InstGS: Shared-Template Gaussian Instancing for Object-Redundancy-Free Rendering

Zi'ang Lu[1], Kang Du[1], Xinyao Wei[4], Qian Zhang[4], John Li[4], Dong Liang[3,4✉], Zeyu Wang[1,2✉], and Jinyuan Jia[4✉]

[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] The Hong Kong University of Science and Technology
[3] City University of Hong Kong [4] Tongji University
donliang5-c@my.cityu.edu.hk, zeyuwang@ust.hk, csjyjia@aliyun.com
https://github.com/Strange-tech/InstGS

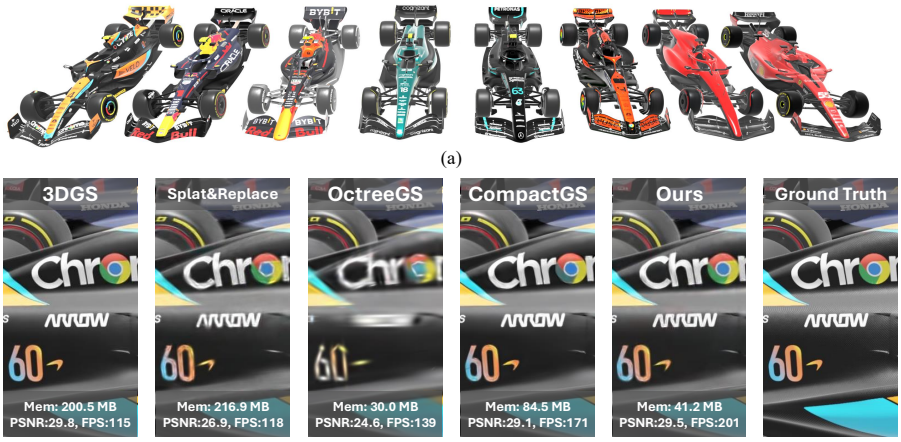

Fig. 1: Rendering comparison of 8 Formula racing cars. (a) Our InstGS represents multiple cars with similar geometry but distinct appearances using only a small set of shared parameters. (b) Compared with existing methods, our method achieves lower memory, higher PSNR, and significantly faster rendering in repetitive-object scenes.

Abstract. 3D Gaussian Splatting (3DGS) and Neural Radiance Fields (NeRF) have demonstrated remarkable capabilities in photo-realistic novel view synthesis. However, their practical adoption is often hindered by substantial storage requirements and limited rendering efficiency, particularly for scenes with repetitive structures. While existing acceleration methods primarily focus on optimizing individual Gaussian primitives or neural network architectures, they fail to address the fundamental redundancy inherent in repetitive content. To overcome this limitation, we introduce InstGS, the first Gaussian instancing-based accelerated rendering framework. To eliminate redundancy at the representation level, we perform gradient-driven cross-frame instance segmentation to group

✉ Corresponding authors.

similar Gaussians into reusable components. A shared Gaussian template with instance-specific offsets is optimized to replace all similar instances, yielding substantial memory saving with negligible loss in visual fidelity. Extensive experiments demonstrate that InstGS achieves high-quality, high-frame-rate, and low-memory rendering performance.

1 Introduction

Recent advances in implicit [10, 11, 30, 31] and explicit 3D representations have significantly improved the fidelity of novel view synthesis. Among them, Neural Radiance Fields [32] and 3D Gaussian Splatting [17] have emerged as two dominant paradigms for reconstructing and rendering photorealistic 3D scenes from multi-view images. NeRF leverages volumetric neural fields to model continuous radiance and density, achieving impressive results but often suffering from slow rendering due to its reliance on neural network inference at each pixel. In contrast, 3DGS represents scenes as a collection of explicit Gaussian primitives, enabling real-time rendering with competitive visual quality.

Despite these successes, both NeRF and 3DGS face challenges in scalability and efficiency when applied to large or structurally repetitive environments, such as urban scenes with buildings or indoor spaces with chairs. The sheer number of primitives or network parameters required leads to substantial storage overhead and inefficient rendering, limiting their deployment in resource-constrained or interactive applications.

