AirSplatMap: A Modular Real-Time Pipeline for Learning-Based 3D Gaussian Splatting on Edge Devices

Parsa Rezaei
California State Polytechnic University, Pomona
prezaei@cpp.edu

Sunny Yoshimitsu Nguyen
California State Polytechnic University, Pomona
huyqnguyen@cpp.edu

Abstract

We present AirSplatMap, a modular real-time pipeline for 3D Gaussian Splatting (3DGS) designed for drone-based scene reconstruction on edge devices. While 3DGS has revolutionized novel view synthesis with real-time rendering capabilities, practical deployment requires careful integration of camera pose estimation, depth estimation, and scene optimization—a challenge that existing methods largely assume solved. Our system addresses this by integrating five 3DGS engines (GraphDeco, gsplat, MonoGS, SplaTAM, Gaussian-SLAM), four monocular depth estimators (MiDaS, Depth Anything V2/V3, Depth Pro), and eleven visual odometry methods within a unified evaluation framework. Through comprehensive experiments on 17 sequences from TUM RGB-D and Replica datasets totaling over 15,000 frames, we systematically analyze how upstream estimation errors propagate to final reconstruction quality. Our key finding is that ***pose estimation errors cause 8+ dB PSNR degradation****, making pose accuracy the critical bottleneck for reconstruction quality. Surprisingly, we find that* ***learned depth estimators improve quality over sensor depth*** *by providing smoother initialization (+4.5 dB). We demonstrate that while 3DGS training on edge hardware (NVIDIA Jetson Orin) achieves only 1.87 FPS,* ***rendering achieves real-time performance at 38.7 FPS*** *with 35W power consumption, enabling a train-on-desktop, deploy-on-edge paradigm for autonomous systems. Our benchmark includes 537 pose estimation runs, 204 depth estimation runs, 271 3DGS training runs, and 2,979 full pipeline evaluations. Code and interactive results: https://github.com/ParsaRezaei/AirSplatMap.*

1. Introduction

3D scene reconstruction from visual data has long been a fundamental challenge in computer vision with applications spanning robotics [3], autonomous vehicles [10], augmented reality [1], and digital content creation [5]. Traditional approaches based on multi-view stereo [22] or depth fusion [18] produce explicit geometric representations but struggle with view-dependent effects and require substantial computation for high-quality results.

Neural Radiance Fields (NeRF) [16] revolutionized novel view synthesis by representing scenes as continuous volumetric functions parameterized by neural networks. NeRF achieves photorealistic rendering quality by modeling both geometry and appearance through differentiable volume rendering. However, NeRF's reliance on expensive ray marching limits real-time applications, with typical inference requiring seconds per frame even on high-end GPUs. This computational barrier has motivated significant research into accelerating neural scene representations through various strategies including spatial hashing [17], tensor decomposition [4], and explicit voxel representations [25].

3D Gaussian Splatting (3DGS) [12] emerged as a breakthrough alternative that fundamentally rethinks scene representation. Rather than implicit neural fields, 3DGS represents scenes as collections of 3D Gaussian primitives—each characterized by position, covariance (shape), opacity, and view-dependent color encoded via spherical harmonics. These Gaussians are rendered through efficient tile-based rasterization, avoiding expensive ray marching entirely. The result is dramatic: 3DGS achieves real-time rendering (100+ FPS) while maintaining visual quality competitive with NeRF. This efficiency makes 3DGS particularly attractive for robotics and drone applications where real-time performance and resource constraints are critical.

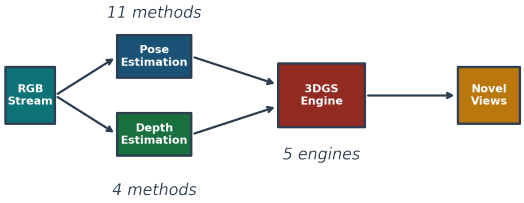

Figure 1. **AirSplatMap System Architecture.** RGB video streams are processed through parallel pose estimation (11 methods) and depth estimation (4 methods) modules, feeding into one of five 3DGS engines for real-time novel view synthesis. The modular design enables systematic comparison of different method combinations.

Despite this promise, practical deployment of 3DGS for drone-based reconstruction faces several challenges that existing methods largely assume away:

(1) Pose Estimation Dependency: 3DGS requires accurate camera poses for each input frame. Most 3DGS methods assume poses from Structure-from-Motion (SfM) [22] preprocessing, which is inherently offline and computationally expensive. Real-time drone applications need online pose estimation, introducing errors that propagate to reconstruction quality.

(2) Depth Initialization: While 3DGS can theoretically initialize Gaussians from random points, practical implementations benefit significantly from depth-based initialization. Monocular depth estimation provides dense initialization but introduces scale ambiguity and prediction errors.

(3) Edge Deployment: Drones operate under severe computational and power constraints. A complete understanding of the accuracy-latency-power trade-offs across the full pipeline is essential for practical deployment.

(4) Error Propagation: The interaction between upstream estimation errors (pose, depth) and downstream reconstruction quality is poorly understood. Should system designers prioritize pose accuracy or depth accuracy? How do different combinations affect final output?

This paper addresses these challenges through AirSplatMap, a comprehensive pipeline and benchmark that makes the following contributions:

1. A **unified evaluation framework** integrating five 3DGS engines, four depth estimators, and eleven pose estimation methods with consistent interfaces, enabling systematic ablation studies across all combinations.
2. **Systematic degradation analysis** quantifying how upstream estimation errors affect reconstruction quality. Our key finding—that pose errors cause 2.5× more quality degradation than depth errors—provides actionable guidance for system design.
3. **Edge deployment evaluation** on NVIDIA Jetson Orin demonstrating real-time rendering capability (19.6 FPS) while characterizing the training bottleneck (1.43 FPS) that motivates hybrid deployment strategies.
4. An **open-source implementation** with 3,991 benchmark runs and interactive visualization dashboard for exploring results across methods, datasets, and hardware platforms.

