 iterations for tracking and mapping. The *tiny* setting is our minimal testing setup with 4/6 iterations for tacking and mapping.

The **methods** indicate whether the results are directly from the tables in the original paper (*reported*), evaluating the original implementation on our hardware (*reproduced*), and evaluating our improved implementation on our hardware (*ours*). Note that all *reported* experiments were done on a Desktop with a 3080 Ti by the original authors, while all *reproduced* and *ours* experiments were done on our Laptop with a 4090.

We make available our open-source **implementation** at https://github.com/chahyon-ku/gsplatam along with a **video** of our presentation at https://youtu.be/T2NLEBzQ5w8.

A. Replica Dataset

The results on the Replica dataset are reported in Table II.

Runtime. The reproduced runtime is 1.3-3.4 times slower than the reported, with the biggest gap in the *base* setting. This is likely due to the limited compute provided by the mobile GPU. On the other hand, our improved implementation runs 3 to 8 times faster than the original implementation on the same hardware, thanks to the performance optimizations we implemented. Lastly, the total time highlights a hidden runtime cost to SplaTAM that is not included in the track and mapping times: keyframe selection and densification. Even with similar number of Gaussians, the geometric processing in these steps take 6 times as longer for the original implementation (0.84 - 0.45 - 0.27 = 0.12) compared to ours (0.24 - 0.14 - 0.08 = 0.02).

Accuracy. Our reproduced results match the reported performance metrics within a margin. In Table II, ATE RMSE, PSNR, and Depth L1 represent the error in the camera trajectory, the photometric error, and the geometric error, respectively. Compared with the original implementation, our camera tracking is significantly better in all settings; our mapping is comparable for the tiny and small setups, while superior in the base setup.

B. TUM Dataset

The results on the TUM dataset are reported in Table III. Due to the noisy and jittery camera views, a lot more tracking and mapping iterations are required to optimize the camera pose the map. To specify, both the *small* and *tiny* settings lose track of the camera in the middle, resulting in extremely

Setting	Methods	Track Time (s/frame) ↓	Map Time (s/frame) ↓	Total Time (s/frame) ↓	Num Gaussians ↓	ATE RMSE (cm) ↓	PSNR (db) ↑	Depth L1 (cm) ↓
tum	reported	-	-	-	-	3.35	-	-
	reproduced	3.76	0.64	4.46	970,241	3.28	22.91	3.05
	ours	1.50	0.26	1.78	1,037,469	0.29	23.46	2.78
base	reported	-	-	-	-	-	-	-
	reproduced	0.73	1.24	2.03	846,272	3.52	23.10	2.83
	ours	0.28	0.49	0.79	850,027	0.18	22.92	2.72
small	reported	-	-	-	-	-	-	-
	reproduced	0.19	0.32	0.56	1,055,474	31.77	18.23	5.62
	ours	0.06	0.13	0.21	1,252,392	0.21	16.22	7.78
tiny	reported	-	-	-	-	-	-	-
	reproduced	0.11	0.20	0.37	2,597,056	130.51	13.19	16.09
	ours	0.06	0.11	0.19	5,810,278	0.51	13.89	14.28

TABLE III: TUM desk

Type	Covariance	Track Time (s/frame) ↓	Map Time (s/frame) ↓	Total Time (s/frame) ↓	Num Gaussians ↓	ATE RMSE (cm) ↓	PSNR (db) ↑	Depth L1 (cm) ↓
3D	Isotropic	0.03	0.05	0.10	984,531	0.26	23.17	2.58
3D	Anisotropic	0.03	0.05	0.10	954,972	0.26	22.92	3.28
2D	Isotropic	0.12	0.16	0.30	1,193,273	0.60	23.66	2.33
2D	Anisotropic	0.11	0.16	0.29	1,171,766	0.60	23.11	2.74

TABLE IV: Gaussian Ablation

low performance. However, for higher-iteration settings such as *tum* (200/30, from original paper's TUM experiments) and *base*, where the camera is correctly tracked, the same findings from the Replica dataset apply: our implementation shows substantial improvement in camera tracking and similar performance in mapping accuracy, while being 1.8-3.2x faster. While these results indicate that solely relying on Gaussian splats for both camera tracking and mapping may not be computationally feasible in realistic scenarios, we believe the performance improvements will still benefit future work with integrated keypoint-based SLAM backends.

