A Survey of 3DGS SLAM

Huang Xiaotong

2025–11–14

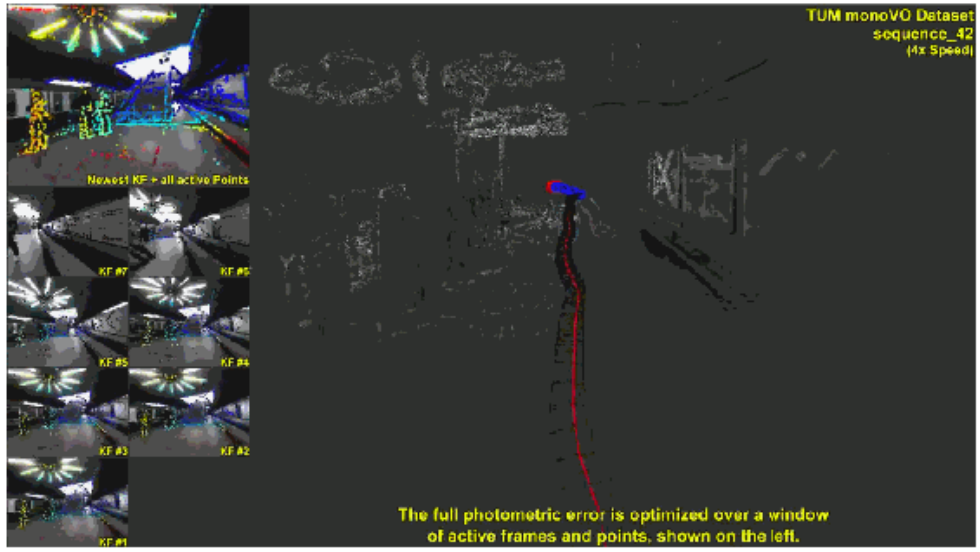
Outline

- Background & Motivation
- REACT3D
- RTGS
- Summary

Background

- **What is SLAM?**

  Simultaneous Localization and Mapping (SLAM) aims to estimate a camera's (or robot's) pose while reconstructing the surrounding environment.



Background

- **Why use 3DGS for SLAM?**
  - Unified representation for tracking, mapping, and rendering.
  - Differentiable structure for joint pose and scene optimization.
  - Supports dense, **high-quality reconstructions** at arbitrary resolutions.
  - **Continuous, smooth geometry** without voxel discretization.
-

Fig. 12: **SLAM Methods Comparison on the Replica [74] Dataset– Image Rendering.** Images sourced from [12].

Background

- **How to use 3DGS for SLAM?**
-

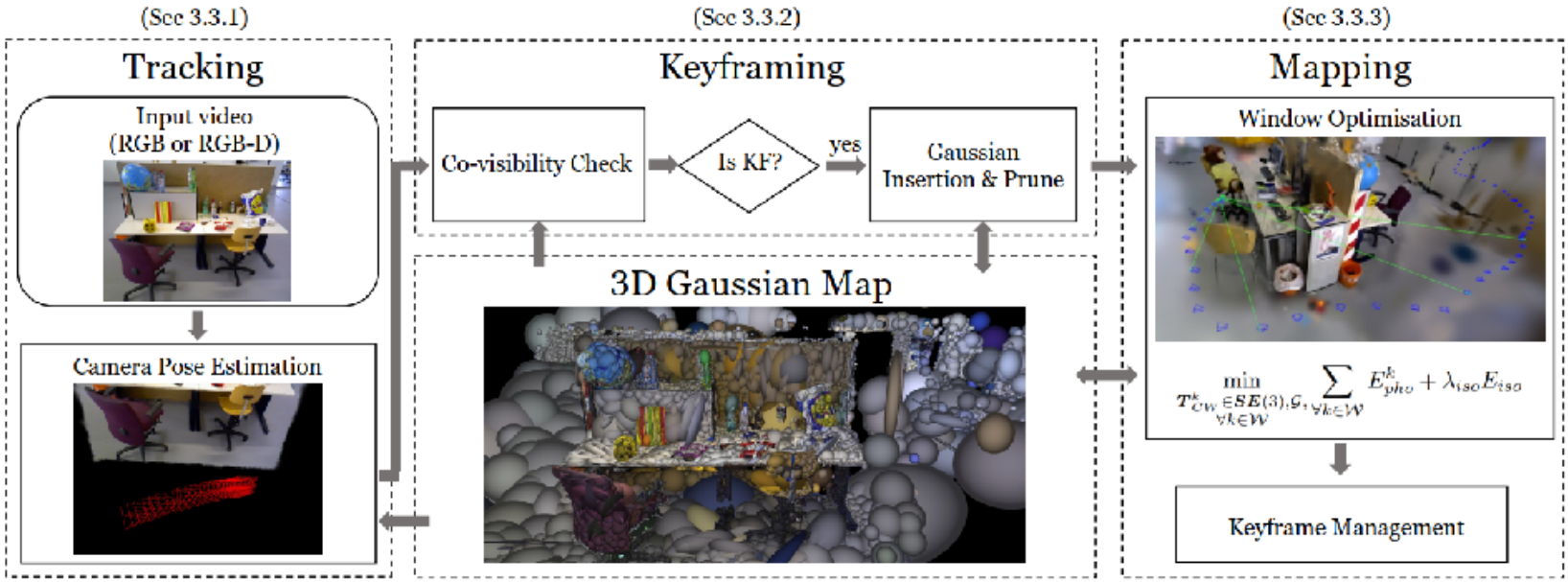


Figure 2. **SLAM System Overview:** Our SLAM system uses 3D Gaussians as the only representation, unifying all components of SLAM, including tracking, mapping, keyframe management, and novel view synthesis.

Background

- However, the runtime performance of existing 3DGS–SLAM solutions remains significantly **below** the threshold required for real–time SLAM applications (⩾ 30 FPS)

.

Table 2: Performance comparison of different SLAM algorithms on ONX edge GPU [1] using Repilica dataset [41], where Absolute Trajectory Error (ATE) measures tracking accuracy (lower is better) and Peak Signal-to-Noise Ratio (PSNR) reflects rendering fidelity (higher is better).