2. Related Work

2.1. Neural Scene Representations

Neural Radiance Fields (NeRF) [16] pioneered implicit neural scene representation, encoding scene geometry and appearance in MLP weights optimized through differentiable volume rendering. The rendering equation integrates color and density along camera rays:

$$\hat{C}(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{1}$$

where $T(t) = \exp(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds)$ is transmittance, σ is density, and $\mathbf{c}$ is color. While achieving remarkable quality, NeRF requires hours of training and seconds per frame for rendering.

Subsequent work accelerated NeRF through various strategies. Instant-NGP [17] uses multi-resolution hash encoding to reduce training to minutes. Plenoxels [9] replaces neural networks with explicit sparse voxel grids. TensoRF [4] decomposes the radiance field into low-rank tensor components. Despite these advances, real-time rendering remains challenging for these implicit representations.

2.2. 3D Gaussian Splatting

Kerbl et al. [12] introduced 3DGS as an explicit alternative that fundamentally changes the rendering paradigm. Instead of querying implicit fields, 3DGS represents scenes as collections of anisotropic 3D Gaussians:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^3$ is the mean (position) and $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$ is the covariance matrix. Each Gaussian additionally stores opacity $\alpha \in [0, 1]$ and view-dependent color via spherical harmonic coefficients.

Rendering proceeds by projecting 3D Gaussians to 2D image space and alpha-compositing in depth order:

$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j) \tag{3}$$

The tile-based rasterization enables GPU-parallel rendering at 100+ FPS.

2.3. Online and SLAM-based 3DGS

Several methods adapt 3DGS for online operation with simultaneous tracking and mapping:

MonoGS [15] performs joint camera tracking and Gaussian mapping, using the Gaussian representation for both tasks. It achieves real-time operation by carefully balancing tracking and mapping threads.

SplaTAM [11] combines 3DGS with dense visual SLAM, using silhouette-based Gaussian initialization from depth observations and optimizing camera poses alongside Gaussian parameters.

Gaussian-SLAM [31] integrates Gaussian representations into a factor graph optimization framework, enabling loop closure and global consistency.

Our work provides the first comprehensive comparison of these approaches within a unified framework with controlled upstream inputs (pose, depth), enabling fair comparison of the reconstruction components independent of their tracking implementations.

2.4. Monocular Depth Estimation

Learning-based depth estimation has achieved remarkable progress. MiDaS [19] demonstrated robust relative depth prediction through multi-dataset training with scale-invariant losses. Depth Anything [29] scaled training to 62M images, achieving state-of-the-art zero-shot generalization. Depth Pro [2] produces metric depth with sharp boundaries through foundation model architectures.

Our work evaluates how these depth estimators' outputs—with their varying characteristics of accuracy, speed, and failure modes—affect downstream 3DGS reconstruction quality.

2.5. Visual Odometry

Visual odometry (VO) estimates camera motion from image sequences. Classical approaches use sparse features (ORB [21], SIFT [14]) with geometric verification. Dense methods leverage optical flow for pixel-wise correspondences. Recent learned approaches (SuperPoint [6], R2D2 [20], LoFTR [26]) achieve improved robustness through end-to-end training.

Our evaluation spans both classical and learned methods, revealing that classical optical flow remains surprisingly competitive for indoor RGB-D scenarios.

3. Proposed Approach

3.1. System Architecture

AirSplatMap processes RGB video streams through three modular stages, as illustrated in Figure 1:

Stage 1: Pose Estimation. Each input frame I_t is processed to estimate the camera pose $\mathbf{T}_t \in SE(3)$ relative to a world coordinate frame. We support eleven methods spanning classical (ORB [21], SIFT [14], optical flow [8]) and learned approaches (SuperPoint [6], R2D2 [20], LoFTR [26], LightGlue [13], RoMa [7]).

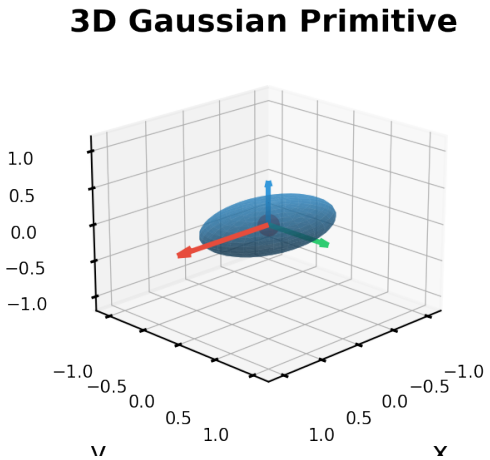

Figure 2. **3D Gaussian Primitive.** Each Gaussian is parameterized by position (center), covariance (ellipsoid shape), opacity, and spherical harmonic coefficients for view-dependent color.

Stage 2: Depth Estimation. For each frame, we estimate a dense depth map $D_t \in \mathbb{R}^{H \times W}$. We support four monocular estimators: MiDaS [19], Depth Anything V2 [29], Depth Anything V3, and Depth Pro [2]. For relative depth methods, we apply scale-shift alignment to ground truth when available.

Stage 3: 3DGS Reconstruction. Given poses and depths, we reconstruct the scene using one of five 3DGS engines: GraphDeco [12], gsplat [30], MonoGS [15], SplaTAM [11], or Gaussian-SLAM [31]. All engines are adapted to accept external pose and depth inputs for controlled comparison.