Existing acceleration techniques primarily focus on optimizing Gaussian primitives [4,5,36], network architectures [1,2,34], or rendering pipelines [21,29,37,41] to improve performance. However, these approaches typically treat each primitive or voxel as independent, overlooking the inherent redundancy present in repetitive content. These challenges suggest the potential of leveraging instance-level priors to share information among repetitive structures. Instance-based Gaussian representations have been explored in tasks such as semantic understanding [3, 23, 39], scene decomposition [7, 44, 49], and reconstruction completion [24, 48]. None of them focus on instancing-based acceleration at the GPU level for rendering. As a result, repetitive structures in real-world scenes are still redundantly encoded and rendered multiple times, leading to unnecessary memory usage and computational overhead.

To address this gap, we present InstGS, a novel GPU-accelerated instancing-capable pipeline for object-redundancy-free rendering. First, we perform instance segmentation via gradient-driven segmentation using consistent 2D masks. Then, we introduce a template-and-offset instancing scheme, enabling efficient optimization and rendering of repetitive objects. InstGS enables reuse of a learned template Gaussians, together with lightweight per-instance offsets in position, base color, and opacity. By decoupling the shared geometry/appearance from per-instance variation, InstGS supports high-frame-rate rendering and low memory footprint for redundant scenes.

Our contributions can be summarized as follows:

- We propose an improved instance segmentation approach that achieves segmentation with low memory usage and no additional training cost.
- We propose a compact, learnable template-and-offset representation and a differentiable instancing rasterizer, which jointly enable GPU heterogeneous instancing for low-memory, high-FPS rendering by decomposing repetitive objects into a shared template and lightweight offsets that capture geometric and appearance variations.
- We construct a diverse dataset consisting of both synthetic and real-world scenes, all of which contain similar and repetitive objects to varying degrees, for evaluating the effectiveness and robustness of our method.

2 Related Work

2.1 Memory-Efficient Representation

Early efforts aimed at accelerating NeRF training and inference through multi-resolution hash encoding [34] or sparse voxel grids [12]. 3DGS achieves faster rendering through rasterization, at the cost of higher data volume. Some works employ template pruning, clustering, and vector quantization to achieve high compression ratios [8, 35, 36]. Others utilize structured or hierarchical representations, such as grid- or tri-plane-based layouts [21,50], to compactly encode spatial redundancy. Hash-grid–assisted contextual coding and constrained Gaussian counts [4, 9, 14] are further explored to improve efficiency and scalability, while several extensions support both static and dynamic scenes [27]. A few lossless strategies aim to reduce memory footprint through resolution-aware pruning or 2D Gaussian reformulation [37, 58], maintaining visual fidelity while improving rendering performance.

However, most of these methods focus solely on individual Gaussian optimization and overlook repetitive or shared structures. Moreover, they require decompression before rendering, meaning that the runtime data size remains nearly unchanged.

2.2 Large-Scale Scene Reconstruction

NeRF-based methods [40,46,47] have been extended to city-scale scenes through various scalable designs, but struggle with slow rendering rates and excessive memory consumption. Recent works extend 3D Gaussian Splatting to large-scale and hierarchical settings to improve scalability, memory efficiency, and rendering stability. They introduce block-based [25, 28, 51] or octree-structured hierarchies [41] to manage massive urban scenes, enabling adaptive Level-of-Detail (LOD) control and consistent rendering across scales [56]. Others employ progressive or multi-resolution optimization [6, 16, 18, 22] to reconstruct and render large scenes efficiently.

However, these approaches mainly rely on divide-and-conquer or LOD strategies, without considering instance reuse or shared Gaussian representations, leading to redundant storage and computation across repeated structures.