Algorithm	Accuracy Performance		Speed Performance		Storage Efficiency	Dataset
	ATE (cm) ↓	PSNR (dB) ↑	Tracking FPS ↑	Overall FPS[1] ↑	Peak Gaussian Mem. Capacity ↓	Monocular Support
SplaTAM [14]	Medium (0.36-2.25)	High (25.12-34.11)	Slow (0.26-0.46)	Slow (0.42-0.78)	Inefficient (7-10 GB)	✗
GS-SLAM [51]	Low (0.5-3.7)	High (21.6-31.56)	Moderate (1.45-2.37)	Moderate (1.45-2.34)	Inefficient (8-12 GB)	✗
MonoGS [30]	High (0.32-1.58)	High (25.82-34.83)	Moderate (0.81-1.32)	Moderate (0.83-1.3)	Inefficient (13-15 GB)	✓
Photo-SLAM [13]	Low (0.53-2.8)	High (20.12-31.97)	Fast[2] (11.7-14.3)	Fast[2] (8.3-9.4)	Acceptable (4-5 GB)	✓

[1]Overall FPS includes both tracking and mapping iterations.
[2]Photo-SLAM uses feature point matching and thus achieves higher throughput.

Background

- Why ?
  - Tracking and Mapping are the primary bottlenecks. **Rendering and Rendering BP** dominate the cost of both tracking and mapping stages.
  - Redundant computation across **frames** and **iterations**.

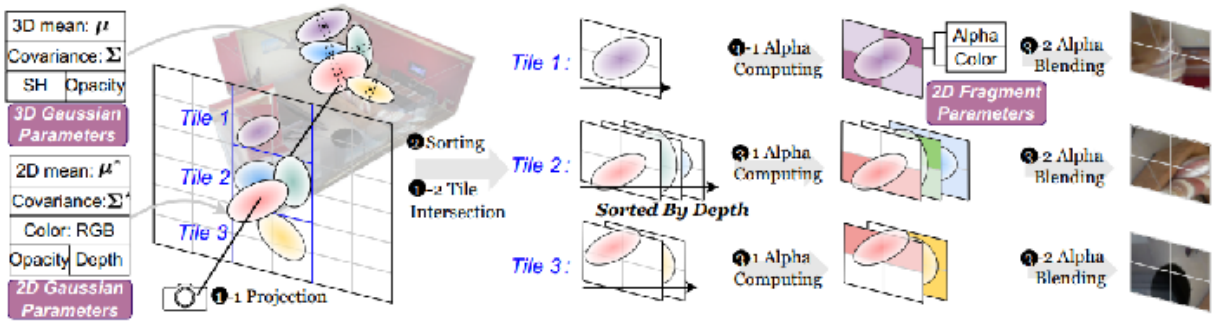


Figure 1: Rendering pipeline: (a) Projecting 3D Gaussians into 2D Gaussians. (b) Sorting 2D Gaussians by depth. (c) Calculating the influence of each Gaussian on the pixels.

REACT3D: Real-time Edge Accelerator for Incremental Training in 3D Gaussian Splatting based SLAM Systems

Hongyi Wang*
Tsinghua University
Beijing, China
The Hong Kong University of Science and Technology
Hong Kong SAR, China
hwangid@connect.ust.hk

Zhenhua Zhu*†
Tsinghua University
Beijing, China
The Hong Kong University of Science and Technology
Hong Kong SAR, China
zhuzhenhua@mail.tsinghua.edu.cn

Tianchen Zhao
Tsinghua University
Beijing, China
ztc23@mails.tsinghua.edu.cn

Yunfei Xiang
Tsinghua University
Beijing, China
xiang-yf22@mails.tsinghua.edu.cn

Zehao Wang
Tsinghua University
Beijing, China
wangzeha24@mails.tsinghua.edu.cn

Jincheng Yu
Tsinghua University
Beijing, China
yu-jc@mail.tsinghua.edu.cn

Huazhong Yang
Tsinghua University
Beijing, China
yanghz@mail.tsinghua.edu.cn

Yuan Xie
The Hong Kong University of Science and Technology
Hong Kong SAR, China
yuanxie@ust.hk

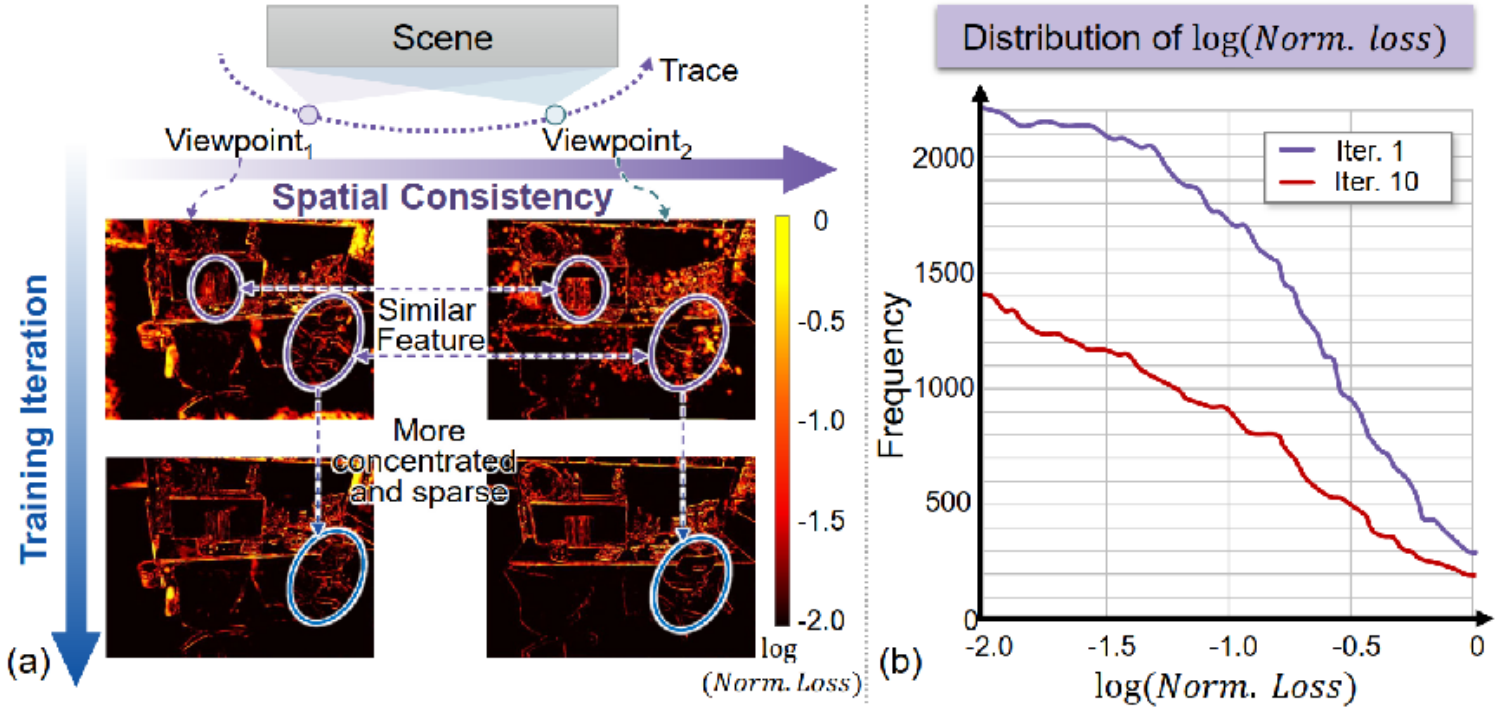
Yu Wang†
Tsinghua University
Beijing, China
yu-wang@mail.tsinghua.edu.cn

