SparseSplat: Towards Applicable Feed-Forward 3D Gaussian Splatting with Pixel-Unaligned Prediction

Supplementary Material

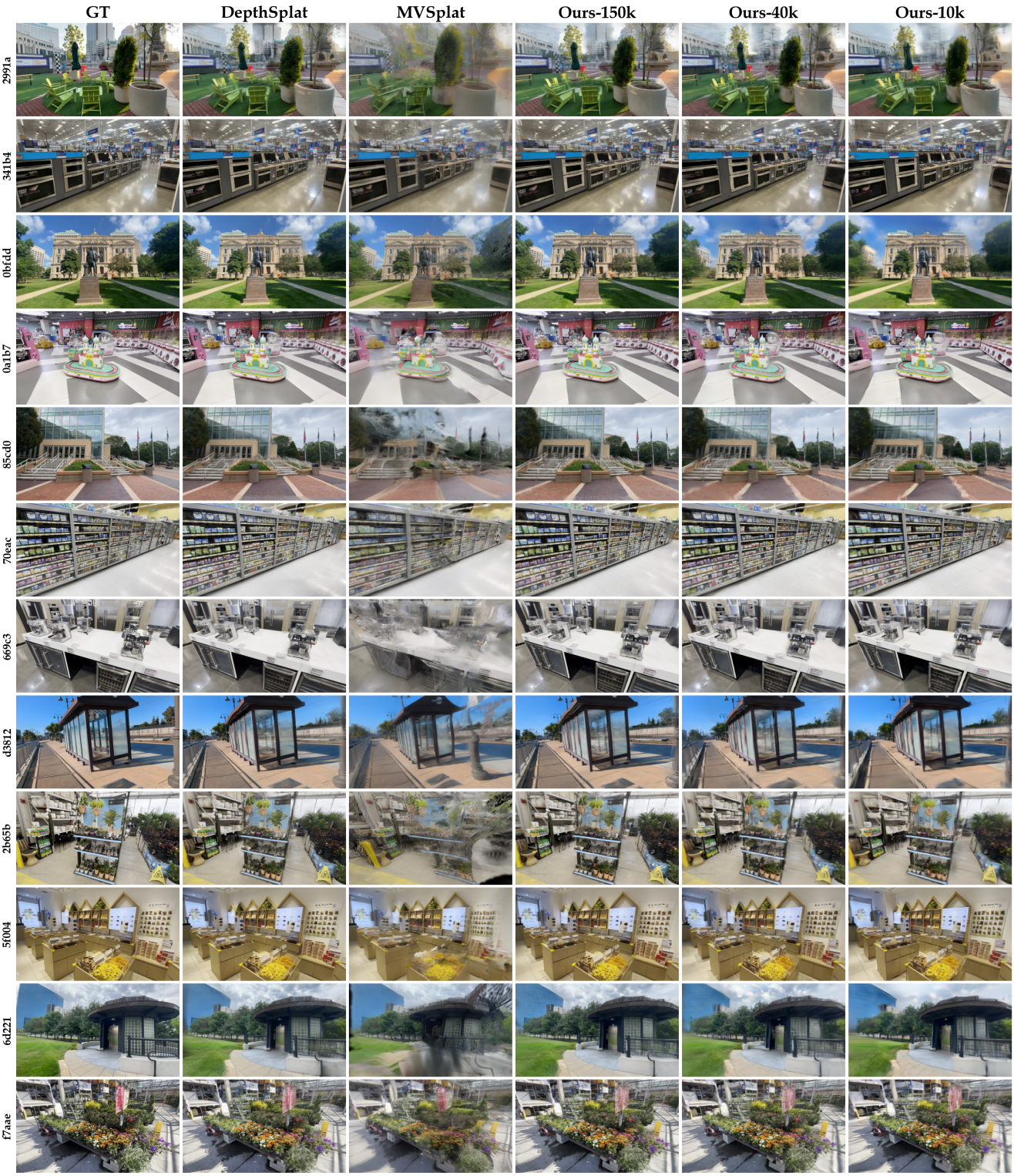

Figure 5. Additional qualitative comparisons.