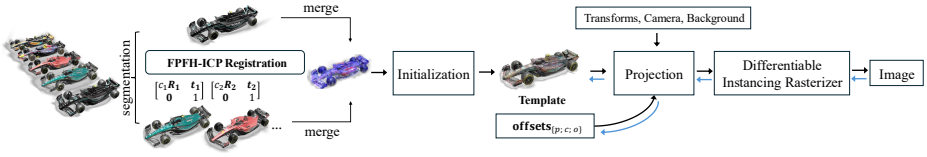

Fig. 2: Overview of our InstGS pipeline. 1) **Template Initialization.** All object instances are first segmented and registered to build a shared geometric template that captures their common structure. 2) **Differentiable Instancing Rasterizer.** For each instance, we apply learnable attribute offsets, such as appearance and fine geometry, to adapt the shared template for instance-specific rendering.

2.3 Instance-Based Gaussian

One popular task of instance-based Gaussians is segmentation. Recent studies extend 3D Gaussian Splatting to semantic and instance segmentation. These methods leverage 2D-3D mask lifting, open-vocabulary models, or graph-based optimization to assign semantic or instance labels directly in the Gaussian domain [3, 13,38,57]. Another task focuses on modular reconstruction. Recent works explore modular and hierarchical Gaussian representations [44, 54], enabling instance-level decomposition and part-controllable 3D generation for more efficient and adaptive rendering. The most representative works include: ProcGS [24], which decomposes and recombines architectural components (such as windows) to generate diverse urban landscapes, and Splat-and-Replace [48], which completes identical instances to improve reconstruction completeness. However, their implementations mainly rely on copying identical Gaussians, without considering GPU-level acceleration for rendering similar Gaussians.

Inspired by these instance-based designs, we extend the concept to efficient Gaussian rendering, aiming to reuse shared Gaussian templates while maintaining instance-specific appearance and full differentiability.

3 Overview

The pipeline of InstGS is illustrated in Figure 2. The input of our method is a well-trained 3DGS representation. First, we initialize a shared template from similar instances (Section 4.2), which involves open-vocabulary-based instance segmentation followed by instance merging. Next, we design a differentiable instancing rasterizer (Section 4.3), which takes the template derived from the original scene, along with instance-specific offsets and the background, and jointly optimizes Gaussian attributes to render scenes with many similar instances using lower memory and achieving higher frame rates.

4 Method

4.1 Preliminaries

Given a training dataset of multi-view 2D images with camera views $\mathcal{V}$, 3DGS learns a representation with a set of 3D Gaussians $\mathcal{G}$. Position, scale, rotation, opacity, and color features (SH coefficients) of every Gaussian are optimized during training. For a specific view, 3DGS rasterizes the Gaussians onto the 2D screen and computes the color $\mathbf{C}$ of a pixel $\mathbf{p}$:

$$\mathbf{C}(\mathbf{p}) = \sum_{i=1}^{N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) = \sum_{i=1}^{N} \mathbf{c}_i \alpha_i T_i, \tag{1}$$

where N is the total number of Gaussians, each indexed by its sorted position, $\mathbf{c}_i$ and α_i are the color and opacity associated with the i-th Gaussian, and $T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$ is the transmittance of i-th Gaussian at $\mathbf{p}$.

According to Equation 1, one Gaussian can correspond to a pixel. Conversely, from a pixel, we can obtain the Gaussians that contribute to its color:

$$\frac{\partial \mathbf{C}}{\partial \mathbf{c}_i} = \alpha_i T_i, \ \ i \in [1, N] \tag{2}$$

This derivative is positive if and only if both the transmittance and opacity are positive, indicating that the Gaussian contributes to the final color [15]. This is quite useful for the task of mapping image segmentation to 3DGS segmentation.

4.2 Template Initialization

Our method depends on known repeated categories, which is acquired by open-vocabulary instance segmentation. Subsequently, a template is initialized via merging similar Gaussian instances.

Instance Segmentation. Existing 3DGS segmentation methods often face the following challenges: 1) Occlusion and cross-frame inconsistency introduce noise into the segmentation results. 2) Feature learning based on contrastive learning requires high training time and memory costs. Moreover, most existing methods tend to focus on panoptic segmentation, whereas we only care about repetitive objects. So we first need to obtain cross-frame consistent masks based on object tracking. To address this issue, inspired by InstaScene [55] and Equation 2, we introduce gradient-driven 3D segmentation based on object tracking. Given brief text prompts for all target instances, Grounding DINO [26] is employed to provide 2D bounding boxes around the instances. Since it incorporates semantic understanding, segmentation can resist interferences such as lighting and shading variations. Then, we utilize SAM [19] to perform pixel-level segmentation of the same instance across frames. Since the masks generated by SAM are inconsistent across frames, we need to determine which two masks belong to the same object. We adopt the view consensus rate as proposed in [53] to obtain cross-frame consistent masks $\mathcal{M} = \{\mathbf{M}^k\}$ for the same object.