Overview:

- Design a real-time edge accelerator equipped with **sparsification mapping algorithm** and Dual-index Gaussian Buffer to resolve **discontinuous memory accesses**.
- 12.10× performance and 33.69× energy efficiency improvement over NVIDIA Jetson AGX

REACT3D: Key Insights

- During training, the loss map exhibits **strong spatial consistency across frames** and **convergence across iterations.**



REACT3D: Sparsification Mapping

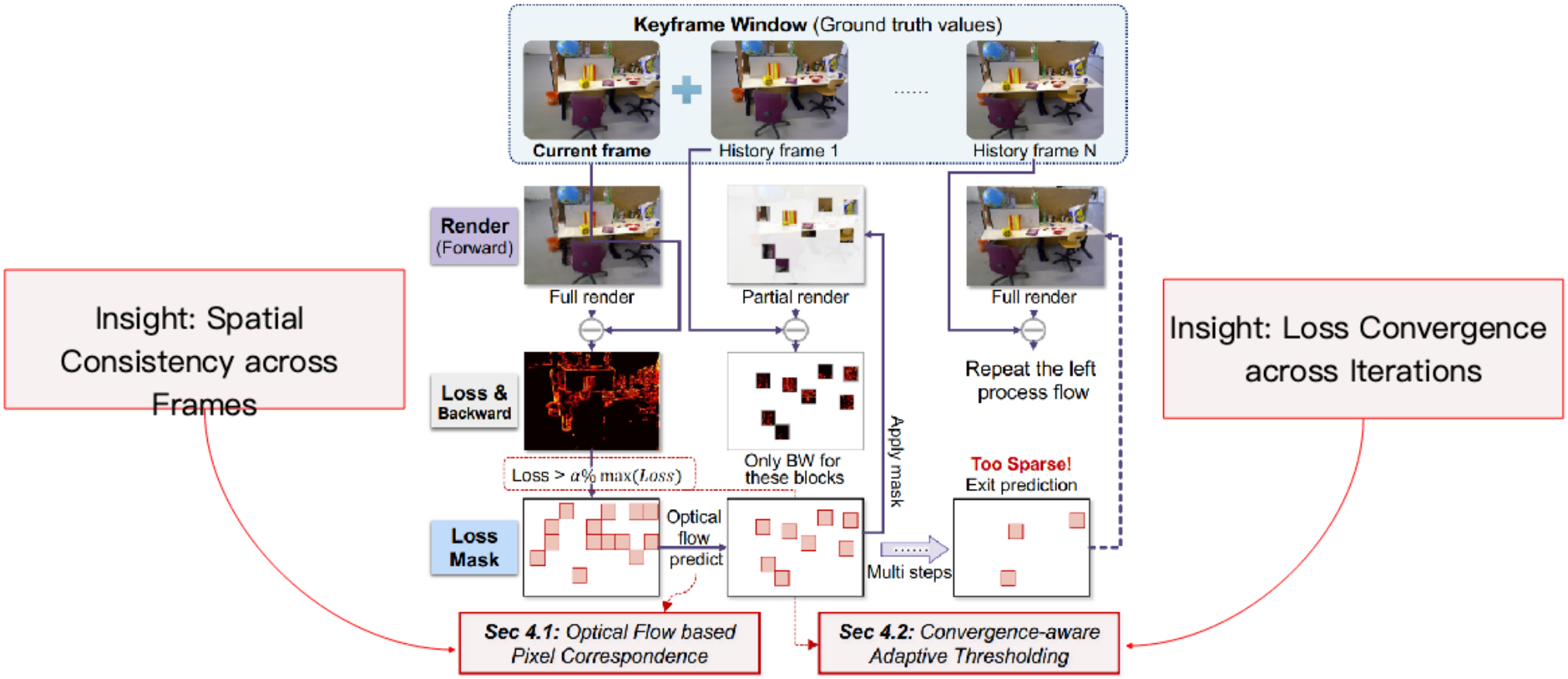


Figure 6: The proposed sparse 3DGS SLAM training pipeline.

REACT3D: Key Insights

- Discarding the D-SSIM loss term provides the opportunity to **eliminate the explicit loss computing** by leveraging **pixelwise gradient independence**.

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}. \longrightarrow \mathcal{L} = \frac{1}{3N}\sum_{i=1}^{N}\sum_{c\in\{R,G,B\}}\left|\hat{I}_{i,c} - I_{i,c}\right| + \lambda\sum_{i=1}^{|\mathcal{G}|}\|s_i - \tilde{s}_i \cdot \mathbf{1}\|_1 .$$