This supplementary material provides additional quantitative and qualitative results, along with an assessment of SparseSplat's applicability to downstream scenarios. We also present architectural details of the prediction heads and confirm that our sparse-by-design representation delivers strong accuracy–efficiency trade-offs across varying Gaussian budgets.

6. Applicability to Downstream Tasks

To demonstrate the practical value of SparseSplat, we evaluate how it integrates into various downstream tasks. We categorize these tasks based on two different forms of "real-time" requirements: **Reconstruction Real-time-ness**, which underpins online mapping and robotic perception, and **Rendering Real-time-ness**, which is indispensable for immersive AR/VR experiences and large-scale simulation. Owing to its sparse-by-design architecture, SparseSplat naturally accommodates both types of real-time constraints, whereas pixel-aligned feed-forward methods struggle to satisfy them simultaneously due to their rigid, dense Gaussian allocation.

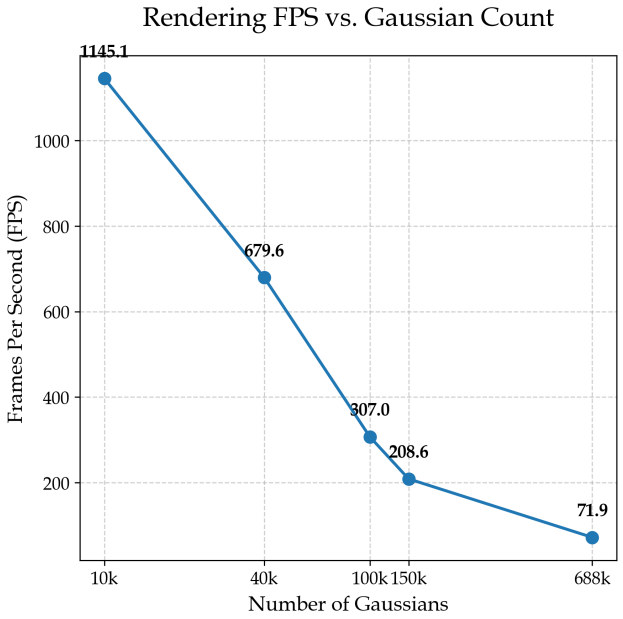

Figure 6. **Rendering Efficiency vs. Gaussian Count. X-axis log-scaled.** We evaluate the rendering frame rate (FPS) across scenes with varying degrees of sparsity. While pixel-aligned baselines (typified by ∼688k Gaussians) operate at 71.9 FPS, our sparse-by-design approach significantly accelerates rendering. Our 150k model achieves ∼3× speedup (208.6 FPS), and extremely sparse settings (10k–40k) unlock rates suitable for high-frequency robotics control loops (>600 FPS).

6.1. Reconstruction Real-time-ness: 3DGS-SLAM

Context. Simultaneous Localization and Mapping (SLAM) requires the online construction of 3D scenes from continuous video streams. Leveraging 3D Gaussian Splatting, recent methods like GS-SLAM [40] and SplaTAM [14] enable faster updates and better reconstruction than prior baselines. A crucial objective for these systems is controlling the primitive count: as shown in Fig. 6, rendering latency is strictly tied to the Gaussian budget, where lower counts ensure real-time performance.

Integration and Sparsity. Feed-forward methods can be seamlessly integrated into these pipelines to replace computationally expensive per-frame optimization. Specifically, given a window of N incoming frames and their estimated poses (from the tracking thread), SparseSplat directly predicts a set of 3D Gaussians.

However, a critical bottleneck for existing feed-forward methods (e.g., DepthSplat [39], MVSplat [3]) is their "pixel-aligned" nature. They rigidly generate a Gaussian for every pixel, adding 688k primitives per inference. In one SLAM sequence with thousands of frames, this rapid accumulation leads to memory explosion and tracking lag, effectively rendering them unusable for long-sequence operation. In contrast, SparseSplat employs a "sparse-by-design" representation. By adaptively allocating primitives only where necessary (e.g., 10k–40k per view), our method significantly reduces memory consumption and update overhead, enabling robust operation on resource-constrained platforms where pixel-aligned baselines fail.