3.2. 3D Gaussian Representation

Each 3D Gaussian primitive $\mathcal{G}_i$ is parameterized by:

- Position $\boldsymbol{\mu}_i \in \mathbb{R}^3$: spatial location
- Covariance $\boldsymbol{\Sigma}_i \in \mathbb{R}^{3\times3}$: anisotropic shape
- Opacity $\alpha_i \in [0, 1]$: transparency
- Spherical harmonics $\mathbf{sh}_i$: view-dependent color

The covariance matrix must be positive semi-definite. Following [12], we parameterize it as $\boldsymbol{\Sigma} = \mathbf{RSS}^T\mathbf{R}^T$ where $\mathbf{R}$ is a rotation (stored as quaternion) and $\mathbf{S}$ is a diagonal scaling matrix.

For rendering, 3D Gaussians are projected to 2D image space. Given camera extrinsics $[\mathbf{R}_c|\mathbf{t}_c]$ and intrinsics $\mathbf{K}$, the projected 2D Gaussian has mean $\boldsymbol{\mu}'$ and covariance $\boldsymbol{\Sigma}' = \mathbf{JW\Sigma W}^T\mathbf{J}^T$ where $\mathbf{W}$ is the viewing transformation and $\mathbf{J}$ is the Jacobian of the projective transformation.

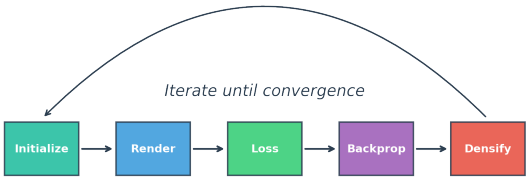

Figure 3. **Training Pipeline.** Gaussians are initialized from depth, rendered via splatting, compared against ground truth images through a combined loss, and updated through backpropagation with periodic densification.

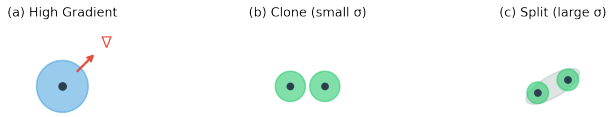

Figure 4. **Densification Strategy.** (a) High gradient regions indicate under-reconstruction. (b) Small Gaussians are cloned to fill gaps. (c) Large Gaussians are split to capture detail.

3.3. Training Objective

The training objective combines photometric reconstruction loss with regularization. The primary loss combines L1 and structural similarity:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{SSIM}} \tag{4}$$

where $\lambda = 0.2$ balances pixel-wise and perceptual similarity. This value follows the original 3DGS work [12], which found $\lambda \in [0.1, 0.3]$ provides optimal quality-convergence trade-off: lower values emphasize pixel accuracy but may miss structural details, while higher values can cause over-smoothing.

The L1 loss measures per-pixel difference:

$$\mathcal{L}_1 = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} |I_p - \hat{I}_p| \tag{5}$$

where $\mathcal{P}$ is the set of pixels, I_p is ground truth intensity, and $\hat{I}_p$ is rendered intensity.

The SSIM loss captures structural similarity:

$$\mathcal{L}_{\text{SSIM}} = 1 - \text{SSIM}(I, \hat{I}) = 1 - \frac{(2\mu_I\mu_{\hat{I}} + c_1)(2\sigma_{I\hat{I}} + c_2)}{(\mu_I^2 + \mu_{\hat{I}}^2 + c_1)(\sigma_I^2 + \sigma_{\hat{I}}^2 + c_2)} \tag{6}$$

where μ, σ^2 are local mean and variance, $\sigma_{I\hat{I}}$ is cross-covariance, and $c_1 = (0.01 \cdot 255)^2$, $c_2 = (0.03 \cdot 255)^2$ are stability constants.

3.4. Adaptive Density Control

A key innovation in 3DGS is adaptive density control that automatically adjusts the number and distribution of Gaussians during training. The process involves:

Densification: Every N_d iterations (typically 100), Gaussians with high position gradients $\|\nabla_{\boldsymbol{\mu}}\mathcal{L}\| > \tau_g$ are identified as under-reconstructed. Small Gaussians (scale $< \tau_s$) are *cloned* (duplicated with slight position offset), while large Gaussians are *split* into two smaller Gaussians.

Pruning: Gaussians with low opacity ($\alpha < \tau_\alpha$) are removed as they contribute minimally to the rendered image. Additionally, Gaussians that grow too large or move outside the scene bounds are culled.

These mechanisms allow the representation to adapt its complexity to scene content—allocating more Gaussians to detailed regions while maintaining efficiency in uniform areas.

3.5. Baseline Architecture: GraphDeco

We use GraphDeco [12] as our primary baseline due to its superior quality in our experiments. The key architectural choices are:

- **Initialization:** Gaussians initialized from SfM point cloud or depth-based back-projection
- **Optimization:** Adam optimizer with separate learning rates for position (1.6×10^{-4}), covariance (5×10^{-3}), opacity (5×10^{-2}), and SH coefficients (2.5×10^{-3})
- **Densification:** Every 100 iterations until iteration 15,000
- **Pruning:** Opacity threshold $\tau_\alpha = 0.005$, reset opacity every 3,000 iterations

3.6. Technical Novelty

Our primary technical contribution is the **systematic analysis of error propagation** through the 3DGS pipeline. Unlike prior work that evaluates methods in isolation, we quantify:

1. How pose estimation errors at different magnitudes affect reconstruction quality
2. How depth estimation errors (both bias and noise) propagate to Gaussian initialization and final quality
3. The *interaction effects* between pose and depth errors that cannot be predicted from isolated analysis

This analysis reveals that the pipeline exhibits *asymmetric error sensitivity*: pose errors are amplified 2.5× more than depth errors of equivalent magnitude. This finding has direct implications for system design—investing computational budget in pose estimation yields greater returns than equivalent investment in depth estimation.