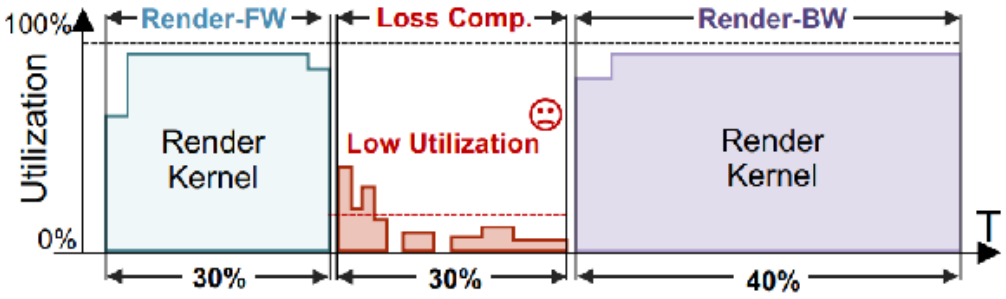
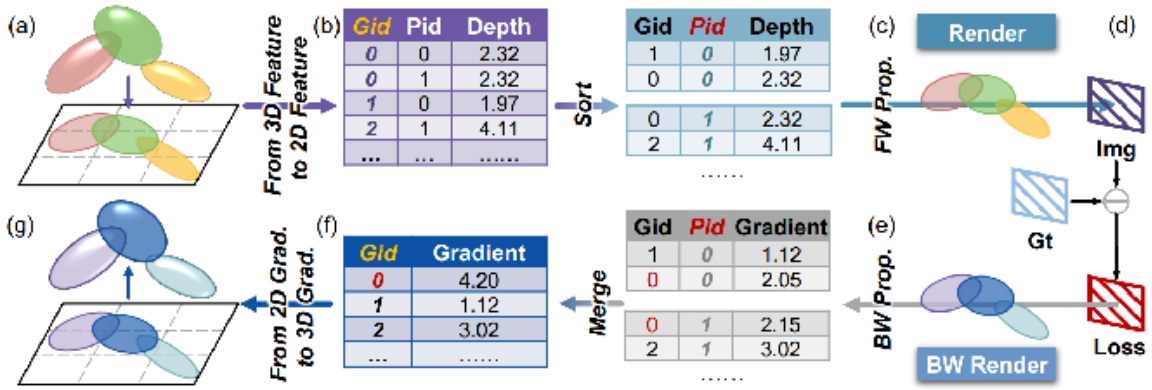


Figure 4: GPU execution timeline from Nsight Systems.

$$\frac{\partial\mathcal{L}}{\partial\hat{I}_{i,c}} = \begin{cases} \frac{1}{3N}, & \text{if } \hat{I}_{i,c} - I_{i,c} > 0 \\ -\frac{1}{3N}, & \text{if } \hat{I}_{i,c} - I_{i,c} < 0 \\ 0, & \text{if } \hat{I}_{i,c} = I_{i,c} \end{cases}$$

REACT3D: Key Insights

- Dual-index access introduces a **structural mismatch** between the data layout and memory access behavior.
  - Data are initially stored in memory by **the primary index**, but accesses are dictated by **the secondary index**.
  - Naive re-sorting by **the secondary index** causes high latency due to irregular mapping.
  - **Solution**: The hardware should simultaneously support **primary-index-based** data layout and **secondary-index-based** data access.



REACT3D: CAM–based Dual–index Gaussian Buffer

- Multiple banks, each with CAM + SRAM + **shared** auxiliary buffer.
- CAM stores Gid/Pid; SRAM stores Pid/Gid & features.
- Auxiliary buffer tracks active Pid for **content–based** lookups.

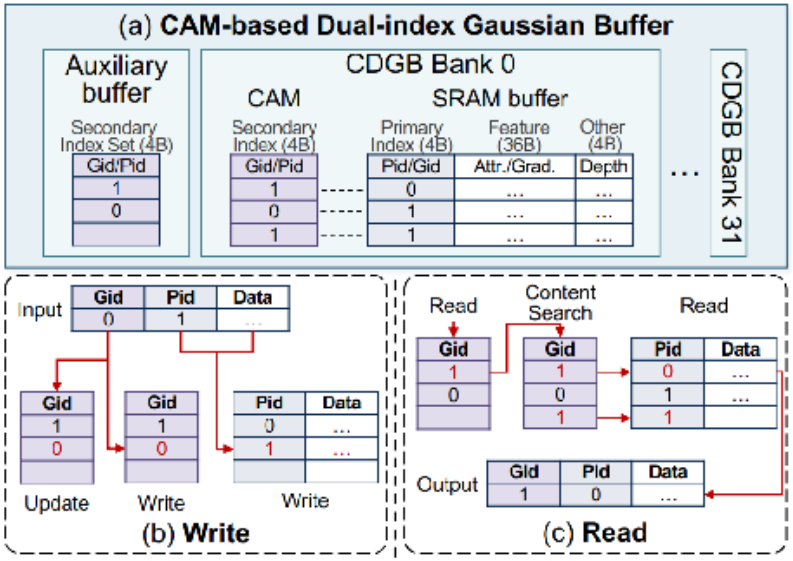


Figure 11: Details of CDGB design. (a) CDGB architecture. (b) Content-based write. (c) Content-based read.

Stage	Primary index	Secondary index
Sorting	Gid	Pid
Gradient merging	Pid	Gid

REACT3D: Evaluation

- **Experiment Setup:**
  - Baseline: NVIDIA Jetson AGX Orin
  - Model: MonoGS
- **Results:**
  - With an initial 80% sparsity threshold, their **sparse kernel** achieves average speedups of **3.73 × and 4.24×** in the FW and BW rendering stages.
  - The accelerator achieves **12.10×** speedup and occupies **12.45 mm2** area and consumes **3.55 W** typical power consumption.
- **Takeaways:**
  - The **sparsification method** achieves average speedups of 2.25× in forward rendering, 4.66× in backward rendering, and 6.20× in gradient merging, which is the main reason for speedup.
  - **Efficient memory access** can be achieved through architecture–level designs such as index–aware caching.

Table 2: Area and power breakdown of REACT3D.

Module	ConFigure	Area (mm^2)	Power (W)
GE	4 Engines	0.96 (7.7%)	0.29 (8.2%)
Sorting Unit	4 Engines	0.07 (0.6%)	0.07 (2.0%)
FE	8×(4×4) Engines	2.35 (18.9%)	0.61 (17.2%)
BE	8×(4×4) Engines	7.38 (59.3%)	1.82 (51.3%)
CDBG	32 Banks	1.66 (13.3%)	0.74 (20.9%)
-Buffer	800KB	0.86 (6.9%)	0.42 (11.9%)
-CAM	64KB	0.80 (6.4%)	0.32 (9.0%)
ILB	32KB	0.03 (0.2%)	0.02 (0.6%)
Total		12.45 (100%)	3.55 (100%)