C. 3D Gaussian Splatting vs. 2D Gaussian Splatting

The performance of 3D Gaussian Splatting vs. 2D Gaussian Splatting [3] in the rendering backend is shown in Table IV. 2DGS provides slightly better accuracy for both isotropic and anisotropic settings due to its geometric accuracy, traded off with 3 times slower runtime from a different, more complicated rasterization equation. Furthermore, in the "tiny" setting we are using for this experiment, it seems that the isotropic Gaussians optimize more quickly, leading to slightly better accuracy of the map.

VI. Conclusion

In summary, we present runtime optimization, online visualization, and a comparison of various types of Gaussians using various open-source libraries such as gsplat [10], open3d [24], and viser [25]. We successfully demonstrate the benefits of both the optimization and the visualizer in quantitative analysis on offline datasets as well as real-time demos streaming RGBD images from our iPhones. Finally, we open-source our implementation along with instructions to reproduce the quantitative results as well as our demo at https://github.com/chahyon-ku/gsplatam.

The main limitation of our work is the loss of tracking. 3D Gaussian splatting excels at local optimization of camera poses from photometric and depth losses, but often loses tracking with larger camera movement or motion blur, as it does not have a way to associate distant pixels based on keypoints. Hence, future work includes integrating sparse keypoint SLAM for initializing the pose optimization as well as a more robust backend for recovery even in case of tracking failure.

REFERENCES

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, 2023.

[2] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten, "Splatam: Splat, track & map 3d gaussians for dense rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[3] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024.

[4] S. Zhu, G. Wang, X. Kong, D. Kong, and H. Wang, "3d gaussian splatting in robotics: A survey," 2024.

[5] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li, "Gs-slam: Dense visual slam with 3d gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[6] H. Matsuki, R. Murai, P. H. J. Kelly, and A. J. Davison, "Gaussian Splatting SLAM," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[7] Y. Fu, S. Liu, A. Kulkarni, J. Kautz, A. A. Efros, and X. Wang, "Colmap-free 3d gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[8] H. Huang, L. Li, C. Hui, and S.-K. Yeung, "Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[9] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, "Gaussian-slam: Photo-realistic dense slam with gaussian splatting," 2023.

[10] V. Ye, R. Li, J. Kerr, M. Turkulainen, B. Yi, Z. Pan, O. Seiskari, J. Ye, J. Hu, M. Tancik, and A. Kanazawa, "gsplat: An open-source library for Gaussian splatting," 2024.

[11] H. C. Y. L. D. S. J. N. I. R. Cesar Cadena, Luca Carlone and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," in *IEEE Transactions on robotics, 32(6):1309–1332*, 2016.

[12] M. L. Brian Curless, "A volumetric method for building complex models from range images," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.

[13] S. I. M. S. M Nießner, M Zollhöfer, "A volumetric method for building complex models from range images," in *ACM Transactions on Graphics (ToG)*, 2013.

[14] C. X. C. W. Y. Q. Meng Wang, Junyi Wang, "Og-mapping: Octree-based structured 3d gaussians for online dense mapping," in *arXiv preprint arXiv:2408.17223*, 2024.

[15] D. F. R. A. Newcombe and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[16] S. P. V. L. W. X. H. B. Z. C. M. R. O. Zhu, Zihan and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *IEEE/CVF conference on computer vision and pattern recognition*, 2022.

[17] S. P. T. M. B. J. R. R. N. R. Mildenhall, B., "Nerf: Representing scenes as neural radiance fields for view synthesis," in *The European Conference on Computer Vision*, 2020.

[18] M. B. T. M. H. P. M.-B. R. S. P. Barron, J.T., "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *arXiv:2103.13415*, 2021.

[19] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li, "Gs-slam: Dense visual slam with 3d gaussian splatting," in *CVPR*, 2024.

[20] J. Abou-Chakra, Mar 2023. [Online]. Available: https://github.com/jc211/NeRFCapture

[21] Eclipse, 2018. [Online]. Available: https://github.com/eclipse-cyclonedds/cyclonedds

[22] M. Simonik, 2023. [Online]. Available: https://record3d.app/

[23] WebRTC, 2021. [Online]. Available: https://webrtc.org/

[24] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[25] N. S. Team, 2022. [Online]. Available: https://github.com/nerfstudio-project/viser

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[27] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, "The Replica dataset: A digital replica of indoor spaces," *arXiv preprint arXiv:1906.05797*, 2019.

[28] E. Sucar, S. Liu, J. Ortiz, and A. Davison, "iMAP: Implicit mapping and positioning in real-time," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[29] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.