Table 6. Quantitative comparison of Gaussian primitive counts across different methods and datasets. Data sourced from CaRtGS.

Dataset	Method	Sensor	Gaussian Primitives (M)
Replica (office0)	GS-ICP-SLAM	RGB-D	1.57 M
	Photo-SLAM	RGB-D	0.08 M
	Photo-SLAM	Monocular	0.08 M
TUM-RGBD (fr1/desk)	GS-ICP-SLAM	RGB-D	1.56 M
	SplaTAM	RGB-D	0.96 M
	Gaussian-SLAM	RGB-D	0.70 M
	MonoGS	Monocular	0.04 M
	Photo-SLAM	Monocular	0.02 M
TUM-RGBD (fr3/office)	GS-ICP-SLAM	RGB-D	1.09 M
	Gaussian-SLAM	RGB-D	1.04 M
	SplaTAM	RGB-D	0.79 M
	MonoGS	Monocular	0.03 M
	Photo-SLAM	Monocular	0.01 M

Scalability Analysis. A potential concern regarding our method is the use of K-Nearest Neighbors (KNN) for local attribute prediction: does the inference time increase unacceptably as the global map grows? It is important to clarify that while the *global* map accumulates millions of points over time, our inference is local. We only perform KNN

queries for the sparse anchor points generated from the current input batch of N frames. Therefore, the computational cost per update step remains constant and low, preserving the real-time nature of the system regardless of the total map size.

6.2. Rendering Real-time-ness: AR/VR and Simulation

The second category of applications prioritizes the quality and speed of the *final rendering* rather than the speed of the reconstruction process itself.

AR/VR on Edge Devices. In Virtual and Augmented Reality, devices (e.g., headsets, mobile phones) operate under strict VRAM and thermal constraints. A scene represented by 688k Gaussians (e.g., via DepthSplat) imposes a significantly heavier rendering load than one represented by 150k Gaussians (Ours). By offering a 4.5× reduction in primitive count without sacrificing quality, SparseSplat ensures higher rendering frame rates (FPS) and lower memory footprint, making it ideal for on-device deployment.

High-Throughput Simulation for Robotics. In robotics, agents are often trained in simulated environments (e.g., for visual navigation). These training loops require rendering millions of observations. Consequently, the rendering speed of the environment becomes the primary bottleneck for training throughput. Our method generates highly compact digital assets that render significantly faster than those produced by pixel-aligned methods. This efficiency directly translates to faster simulation speeds, allowing for more efficient large-scale training of visual reinforcement learning algorithms.

7. Runtime Breakdown

We present a detailed runtime breakdown of individual components across varying Gaussian counts in Tab. 7. The latency of backbone inference and entropy-based sampling remains constant regardless of the sparsity level. In contrast, the computational costs of the KNN query and the Attention prediction head scale with the number of generated Gaussians.

8. Structure of Different Heads

As described in Sec. 3.3, our 3D-Local Attribute Prediction framework employs a lightweight predictor to regress Gaussian attributes based on K-nearest neighbors in 3D space. We explored four different prediction head architectures, all sharing the same dual projection strategy for processing geometric and image features but differing in how they aggregate information from the local neighborhood. This section provides structural details of these variants.

Table 7. **Inference Time Breakdown (ms).** We detail the computational cost of each component in SparseSplat across different Gaussian counts. Note that the total time reported here is slightly higher than in the main text due to the synchronization overhead introduced by time profiling.