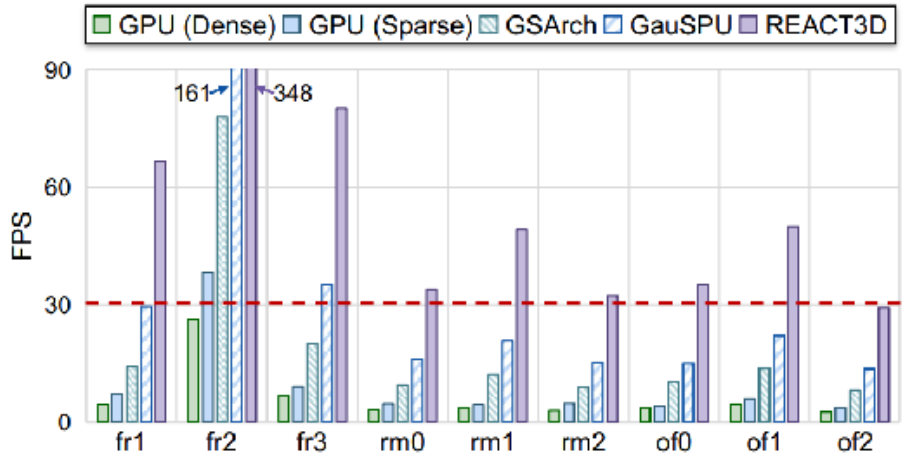


Figure 13: End-to-end mapping throughput comparison across different hardware platforms.

REACT3D: Evaluation

- **Experiment Setup:**
  - Baseline: NVIDIA Jetson AGX Orin
  - Model: MonoGS
- **Results:**
  - With an initial 80% sparsity threshold, their **sparse kernel** achieves average speedups of **3.73 × and 4.24×** in the FW and BW rendering stages.
  - The accelerator achieves **12.10×** speedup and occupies **12.45 mm2** area and consumes **3.55 W** typical power consumption.
- **Takeaway:**
  - The **sparsification method** achieves average speedups of 2.25× in forward rendering, 4.66× in backward rendering, and 6.20× in gradient merging, which is the **main reason** for speedup.
  - **Efficient memory access** can be achieved through architecture–level designs such as index–aware caching.

Table 2: Area and power breakdown of REACT3D.

Module	ConFigure	Area (mm^2)	Power (W)
GE	4 Engines	0.96 (7.7%)	0.29 (8.2%)
Sorting Unit	4 Engines	0.07 (0.6%)	0.07 (2.0%)
FE	8×(4×4) Engines	2.35 (18.9%)	0.61 (17.2%)
BE	8×(4×4) Engines	7.38 (59.3%)	1.82 (51.3%)
CDBG	32 Banks	1.66 (13.3%)	0.74 (20.9%)
-Buffer	800KB	0.86 (6.9%)	0.42 (11.9%)
-CAM	64KB	0.80 (6.4%)	0.32 (9.0%)
ILB	32KB	0.03 (0.2%)	0.02 (0.6%)
Total		12.45 (100%)	3.55 (100%)

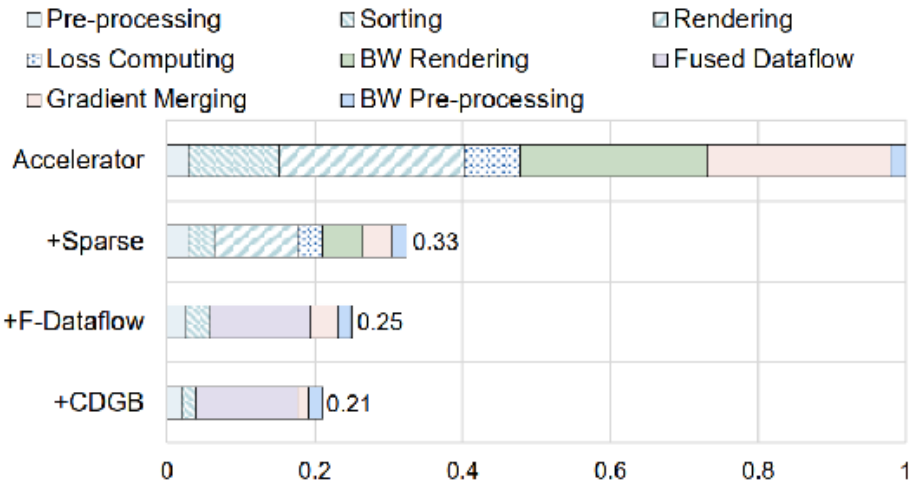


Figure 18: Execution time breakdown on REACT3D with incremental optimizations.

RTGS: Real-Time 3D Gaussian Splatting SLAM via Multi-Level Redundancy Reduction

Leshu Li[*1], Jiayin Qin[*1], Jie Peng[2], Zishen Wan[3], Huaizhi Qu[2], Ye Han[1], Pingqing Zheng[1], Hongsen Zhang[1], Yu (Kevin) Cao[1], Tianlong Chen[2], Yang (Katie) Zhao[1]

[1]Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA

[2]Department of Computer Science, University of North Carolina at Chapel Hill, USA

[3]Department of Electrical and Computer Engineering, Georgia Institute of Technology, USA

{li003385,qin00162,yangzhao}@umn.edu

RTGS: Algorithm

- Insights & Algorithm
-

Insights	Proposed Algorithm
Most of Gaussians contributes negligibly to camera pose	Adaptive Gaussian Pruning
Redundant computation across all frames	Dynamic Downsampling Rendering Resolution

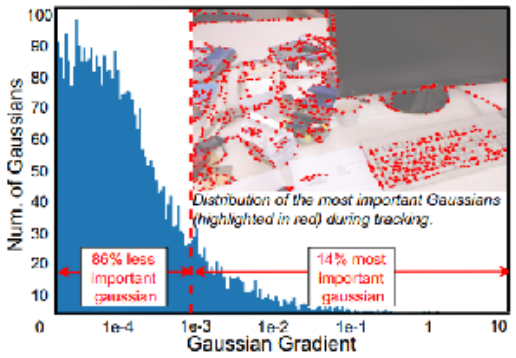


Figure 4: Gaussian gradient distribution during tracking.

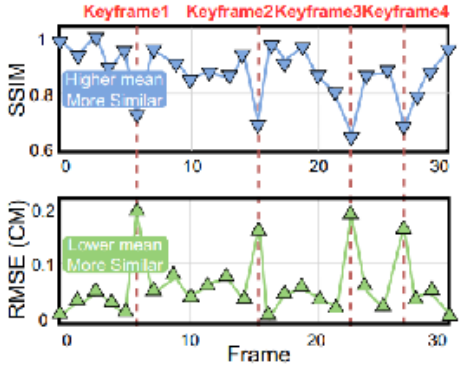


Figure 5: Similarity analysis in consecutive frames.

RTGS: Architecture

- Overview
  - Original GPU accelerates preprocessing and sorting step.
  - A RTGS plug–in for other steps.