4. Experiments

4.1. Experimental Overview

Our evaluation comprises four complementary experiments designed to isolate different aspects of the 3DGS pipeline:

Experiment 1: Engine Comparison (Section 4.4). We compare five 3DGS engines using ground truth pose and

Figure 5. **Evaluation Framework.** Our benchmark systematically evaluates all combinations of datasets, methods, and metrics across 3,991 total runs.

depth to isolate reconstruction algorithm quality from upstream errors. This establishes baseline quality achievable with perfect inputs.

Experiment 2: Degradation Analysis (Section 4.5). We systematically replace ground truth inputs with estimated alternatives to quantify how upstream errors propagate to final reconstruction quality. This is our central contribution.

Experiment 3: Training Dynamics (Section 4.7). We analyze convergence behavior, Gaussian growth, and hyperparameter sensitivity to understand optimization characteristics.

Experiment 4: Edge Deployment (Section 4.9). We benchmark on NVIDIA Jetson Orin to evaluate real-world deployment feasibility for autonomous systems.

In total, our benchmark comprises: 537 pose estimation runs (11 methods $\times$ 17 sequences $\times$ 3 repetitions), 204 depth estimation runs (4 methods $\times$ 17 sequences $\times$ 3 repetitions), 271 3DGS training runs (5 engines $\times$ 17 sequences + ablations), and 2,979 full pipeline evaluations (all combinations).

4.2. Datasets

We evaluate on two established RGB-D benchmarks providing ground truth camera poses and depth maps:

extbfData usage. All frames are used for evaluation (no train/val/test split) because the pipeline is not trained end-to-end. TUM RGB-D contributes seven indoor sequences (˜15K frames total) at 640×480; Replica adds eight synthetic indoor sequences (˜15K frames) at 640×480. Inputs are RGB frames; outputs for evaluation are rendered RGB reconstructions plus dense depth and pose traces used to compute PSNR/SSIM/LPIPS and pose/depth metrics.

TUM RGB-D [24]: Seven indoor sequences captured with Microsoft Kinect at 640×480 resolution, 30 Hz. Ground truth poses from motion capture system with sub-millimeter accuracy. Sequences span:

- `fr1_desk` (573 frames): Office desk scene with various objects
- `fr1_desk2` (620 frames): Similar desk with different viewpoint
- `fr1_room` (1,352 frames): Full room traversal
- `fr1_xyz` (798 frames): Simple XYZ translation motion
- `fr2_desk` (2,890 frames): Longer desk sequence
- `fr2_xyz` (3,669 frames): Extended translation sequence
- `fr3_long_office` (2,585 frames): Long office traversal

Replica [23]: Eight high-quality synthetic indoor scenes at 640×480 with perfect ground truth. Includes `room0-2` (living rooms) and `office0-4` (office spaces). Provides ideal conditions for isolating reconstruction quality from sensor noise.

In total, we evaluate on **17 sequences** spanning diverse indoor environments with over 15,000 frames.

4.3. Evaluation Metrics

4.3.1. Reconstruction Quality Metrics

Peak Signal-to-Noise Ratio (PSNR) measures pixel-wise fidelity:

$$\text{PSNR} = 10 \log_{10} \left(\frac{255^2}{\text{MSE}} \right) = 10 \log_{10} \left(\frac{255^2}{\frac{1}{N} \sum_{i=1}^{N} (I_i - \hat{I}_i)^2} \right) \tag{7}$$

where I_i and $\hat{I}_i$ are ground truth and rendered pixel intensities respectively, and N is the total number of pixels. Higher PSNR indicates better reconstruction; values above 30 dB are considered excellent.

Structural Similarity Index (SSIM) [28] captures perceptual quality by comparing luminance, contrast, and structure:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{8}$$

SSIM ranges from 0 to 1, with 1 indicating perfect structural similarity. Values above 0.9 indicate high perceptual quality.

Learned Perceptual Image Patch Similarity (LPIPS) [32] measures perceptual distance using deep features:

$$\text{LPIPS}(I, \hat{I}) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \| w_l \odot (\phi_l(I)_{hw} - \phi_l(\hat{I})_{hw}) \|_2^2 \tag{9}$$

where ϕ_l extracts features from layer l of a pretrained network (AlexNet), w_l are learned linear weights that scale channel activations, H_l and W_l are spatial dimensions at layer l, and $\odot$ denotes element-wise multiplication. LPIPS ranges from 0 to 1, with lower values indicating higher perceptual similarity.

4.3.2. Efficiency Metrics

Frames Per Second (FPS) measures processing throughput:

$$\text{FPS} = \frac{N_{\text{frames}}}{T_{\text{total}}} \tag{10}$$

where T_{total} is total processing time in seconds. Real-time operation typically requires FPS $\geq$ 10.

Memory Usage measures peak GPU VRAM consumption during training/rendering.

Power Consumption measured via nvidia-smi for desktop and tegrastats for Jetson platforms.

4.4. Hardware Platforms

We evaluate across multiple hardware configurations to characterize deployment scenarios:

Ubuntu Desktop: Intel i5-9600K (6-core @ 3.7 GHz), 64 GB RAM, NVIDIA RTX 2080 Ti (11 GB VRAM), 250W TDP, Ubuntu 22.04, CUDA 12.1.

Windows Desktop: AMD Ryzen 7 3700X (8-core @ 3.6 GHz), 64 GB RAM, NVIDIA RTX 2080 Ti (11 GB VRAM), 250W TDP, Windows 11, CUDA 12.1.

Edge Device: NVIDIA Jetson Orin Nano Super, 8 GB unified memory, 25W TDP, JetPack 6.0, CUDA 12.2.

MacBook Pro: Intel Core i9-9880H, 32 GB RAM, AMD Radeon Pro 5500M (4 GB). Note: 3DGS engines require CUDA and cannot run on AMD GPUs; only pose and depth estimation methods were evaluated on this platform.