Next, we propose a gradient-driven segmentation method. Given an instance mask $\mathbf{M} \in \{0,1\}^{H\times W}$, where $\mathbf{M}_{ij} = 1$ marks pixels inside the mask, we construct a masked MSE loss to activate gradients of Gaussians within it. During rendering, the function outputs per-Gaussian gradients, and the squared error amplifies deviations, ensuring all relevant Gaussians are captured. To further refine segmentation, we introduce weighted background removal, as large background Gaussians may still affect masked pixels. For each camera view $\mathbf{v}^k \in \mathcal{V}$ with mask $\mathbf{M}^k$, the gradient of $\mathcal{G}$ is computed accordingly.

$$\mathbf{g} = \sum_{\mathbf{v}^k\in\mathcal{V}} \mathrm{GSeg}(\mathcal{G}, \mathbf{v}^k, \mathbf{M}^k) - \alpha \cdot \sum_{\mathbf{v}^k\in\mathcal{V}} \mathrm{GSeg}(\mathcal{G}, \mathbf{v}^k, \mathbf{1} - \mathbf{M}^k), \tag{3}$$

where α serves as a weighting factor for background removal. It is set to be less than 1 because some Gaussians of the instance may lie outside the camera view, and the gradients of these Gaussians need to remain positive. We encourage α to be set to 0.8. $\mathrm{GSeg}(\cdot)$ is defined as a function that performs a single iteration of optimization and retrieves the gradients of Gaussians. For instance, its Gaussians should correspond to the subset of Gaussians with positive gradients:

$$\mathcal{G}_{inst} = \{\mathcal{G}_i \in \mathcal{G} \mid \mathbf{g}_i > 0, i \in [1, N]\} \tag{4}$$

The masks of different instances in each frame are non-overlapping. Therefore, we apply the aforementioned algorithm across the entire video, thereby accomplishing Gaussian instance segmentation.

For background, its Gaussians should correspond to the subset of Gaussians with negative gradients:

$$\mathcal{G}_{bg} = \{\mathcal{G}_j \in \mathcal{G} \mid \mathbf{g}_j < 0, j \in [1, N]\} \tag{5}$$

Instance Merging. For a group of similar instances, one instance is fixed, and the remaining instances are registered with it pairwise. We always select the first instance as the fixed one, as this choice is not critical. We utilize Fast Point Feature Histograms (FPFH) [43] for global coarse registration and point-to-point Iterative Closest Point (ICP) [42] for fine registration. The registration produces a 4×4 transformation matrix between each pair of instances. We use these transformations to align all instances to a common pose and then perform a merge-and-sampling procedure, where the number of sampled Gaussians equals the average number of Gaussians per instance. Only the Gaussian positions and base colors are preserved, while all other Gaussian attributes are reinitialized to form the Gaussian template.

4.3 Differentiable Gaussian Instancing

The Template and Offsets. The template is a complete representation, where each Gaussian contains position, scale, rotation, SH coefficients (including base color and rest terms), and opacity. It serves as the average representation shared by all instances. However, appearance differences between instances are inevitable,

such as variations in lighting, subtle geometric discrepancies, or even entirely different textures. To address this, we design three types of Gaussian offsets corresponding to position, base color, and opacity. These offsets are added to the corresponding attributes of the template to maintain variations of instances. Unlike ProcGS and Splat-and-Replace, our template–offsets paradigm has the following characteristics: 1) The offset mechanism supports substantially larger appearance variations across instances, such as different liveries of Formula racing cars in the teaser. 2) All template–offsets computations are performed entirely on the GPU, rather than relying on simple CPU-side duplication.