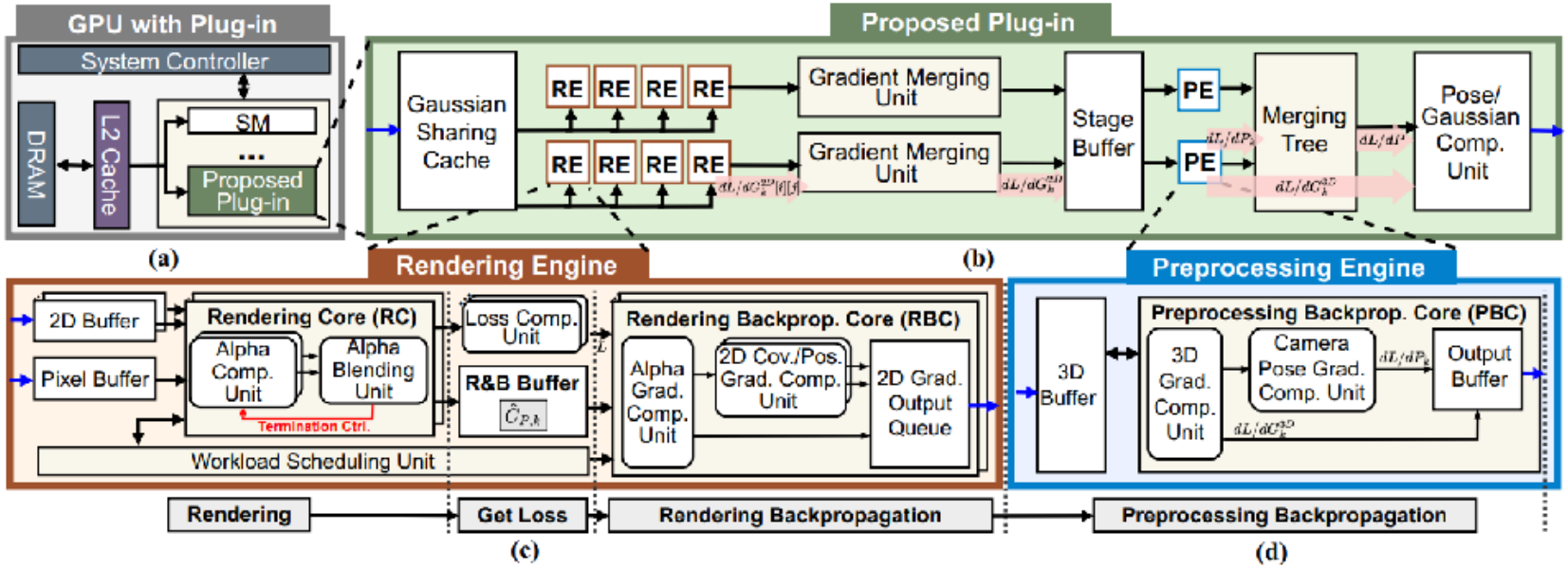


Figure 7: Overall design of our acceleration system: (a) an illustration of the integration of RTGS plug-in with the GPU. RTGS plug-in shares the L2 cache with the GPU, (b) the overview architecture of RTGS; (c) the block diagram of the Rendering Engine (RE), responsible for Step ❸ Rendering, loss, and Step ❹ Rendering BP; and (d) the block diagram of the Preprocessing Engine (PE), responsible for Step ❺ Preprocessing BP.

RTGS: Architecture

- Mitigate pipeline imbalance through resource reallocation:
  - Each pixel is assigned to one alpha computing unit (12 cycles).
  - Two pixels **share** a single alpha blending unit (3 cycles).

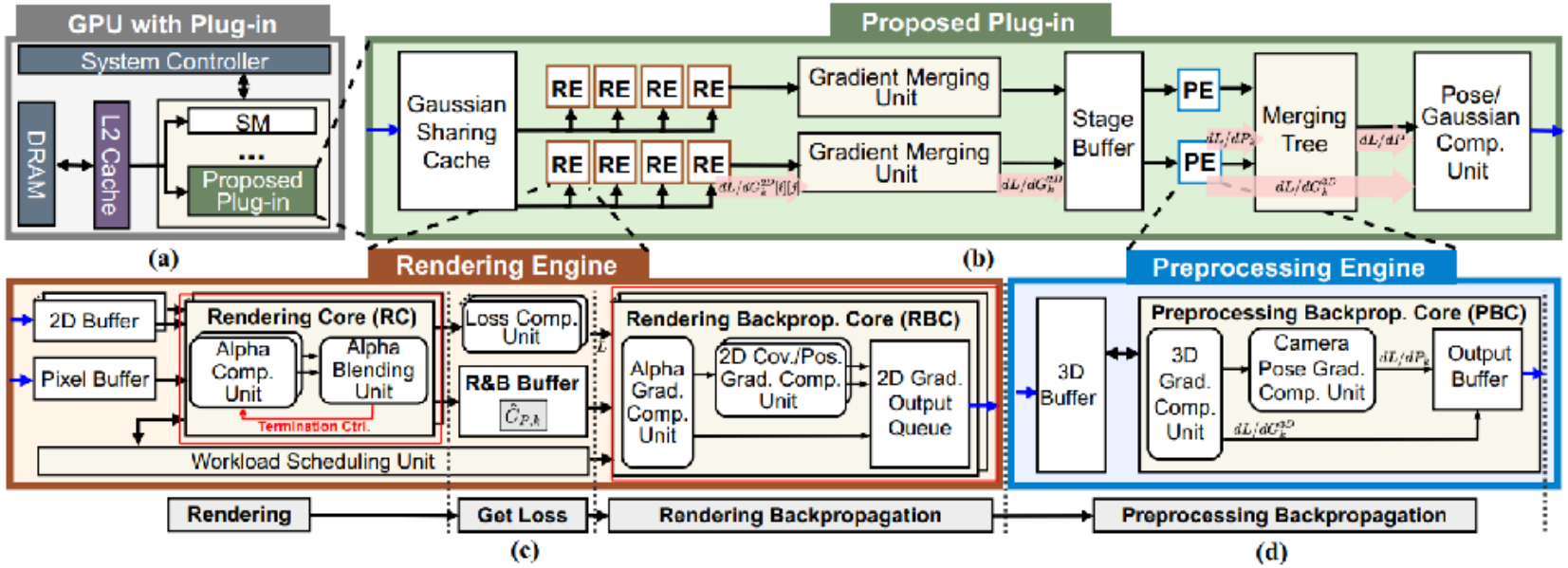


Figure 7: Overall design of our acceleration system: (a) an illustration of the integration of RTGS plug-in with the GPU. RTGS plug-in shares the L2 cache with the GPU, (b) the overview architecture of RTGS; (c) the block diagram of the Rendering Engine (RE), responsible for Step ❸ Rendering, loss, and Step ❹ Rendering BP; and (d) the block diagram of the Preprocessing Engine (PE), responsible for Step ❺ Preprocessing BP.