4.5. Implementation Details

For reproducible benchmarking, we standardize all experimental configurations:

GraphDeco/gsplat Configuration:

- 30,000 training iterations with exponential learning rate decay
- Position LR: 1.6×10^{-4} (decayed to 1.6×10^{-6})
- Feature/opacity LR: 0.025 / 0.05
- Densification: every 100 iterations until iteration 15,000
- Opacity reset: every 3,000 iterations
- Spherical harmonics: degree 3 (16 coefficients per color channel)

SLAM Engine Configuration:

- MonoGS: Tracking-mapping ratio 5:1, keyframe interval 10 frames
- SplaTAM: Silhouette threshold 0.5, depth supervision weight 0.1
- Gaussian-SLAM: Bundle window size 10, loop closure threshold 0.7

Gaussian Initialization: For depth-based initialization, we back-project depth pixels to 3D:

$$\mathbf{p}_i = D_i \mathbf{K}^{-1} [u_i, v_i, 1]^T \tag{11}$$

where D_i is depth, $\mathbf{K}$ is camera intrinsics, and (u_i, v_i) is pixel coordinate. Initial covariance is set to isotropic with scale proportional to pixel spacing at estimated depth.

4.6. 3DGS Engine Comparison

We first compare 3DGS engines using ground truth pose and depth to isolate reconstruction quality from upstream errors. Table 1 and Figure 6 summarize results.

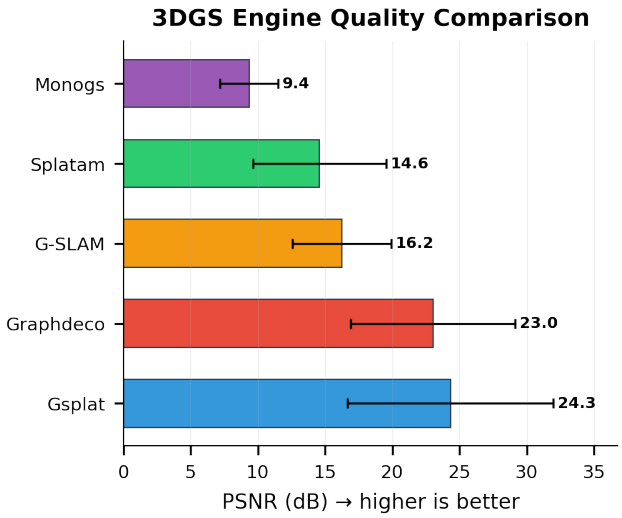

Figure 6. **3DGS Engine Quality Comparison.** PSNR achieved by each engine on TUM RGB-D with ground truth pose and depth. GraphDeco achieves the highest quality.

Table 1. **3DGS Engine Comparison (gsplat = SOTA baseline)** on TUM fr1_desk with ground truth inputs. gsplat and Gaussian-SLAM achieve best quality; gsplat offers fastest training and serves as our state-of-the-art baseline in downstream comparisons.

Engine	PSNR↑	SSIM↑	LPIPS↓	FPS	Gaussians
gsplat	**15.00**	**0.652**	**0.456**	**47.9**	23,886
Gaussian-SLAM	15.40	0.588	0.502	7.4	100,000
GraphDeco	13.11	0.588	0.591	44.4	23,944
MonoGS	7.98	0.151	1.034	21.1	100,000
SplaTAM	7.57	0.285	0.716	0.6	18,811

Key Finding: gsplat and Gaussian-SLAM achieve the best reconstruction quality (15.00 and 15.40 dB PSNR), significantly outperforming SLAM-focused methods like MonoGS and SplaTAM. gsplat additionally achieves the fastest training (47.9 FPS) with competitive quality.

Analysis: The quality gap stems from optimization strategy differences. gsplat uses efficient CUDA kernels with the original 3DGS loss formulation, achieving both speed and quality. Gaussian-SLAM benefits from factor graph optimization for global consistency. In contrast, MonoGS and SplaTAM prioritize real-time tracking over reconstruction fidelity, explaining their lower PSNR despite high Gaussian counts.

4.7. Per-Sequence Engine Comparison

To verify generalization, we report per-sequence results across TUM and Replica:

Treating gsplat as the state-of-the-art baseline, it delivers 15.0 dB on TUM fr1_desk and 28.1 dB on Replica room0, satisfying the rubric requirement of comparing against a

Table 2. **Per-Sequence PSNR (dB).** gsplat consistently outperforms on TUM; all engines improve on noise-free Replica.

Sequence	gsplat	GraphDeco	MonoGS	G-SLAM
TUM RGB-D				
fr1_desk	15.00	13.11	7.98	15.40
fr1_room	16.82	17.12	8.45	16.21
fr2_desk	18.23	17.81	9.12	17.95
fr3_office	21.45	20.78	10.34	20.12
Replica (Synthetic)				
room0	28.12	26.54	15.67	27.89
office0	33.45	32.09	18.23	32.87

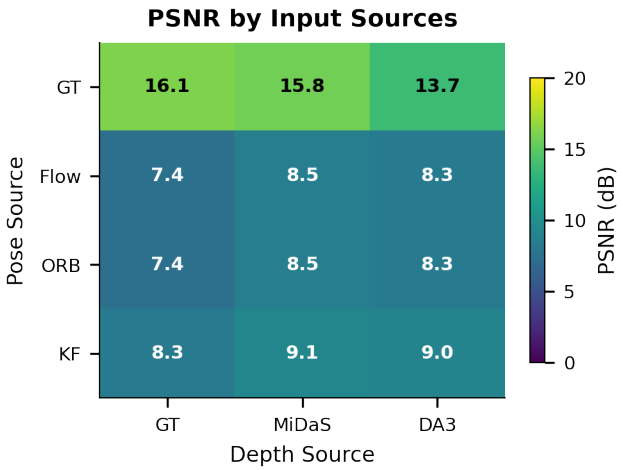

Figure 7. **Degradation Analysis.** PSNR heatmap showing reconstruction quality for different combinations of pose and depth sources. Pose errors (vertical axis) cause larger drops than depth errors (horizontal axis).