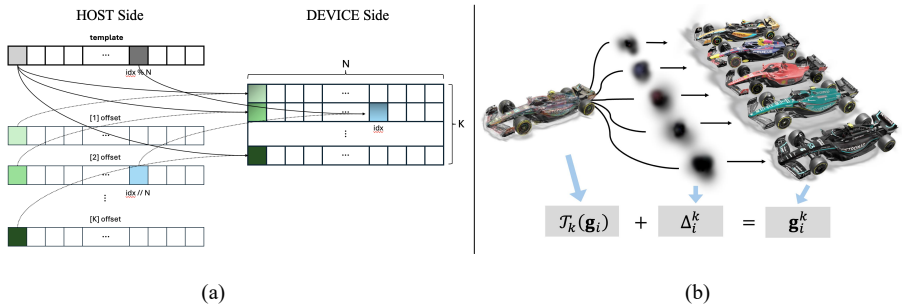

Fig. 3: Illustration of the forward rendering equation. (a) On the GPU, the template and offsets reside on the host side. Each square represents a Gaussian primitive. N and K denote the number of Gaussians and the number of instances, respectively. Each Gaussian in the template is combined with its corresponding offset Gaussian and computed on the device side. (b) Visualization of the template, offsets, and the rendered instances. Δ_i^k are approximately spherical, as the position offsets of most Gaussians between instances are close to zero.

Forward Rendering. Now we have the initialized Gaussian template, the transformation matrix from template to each instance, and the per-instance offsets. These data are transmitted from the CPU to the GPU. On the GPU, each CUDA kernel is responsible for processing one Gaussian point, performing the following operations sequentially: 1) applying the transformation matrix to map the template Gaussian to the instance space. 2) applying the corresponding offset to adjust its geometric or appearance attributes. Formally, the process can be expressed as:

$$\mathbf{g}_i^k = \mathcal{T}_k(\mathbf{g}_i) + \Delta_i^k, \tag{6}$$

where $\mathbf{g}_i$ denotes the i-th Gaussian in the template, $\mathcal{T}_k(\cdot)$ is the geometric transformation of the k-th instance, and Δ_i^k represents the offset applied to that instance. More specifically, the transformation is applied only to the positions and rotations of the Gaussians, while other attributes remain unchanged. The offsets are applied in an element-wise additive manner to these corresponding attributes, enabling each instance to exhibit subtle geometric and appearance

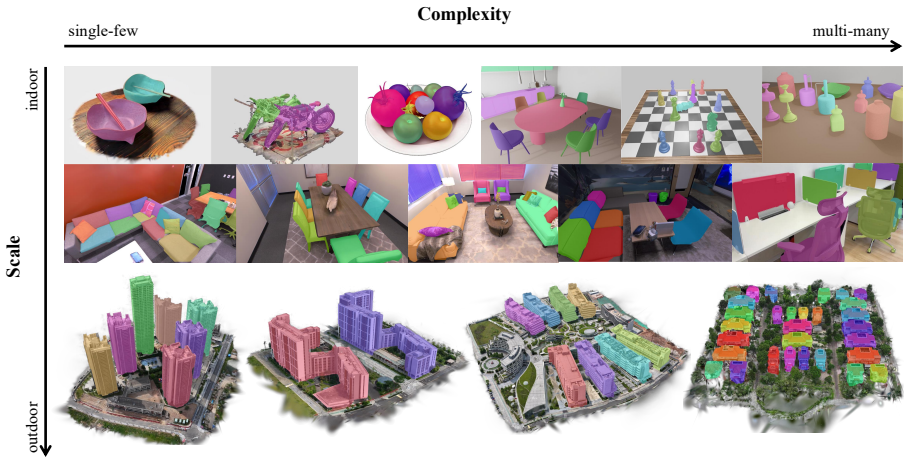

Fig. 4: Representative scenes in *ReScene* range from indoor scenes with single category and few objects to outdoor scenes with multiple categories and many objects. We intentionally retain noticeable differences among instances to better demonstrate the robustness and scalability of our method.

differences while maintaining the shared structure of the template. Figure 3 visualize the process of forward rendering.