RTGS: Architecture

- Use a **double buffer** to enable parameter reuse:
  alpha gradient computation: 20 cycles –> 4 cycles

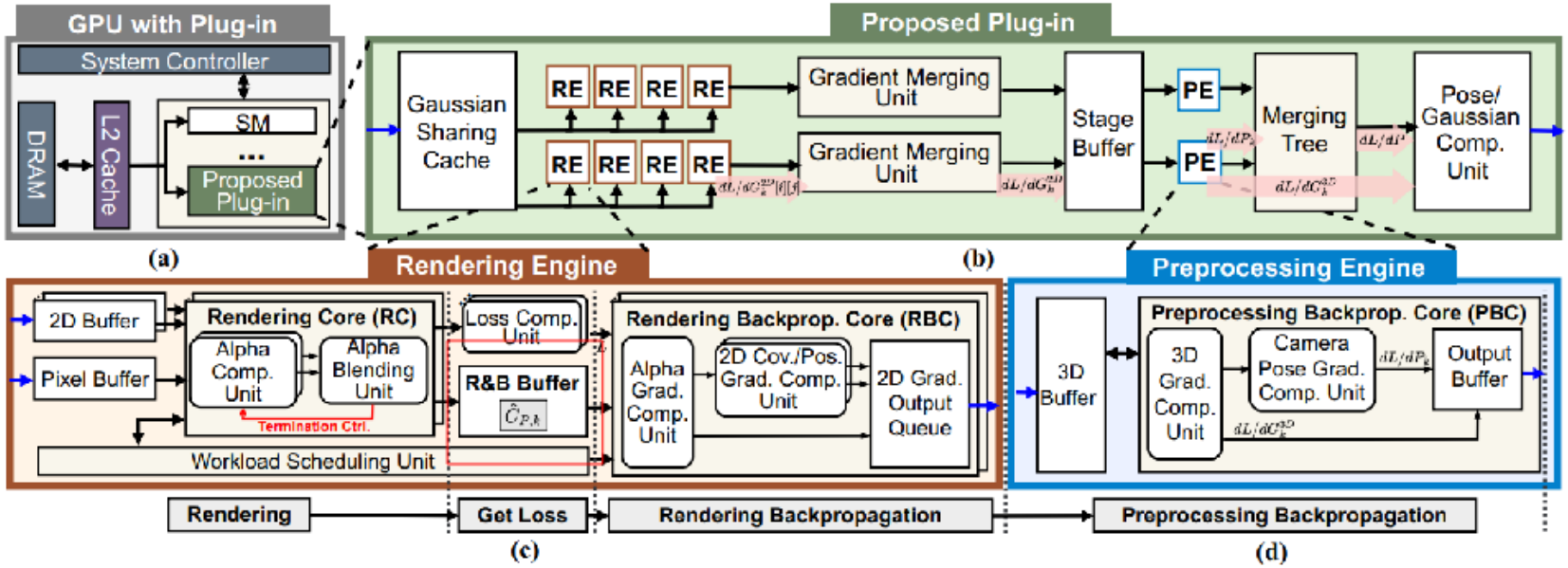


Figure 7: Overall design of our acceleration system: (a) an illustration of the integration of RTGS plug-in with the GPU. RTGS plug-in shares the L2 cache with the GPU, (b) the overview architecture of RTGS; (c) the block diagram of the Rendering Engine (RE), responsible for Step ❸ Rendering, loss, and Step ❹ Rendering BP; and (d) the block diagram of the Preprocessing Engine (PE), responsible for Step ❺ Preprocessing BP.

RTGS: Architecture

- Integration with GPUs:
  - RTGS first polls an **Input_done** flag to detect when SMs finish preprocessing and sorting, then sets a **gradient_ready** flag to notify SMs to start pruning.
  - ...

RTGS: Evaluation

- Hardware baselines: ONX, RTX3090, GauSPU
- Architecture configurations:

Technology Node	28nm	Operating Freq.	500 MHz
Power	8.11W	Area	28.41mm^2
Computation Resources			
RE × 16: 8 RCs & RBCs per RE	WSU × 16	PE × 16: 1 PBC per PE	GMU × 4
Memory Allocation			
Gaussian Cache	80KB	Pixel Buffer	24KB
2D Buffer	20KB	R&B Buffer	16KB
Stage Buffer	16KB	3D Buffer	10KB
Output Buffer	15KB	WSU Buffer	16KB
SRAM	197KB	L2 Cache	2MB

Get by Verilog

Table 5: Comparison of device specifications.

Device	Technology	SRAM	Number of Cores	Area [mm^2]	Power [W]
ONX [1]	8 nm	4 MB	512 CUDA Cores	450	15
RTX 3090 [34]	8 nm	80.25 MB	5248 CUDA Cores	628	352
GauSPU [49]	12 nm	560 KB	128 REs/32 BEs	30	9.4
RTGS	28 nm	197 KB	16 REs/16 PEs	28.41	8.11
RTGS-12nm[1]	12 nm	197 KB	16 REs/16 PEs	6.49	4.63
RTGS-8nm[1]	8 nm	197 KB	16 REs/16 PEs	2.40	3.76

[1] The 12nm and 8nm data are scaled form the *DeepScaleTool* [37] with a voltage of 0.8V and a frequency of 500MHz.

RTGS: Evaluation

- Performance comparison
- Ablation study of performance breakdown

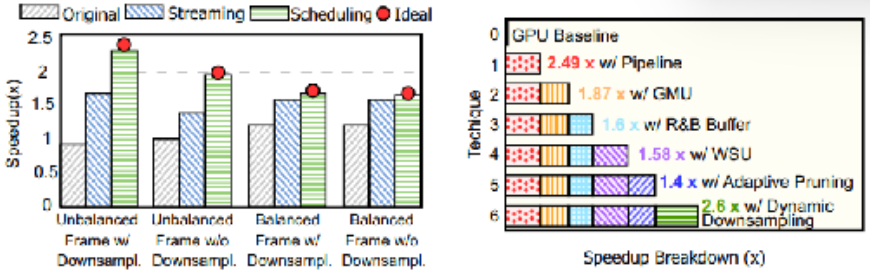


Figure 17: Ablation study of the performance breakdown for (a) two techniques for mitigating workload imbalance and (b) all RTGS techniques on Replica [41] with MonoGS [30] baseline.