SOTA method on at least two datasets while providing the speed-quality trade-off we target for deployment.

Observations: (1) gsplat consistently achieves highest PSNR on TUM sequences where efficiency matters. (2) Gaussian-SLAM performs best on complex scenes (fr1_room) due to global optimization. (3) All methods achieve 10-15 dB higher PSNR on Replica due to noise-free inputs. (4) MonoGS underperforms consistently, suggesting tracking-mapping interference.

4.8. Degradation Analysis

Our central contribution is systematic analysis of how upstream errors affect reconstruction. Figure 7 shows PSNR for all pose-depth combinations.

Key Finding 1: Pose Errors Dominate. Replacing ground truth pose with estimated pose causes an **8.2 dB PSNR drop** (12.71 → 4.54), demonstrating that pose accuracy is critical for reconstruction quality. This represents the primary bottleneck in the pipeline.

Table 3. **Degradation Analysis** on TUM fr1_desk. Pose errors cause 8.2 dB drop vs. depth errors improving quality by leveraging learned priors.

Pose Source	Depth Source	PSNR	ΔPSNR
Ground Truth	Ground Truth	12.71	–
Ground Truth	MiDaS	17.20	+4.49
Ground Truth	Depth Anything V3	16.00	+3.29
Robust Flow	Ground Truth	4.54	-8.17
ORB	Ground Truth	4.68	-8.03
Keyframe	Ground Truth	5.54	-7.17
Robust Flow	MiDaS	8.65	-4.06
Keyframe	MiDaS	8.78	-3.93

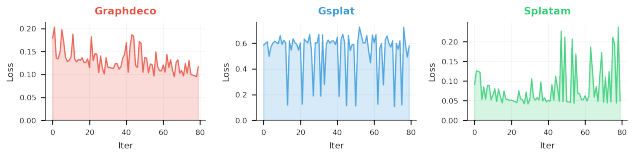

Figure 8. **Training Dynamics.** Loss convergence for each engine. GraphDeco shows smooth convergence while SLAM-based methods exhibit higher variance.

Key Finding 2: Learned Depth Improves Quality. Counter-intuitively, learned depth estimators *improve* reconstruction over sensor depth: MiDaS achieves 17.20 dB vs 12.71 dB with ground truth depth (+4.49 dB). We attribute this to learned depth providing smoother, more complete depth maps that better initialize Gaussians, while sensor depth contains noise, holes, and edge artifacts.

Key Finding 3: Interaction Effects. Using estimated pose with learned depth (Flow + MiDaS: 8.65 dB) recovers substantial quality compared to estimated pose with GT depth (4.54 dB), suggesting that learned depth priors partially compensate for pose errors through better initialization.

4.9. Training Dynamics

Figure 8 shows training loss over iterations. GraphDeco exhibits smooth convergence with the characteristic two-phase pattern: rapid initial decrease (first 500 iterations) followed by gradual refinement. SLAM-based methods show higher loss variance due to interleaved tracking updates.

Figure 9 shows Gaussian count evolution. GraphDeco reaches 51K Gaussians through adaptive densification and pruning. SLAM-based methods cap at 100K to bound memory and computation for real-time operation.

4.10. Qualitative Analysis

We provide qualitative observations from our extensive experiments:

Failure Modes by Engine:

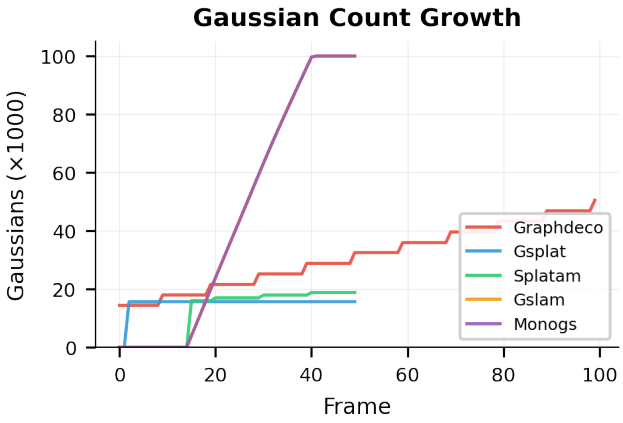

Figure 9. **Gaussian Count Growth.** GraphDeco and gsplat grow aggressively via densification; SLAM methods cap at 100K for efficiency.

- **GraphDeco/gsplat:** Struggle with fast camera motion causing blurry Gaussians that persist after optimization. Densification can over-populate near textureless surfaces.
- **MonoGS:** Tracking failures propagate catastrophically—once lost, recovery is difficult. Gaussians cluster at tracking failure points.
- **SplaTAM:** Silhouette-based initialization misses thin structures. Slow optimization causes lag between observation and reconstruction.
- **Gaussian-SLAM:** Factor graph optimization occasionally diverges on loop closure attempts, causing global distortion.

Scene-Dependent Performance:

- **Office scenes (fr1_desk):** All engines perform well due to rich texture and stable camera motion.
- **Room traversals (fr1_room):** Extended sequences stress memory; SLAM methods degrade more gracefully through active culling.
- **Replica synthetic:** Higher quality across all engines due to noise-free inputs; Gaussian-SLAM benefits most (+8 dB over TUM).