Model	Backbone	Sampling	KNN	Attention	Total
Ours-10k	89.68	1.66	3.21	5.95	**100.50**
Ours-40k	89.68	1.82	14.87	15.04	**121.41**
Ours-100k	89.68	2.16	52.02	65.18	**209.04**
Ours-150k	89.68	2.08	219.64	103.31	**414.71**

8.1. MLP Head

For each anchor point, we gather features from its K nearest neighbors and compute their relative positions $\Delta\mathbf{p}_j = \mathbf{p}_j - \mathbf{p}_i$. The aggregated neighborhood features (relative geometry and appearance) are concatenated with the center point's features and fed into a 4-layer MLP to directly regress the Gaussian parameters.

8.2. DGCNN Head

The DGCNN head adopts the EdgeConv architecture from [35], which dynamically constructs local graphs to capture fine-grained geometric structures. For each anchor point, EdgeConv models pairwise relationships with its neighbors by computing edge features that encode both local geometric context and absolute point features. Specifically, for a center point with feature $\mathbf{f}_i$ and its neighbor $\mathbf{f}_j$, the edge feature is computed as:

$$\mathbf{h}_{ij} = \text{MLP}(\mathbf{f}_i \oplus (\mathbf{f}_j - \mathbf{f}_i)) \tag{9}$$

where $\oplus$ denotes concatenation, and $\mathbf{f}_j - \mathbf{f}_i$ captures the relative geometric offset. This formulation allows the network to learn both global point characteristics (through $\mathbf{f}_i$) and local geometric variations (through the difference term).

The architecture consists of multiple stacked EdgeConv layers with progressively increasing channel dimensions. After each EdgeConv layer, a symmetric max-pooling operation aggregates edge features across neighbors, ensuring permutation invariance. The features from all EdgeConv layers are then concatenated to form a multi-scale representation that captures geometric patterns at different levels of abstraction. A final prediction head regresses the Gaussian attributes from this rich feature descriptor.

8.3. PointNet Head

The PointNet head is based on the classical PointNet architecture [21]. It treats the center point and its K neighbors as an unordered set of $K+1$ points. A shared MLP with architecture [128 → 256 → 512 → 1024] processes each point independently, and a max-pooling operation aggregates the point-wise features into a global descriptor. A subsequent prediction MLP [1024 → 512 → 256 → 38] regresses the

Gaussian parameters from this descriptor. The shared MLP and symmetric pooling ensure permutation invariance.

8.4. Geo-Aware Attention Head

The Geo-Aware Attention head employs a Point Transformer-style vector attention mechanism [46], which we find most effective for this task. Unlike simple pooling-based aggregation, it adaptively weights neighbor contributions based on both feature similarity and geometric relationships.

Position-Aware Vector Attention. The core mechanism computes attention weights that incorporate relative positions. For each center point $\mathbf{f}_i$ and its neighbors $\{\mathbf{f}_j\}_{j=1}^{K}$, we compute queries $\mathbf{Q}_i$, keys $\mathbf{K}_j$, and values $\mathbf{V}_j$ via learned projections. The relative position $\Delta\mathbf{p}_j = \mathbf{p}_j - \mathbf{p}_i$ is encoded through an MLP to obtain position embeddings δ_j. The attention weights are then computed as:

$$\alpha_{ij} = \text{softmax}_j\left(\frac{(\mathbf{Q}_i - \mathbf{K}_j + \delta_j) \cdot \mathbf{Q}_i}{\sqrt{d}}\right) \tag{10}$$

where the position encoding δ_j modulates the attention computation. The output feature is obtained by:

$$\mathbf{f}_i' = \sum_{j=1}^{K} \alpha_{ij}(\mathbf{V}_j + \delta_j) \tag{11}$$

This design allows the network to learn geometry-aware feature aggregation, where the position encoding influences both attention weights and value features.

Among all examined variants, the Geo-aware attention head provides the best balance between accuracy and efficiency, underscoring the importance of geometry-conditioned local aggregation in SparseSplat.

Architecture. We employ three stacked attention layers with hidden dimension 128 and 4 attention heads per layer. Each layer includes a feed-forward network (FFN) with expansion ratio 4, residual connections, and layer normalization. The dual projection strategy separately processes geometric features and image features before concatenation, maintaining modularity between geometry and appearance processing.