Table 6: Performance comparison of 3DGS-SLAM variants across four datasets.

Method	TUM [42]				Replica [41]				ScanNet [4]				ScanNet++ [54]			
	ATE (cm)	PSNR (dB)	FPS	Mem (GB)	ATE (cm)	PSNR (dB)	FPS	Mem (GB)	ATE (cm)	PSNR (dB)	FPS	Mem (GB)	ATE (cm)	PSNR (dB)	FPS	Mem (GB)
GS-SLAM [51]	3.7	15.93	3.3	8.3	0.5	35.41	2.3	9.2	2.85	19.87	1.4	10.4	3.21	24.41	0.92	11.1
Taming 3DGS+GS-SLAM	6.7	14.31	4.7	4.2	3.2	30.3	3.2	4.6	5.6	14.3	1.9	1.9	6.2	17.71	1.3	5.6
Ours+GS-SLAM	3.4	16.01	12.1	3.9	0.51	35.44	8.3	4.3	2.76	21.75	5.1	4.9	3.19	25.13	3.3	5.2
MonoGS [30]	1.47	25.82	1.8	13.1	0.32	38.94	1.2	13.5	3.25	20.43	0.7	14.6	7.46	23.79	0.6	15.0
Taming 3DGS+MonoGS	3.21	20.28	2.6	6.9	0.43	32.51	1.9	7.1	4.33	17.26	1.1	7.6	9.81	20.15	0.8	7.8
Ours+MonoGS	1.41	25.73	4.7	6.2	0.29	39.14	3.6	6.4	3.26	20.44	2.8	6.1	6.76	23.6	1.6	7.1
Photo-SLAM [13]	2.61	20.12	8.1	4.3	0.64	31.97	8.4	7.1	3.73	21.33	6.2	6.3	6.43	25.31	6.2	4.9
Taming 3DGS+Photo-SLAM	4.21	19.23	11.3	2.2	1.23	27.66	11.1	2.6	4.1	20.33	8.8	8.8	6.99	23.12	8.9	6.4
Ours+Photo-SLAM	2.33	21.34	12.96	2.0	0.61	31.9	11.1	2.3	3.68	22.45	10.2	2.2	6.33	26.54	9.92	2.3

ONX

RTGS: Evaluation

- The FPS over the accelerator and baseline GPUs.

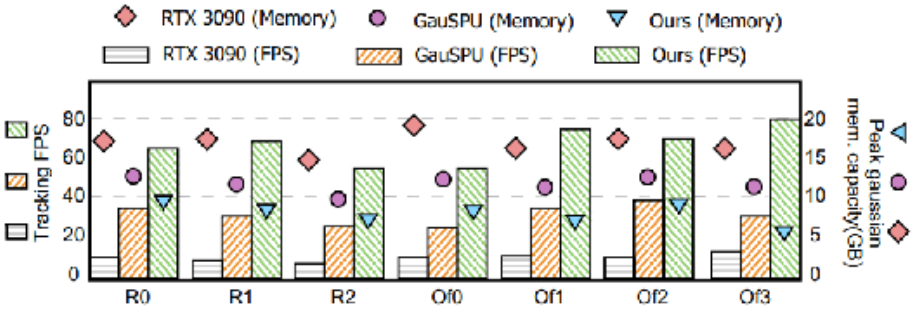


Figure 16: Comparison of RTX 3090 [34], GauSPU [49], and proposed RTGS, using SplaTAM [14] algorithm on Replica [41] dataset: (a) tracking FPS and (b) memory efficiency.

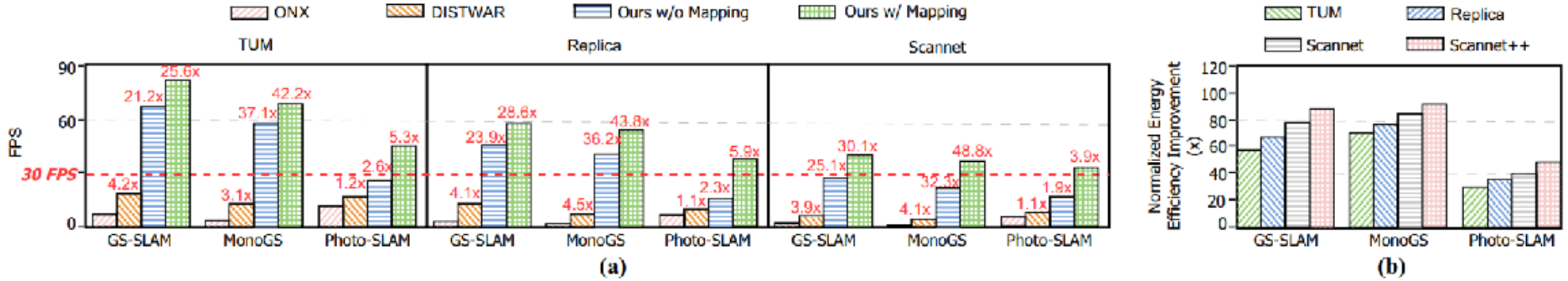


Figure 15: The FPS on the proposed RTGS and the baseline GPU. (a) Comparison of FPS across four baseline algorithms on three datasets using four configurations: ONX edge GPU [1], RTGS with tracking acceleration only, and RTGS with both tracking and mapping acceleration. (b) Improvement in energy efficiency across the three baseline algorithms on four datasets.

Summary

- 3DGS–SLAM achieves high–quality scene reconstructions, but its framework suffers from **computational bottlenecks** (rendering is compute–bound, backward involve costly atomic operations, …). Moreover, **redundant computations** across iterations and frames hinder real–time performance.
- **Algorithm–level optimizations** can significantly accelerate the pipeline by reducing redundant computations, though they may introduce minor degradation in reconstruction quality.
- **Architecture–level optimizations** (such as data reuse via caching, balanced workload distribution, and buffer–based DRAM aggregation) can improve efficiency without sacrificing any accuracy. However, their speedup is generally **smaller** compared to algorithmic acceleration.

Thanks!