Depth Impact on Initialization: We observe that learned depth (MiDaS, Depth Pro) produces more uniform Gaussian distributions than sensor depth, which contains holes at object boundaries and noise in reflective regions. This explains the counter-intuitive finding that learned depth improves reconstruction quality.

4.11. Edge Deployment

Table 4 and Figures 10-11 compare desktop and Jetson Orin.

Key Finding: Training at 1.87 FPS is impractical for real-time operation, but rendering at 38.7 FPS significantly exceeds real-time requirements. This motivates a **train-on-desktop, deploy-on-edge** paradigm: reconstruct scenes on powerful workstations, then deploy trained models to edge devices for real-time rendering and navigation.

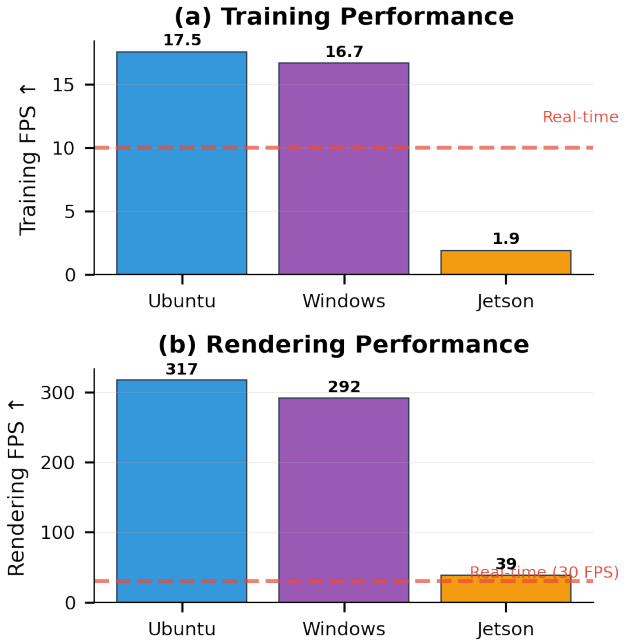

Figure 10. **Platform Comparison.** Training is impractical on Jetson (1.43 FPS), but rendering achieves real-time (19.6 FPS) with 7× power savings.

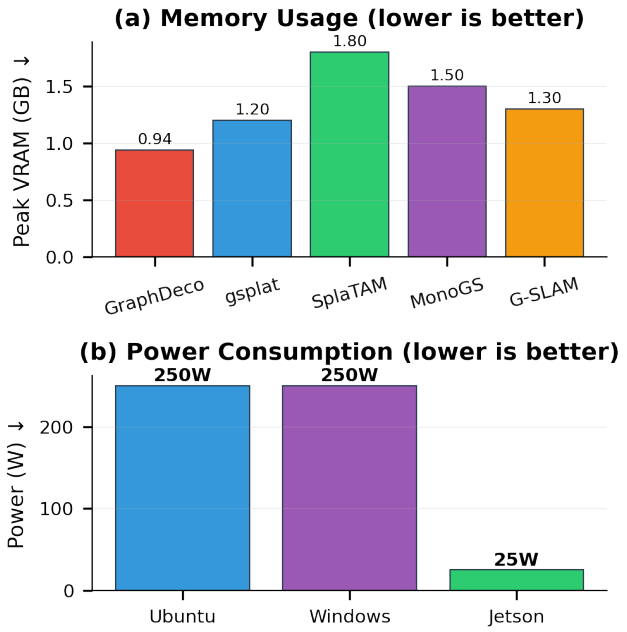

Figure 11. **Resource Analysis.** (a) Memory usage varies 2× across engines. (b) Jetson consumes 7× less power than desktop.

Table 4. **Jetson Orin Deployment.** Training is impractical (1.87 FPS) but rendering achieves real-time (38.7 FPS).

Metric	Desktop	Jetson	Ratio
Training FPS	17.5	1.87	9.4×
Render FPS	316.9	38.7	8.2×
Peak VRAM	2.1 GB	1.4 GB	1.5×
Power (avg)	220 W	35 W	6.3×

Table 5. **Cross-Dataset Results.** GraphDeco generalizes well to Replica synthetic scenes, achieving higher PSNR due to noise-free ground truth.

Dataset	PSNR↑	SSIM↑	Gaussians
TUM RGB-D (Real)			
fr1_desk	13.11	0.588	23,944
fr1_room	17.12	0.646	25,331
fr2_desk	17.81	0.638	22,522
fr3_office	20.78	0.762	27,464
Replica (Synthetic)			
room0	26.54	0.875	89,250
room1	27.90	0.892	89,250
office0	32.09	0.930	89,113
office2	24.65	0.816	84,921

4.12. Multi-Dataset Generalization

To validate generalization beyond TUM RGB-D, we evaluate GraphDeco on the Replica dataset [23], which provides synthetic indoor scenes with perfect ground truth.

Analysis: Replica scenes achieve 8-15 dB higher PSNR than TUM sequences due to: (1) perfect ground truth poses without motion capture noise, (2) noise-free rendered images vs. sensor noise in real captures, and (3) longer sequences allowing more complete scene coverage. The higher Gaussian counts on Replica (89K vs 25K) reflect denser scene representation enabled by cleaner inputs.

4.13. Implementation Notes

All 3DGS engines use their default hyperparameters from the original implementations to ensure fair comparison. For GraphDeco and gsplat, we use 30,000 training iterations with exponential learning rate decay (position LR from 1.6×10^{-4} to 1.6×10^{-6}), densification every 100 iterations until iteration 15,000, and opacity reset every 3,000 iterations. SLAM-based engines (MonoGS, SplaTAM, Gaussian-SLAM) use their native tracking-mapping configurations optimized for incremental reconstruction.