Joint Optimization. We use the same loss function as vanilla 3DGS to compute the gradients of the rendered Gaussians. The loss function $\mathcal{L}$ consists of an L1 term combined with a D-SSIM term. Then, we jointly optimize the template, offsets, and background Gaussians during training. According to Equation 6, the gradients of the template $\mathbf{g}_i$ and offsets Δ_i^k are derived from the gradients of the Gaussians actually rendered in each CUDA kernel, according to the chain rule.

Let $\frac{\partial \mathcal{L}}{\partial \mathbf{g}_i^k}$ denote the gradient obtained from the rendered Gaussian. Then, the gradients of Δ_i^k are computed as:

$$\frac{\partial L}{\partial \Delta_i^k} = \frac{\partial L}{\partial \mathbf{g}_i^k} \frac{\partial \mathbf{g}_i^k}{\partial \Delta_i^k} = \frac{\partial L}{\partial \mathbf{g}_i^k} \cdot \mathbf{I} = \frac{\partial L}{\partial \mathbf{g}_i^k}, \tag{7}$$

the gradients of $\mathbf{g}_i$ are computed as:

$$\frac{\partial L}{\partial \mathbf{g}_i} = \sum_k \frac{\partial L}{\partial \mathbf{g}_i^k} \frac{\partial \mathbf{g}_i^k}{\partial \mathbf{g}_i} = \sum_k \frac{\partial L}{\partial \mathbf{g}_i^k} \frac{\partial \mathcal{T}_k(\mathbf{g}_i)}{\partial \mathbf{g}_i}, \tag{8}$$

where the transformation $\mathcal{T}_k$ is defined as $\mathcal{T}_k(\mathbf{g}) = \mathbf{R}_k\mathbf{g} + \mathbf{t}_k$, and $\mathbf{R}_k$ is an orthogonal matrix. We compute the Jacobian matrix J_k, and Equation 8 can be further written as:

$$\frac{\partial L}{\partial \mathbf{g}_i} = \sum_k J_k^\top \frac{\partial L}{\partial \mathbf{g}_i^k} \tag{9}$$

Here, we map the gradients from the instance space back to the template space. These formulations enable end-to-end optimization of both the shared template and the per-instance offsets within a unified differentiable rendering framework.

Table 1: Overall comparison on synthetic and real datasets. We highlight the first, second, and third best results for each metric.

	Synthetic					Real				
	PSNR↑	SSIM↑	LPIPS↓	CPU Mem↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	CPU Mem↓	FPS↑
InstantNGP	33.1	0.977	0.105	3.8GB	33	22.2	0.639	0.392	4.6GB	56
3DGS	34.9	0.985	0.026	198.4MB	117	27.4	0.912	0.089	312.7MB	98
3DGS*	35.2	0.986	0.024	200.1MB	120	27.2	0.918	0.084	315.6MB	101
Splat&Replace	23.8	0.872	0.179	218.6MB	116	19.9	0.785	0.249	338.9MB	92
SOG	32.4	0.974	0.042	212.0MB	189	26.7	0.904	0.103	324.5MB	118
OctreeGS	35.1	0.985	0.016	31.2MB	141	27.6	0.915	0.083	53.2MB	106
CompactGS	34.8	0.982	0.028	23.9MB	196	26.9	0.906	0.101	42.3MB	113
Ours	35.2	0.983	0.026	20.4MB	204	27.8	0.913	0.089	37.9MB	128

5 Experiments

5.1 Dataset

We propose *ReScene*, a dataset containing repetitive and similar instances in both synthetic and real-world environments, comprising more than 20 diverse scenes on which we evaluate our method. The synthetic data consists of images collected from the public dataset like Replica Dataset [45] as well as images rendered from manually created 3D models. The real-world data are captured by drones flying at low altitude in outdoor environments. To validate robustness and scalability, our datasets span highly consistent man-made objects (e.g., chairs in a hall) and naturally diverse scenes with significant variations in size, color, shape, and external factors like lighting and shadows.

Figure 4 shows representative scenes spanning diverse scales and complexities. For visualization, image sizes are scaled; in experiments, all images are resized uniformly, as detailed in Section 5.2.

5.2 Implementation

Experimental Setup. We implemented our method using the PyTorch framework and wrote custom CUDA kernels for a differentiable instancing rasterizer. All experiments are conducted on a workstation running Ubuntu 24.04 with 16 GB of VRAM.

Data Preprocessing. For synthetic data, we construct scenes using Blender and render a total of 200 images per scene at a resolution of 1600×900. For real-world datasets, all input images are resized while maintaining their aspect ratio, with the image width standardized to 1080 pixels. All 3DGS-related training

iterations are set to 30000, while InstGS is trained for 10000 iterations. We set the densification start and end iterations to 500 and 30000.

5.3 Evaluation

Comparison Results. Our objective is to achieve high-frame-rate rendering with low draw call memory, which inherently represents a multi-objective optimization problem. To this end, we conduct comparative experiments across several specialized domains using state-of-the-art methods. We adopt 3D Gaussian Splatting (3DGS) as the baseline. We compare against Hierarchical 3DGS [18] (3DGS*) for large-scale scenes, while for instance-level redundancy reduction, we include Splat-and-Replace [48] (Splat&Replace), which shares a similar instance substitution concept with ours. Additionally, We evaluate against the most advanced compression method: Self-Organizing Gaussians [33] (SOG), compact representation method: OctreeGS [41] and CompactGS [21], as well as NeRF-based method: InstantNGP [34]. The overall performance across all datasets is summarized in Table 1, where we report results separately for synthetic and real-world scenes. The table results shows that advantages of our method are reflected in lower CPU memory consumption, faster rendering speed, while maintaining high visual quality. From a technical perspective, Lower memory consumption stems from a more compact and cleaner representation. Faster rendering speed is achieved through a GPU-based instanced renderer. High-quality visual results are attributed to the learned offsets. Detailed comparisons on individual scenes are presented in Table 2. Furthermore, We compare our method with Instant-NGP, 3DGS, Splat&Replace, and CompactGS. Figure 9 shows the visualization comparison.

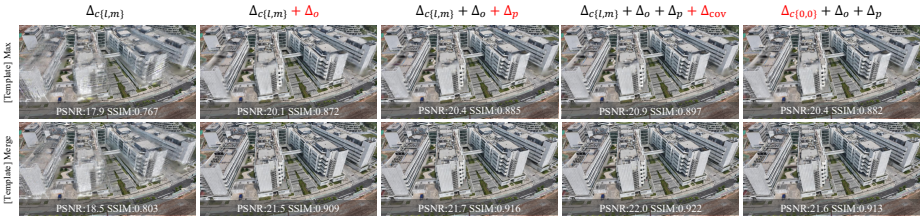

Fig. 5: Ablation studies on template initialization and offset choice. Using the Merge-based initialization preserves the appearance of all instances and yields superior visual quality. Employing the three offset types ($\Delta_{c\{0,0\}}, \Delta_o, \Delta_p$) achieves the best trade-off between data efficiency and visual fidelity. $c\{l, m\}$ denotes the color with SH coefficient of degree l, order m, and $c\{0, 0\}$ denotes the base color.

Instance Segmentation. Gaussian instancing relies on accurate and efficient instance segmentation. We first conducted comparative experiments between existing segmentation methods and our proposed segmentation module. We compared state-of-the-art methods, including LangSplat [39], SAGA [3], and OpenGaussian [52], evaluating instance segmentation performance in terms of

CUDA memory consumption, runtime, and mean accuracy (mAcc). Detailed results are presented in the supplementary material. The visual comparison between our method and SAGA is shown in Figure 6.

Table 2: Comparison of different methods in our teaser.