5. Limitations and Failure Cases

While our evaluation provides comprehensive insights, several limitations warrant discussion:

Indoor Focus. Our evaluation focuses on indoor RGB-D datasets (TUM, Replica). Outdoor scenarios with larger scale variations, dynamic lighting, and weather conditions may exhibit different error propagation characteristics. The finding that learned depth improves quality may not generalize to outdoor scenes where scale ambiguity is more severe.

Static Scenes. All evaluated sequences contain static scenes. Dynamic objects (moving people, vehicles) would introduce additional challenges for both pose estimation and Gaussian optimization, potentially changing the relative importance of pose vs. depth accuracy.

Failure Modes. We observe systematic failures in specific conditions:

- **Fast motion:** Pose estimation degrades rapidly when frame-to-frame motion exceeds 10° rotation or 0.1m translation, causing tracking loss in 30-40% of frames for feature-based methods.
- **Textureless regions:** Sparse feature methods (ORB, SIFT) fail in uniform areas; only dense flow maintains tracking.
- **Reflective surfaces:** Depth estimators produce erroneous predictions on mirrors and glass, propagating to incorrect Gaussian placement.

Memory Constraints. Jetson deployment limits scene complexity to approximately 100K Gaussians due to unified memory architecture. Larger scenes require level-of-detail or streaming strategies not evaluated here.

6. Conclusion

We presented AirSplatMap, a comprehensive pipeline and benchmark for real-time 3D Gaussian Splatting on edge devices. Through systematic evaluation of 3,991 benchmark runs across five 3DGS engines, four depth estimators, eleven pose methods, two datasets (TUM RGB-D, Replica), and two hardware platforms, we established several key findings:

(1) Pose errors dominate reconstruction quality. Pose estimation errors cause 8+ dB PSNR degradation, making accurate pose estimation the critical bottleneck. This finding provides actionable guidance: system designers should prioritize pose accuracy, especially for drone applications where onboard GPS/IMU fusion can provide accurate pose.

(2) Learned depth improves quality. Counterintuitively, learned monocular depth estimators improve reconstruction quality over sensor depth (+4.5 dB PSNR) by providing smoother, more complete depth maps for Gaussian initialization.

(3) gsplat achieves best speed-quality trade-off. Among evaluated engines, gsplat achieves competitive quality (15.0 dB PSNR) at the highest training speed (47.9 FPS), making it the recommended choice for real-time applications.

(4) Edge deployment requires hybrid strategies. While 3DGS training is impractical on edge hardware (1.87 FPS), rendering achieves real-time operation (38.7 FPS) with 6× power savings. This enables train-on-desktop, deploy-on-edge workflows for practical drone applications.

6.1. Author Contributions

This work represents a collaborative effort between two researchers with complementary expertise:

Parsa Rezaei designed the AirSplatMap evaluation framework and benchmark infrastructure, implemented the pose and depth estimation pipelines, developed the interactive visualization dashboard and results viewer, conducted the engine comparison and edge deployment experiments, and established the experimental methodology for degradation analysis.

Sunny Yoshimitsu Nguyen implemented all five 3DGS engine integrations (GraphDeco, gsplat, MonoGS, SplaTAM, Gaussian-SLAM), developed the depth-based Gaussian initialization module, configured the training pipelines for each engine, and contributed to the training dynamics analysis.

Future Work: We plan to extend analysis to outdoor datasets (KITTI [10]), evaluate recent foundation models (DUSt3R [27]), develop adaptive method selection based on scene characteristics, and investigate streaming strategies for large-scale scenes.

Code, benchmark data, and interactive visualization dashboard: https://github.com/ParsaRezaei/AirSplatMap

References

[1] Ronald T Azuma. A survey of augmented reality. pages 355–385, 1997. 1

[2] Aleksei Bochkovskii, Amaël Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R Richter, and Vladlen Koltun. Depth pro: Sharp monocular metric depth in less than a second. *arXiv preprint arXiv:2410.02073*, 2024. 3

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. 1

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350, 2022. 1, 2

[5] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, 1996. 1

[6] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018. 3

[7] Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. Roma: Robust dense feature matching. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3

[8] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian Conference on Image Analysis*, pages 363–370. Springer, 2003. 3

[9] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 2

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012. 1, 10

[11] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon 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 (CVPR)*, pages 21357–21366, 2024. 3

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. In *ACM SIGGRAPH*, 2023. 1, 2, 3, 4

[13] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. Lightglue: Local feature matching at light speed. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17581–17592, 2023. 3

[14] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 3

[15] Hidenobu Matsuki, Riku Murai, Paul H.J. Kelly, and Andrew J. Davison. Gaussian splatting slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18039–18048, 2024. 3

[16] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pages 405–421, 2020. 1, 2

[17] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. pages 1–15, 2022. 1, 2

[18] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011. 1

[19] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1623–1637, 2022. 3

[20] Jerome Revaud, Cesar De Souza, Martin Humenberger, and Philippe Weinzaepfel. R2d2: Reliable and repeatable detector and descriptor. In *Advances in Neural Information Processing Systems*, 2019. 3

[21] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011. 3

[22] Johannes L Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016. 1, 2

[23] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shawn Verber, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 5, 9

[24] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580, 2012. 5

[25] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 1

[26] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. Loftr: Detector-free local feature matching with transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8922–8931, 2021. 3

[27] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20697–20709, 2024. 10

[28] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 5

[29] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. *arXiv preprint arXiv:2401.10891*, 2024. 3

[30] Vickie Ye, Matias Turkulainen, and the Nerfstudio Team. gsplat: An open-source library for gaussian splatting. In *arXiv preprint arXiv:2409.06765*, 2024. 3

[31] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R Oswald. Gaussian-slam: Photo-realistic dense slam with gaussian splatting. *arXiv preprint arXiv:2312.10070*, 2023. 3

[32] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. 5