	PSNR↑	SSIM↑	LPIPS↓	Train ↓	CPU Mem↓	FPS↑
InstantNGP	29.61	0.945	0.152	8m36s	4.61GB	50
3DGS	35.18	0.986	0.025	28m48s	200.51 MB	115
Splat&Replace	23.28	0.866	0.184	+18m39s	216.89 MB	118
SOG	32.16	0.972	0.043	29m04s	211.33 MB	191
OctreeGS	35.07	0.984	0.015	28m48s	30.04 MB	139
CompactGS	25.53	0.937	0.054	24m43s	84.50 MB	171
Ours	35.12	0.981	0.027	+3m11s	41.24 MB	201

Multiple Similar-Instance Cases. Our method can also handle scenes containing multiple groups of similar instances, as shown in Figure 7. In a kitchen scene with twelve chairs, two tables, and four vases, we store instance relationships in a JSON file, where instances belonging to the same group share the same template ID. During optimization, all templates are concatenated, while the start index and length of each template are recorded to separate them within the CUDA kernel for individual instancing. During gradient computation, the same ordering is preserved to ensure correct backpropagation.

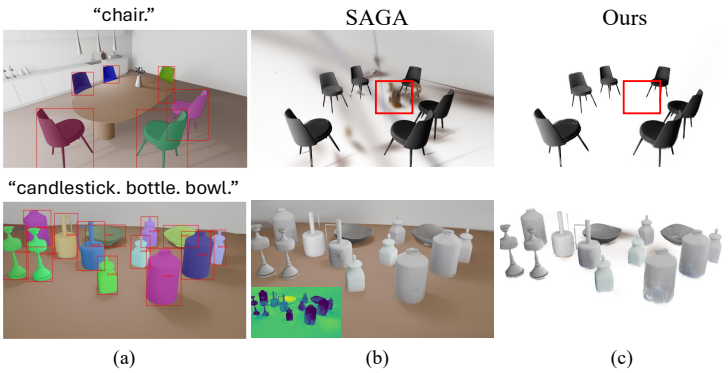

Fig. 6: Comparison of instance segmentation: (a) Grounding DINO generates bounding boxes and masks from text prompts. (b) SAGA suffers from noisy outputs, especially in the second scene. (c) Our method suppresses this noise, yielding cleaner segmentation.

5.4 Ablation Study and Discussion

Choice of Offsets. One might ask *why offsets are applied only to position, color, and opacity*. In theory, every Gaussian attribute can have its offset. In the extreme case where offsets are applied to all attributes, it becomes equivalent to deforming two entirely unrelated objects. Our experiments show that applying

Fig. 7: Support for multi-similar-instance scenarios. We use the prompt "table, chair, vase" to segment instances (blue, orange, and green). Then, we jointly optimize three templates and their offsets.

offsets to position (Δ_p), base color (Δ_c), and opacity (Δ_o) are sufficient to capture most inter-instance variation, achieving high visual quality with compact, lightweight offsets. Figure 5 shows visualizations and comparisons of applying different offset types to the template. Spherical harmonic offsets alone cannot capture geometric variations, while covariance offsets introduce excessive data growth. Fundamentally, Δ_p enables Gaussians to tightly align with the instance surface, Δ_o suppresses irrelevant Gaussians by making them invisible, and Δ_c captures color variations across different instances.

Initialization Strategy of Template. Beyond merging, alternative template initialization strategies exist, but are suboptimal. 1) Directly using a single well-trained 3DGS instance avoids re-optimization but suffers from arbitrary instance selection and mismatched Gaussian counts, causing artifacts. 2) Choosing the instance with the most Gaussians ("Max") ignores inter-instance variation, yielding blurred details, as shown in Figure 5, whereas merging better captures shared structure.

Initialization of the Position Offset. Initial position offsets Δ_p are set as the differences between the template and instances. As shown in Figure 8, compared to zero initialization, this leads to more accurate surface alignment within the same training budget, eliminating blurry, fog-like artifacts.

Capability Boundary. For objects with similar geometry and appearance, instGS can achieve very high reconstruction quality, as shown in Figures 7 9. Here, we further discuss whether the instancing mechanism of instGS remains effective when the Gaussian instances exhibit significant appearance differences or are even completely unrelated. First, consider the case where the geometry is similar but the appearance differs substantially. In principle, Δ_c and Δ_o can learn optimal solutions within a relatively small RGB and opacity space. Therefore, instGS can still achieve high-quality reconstruction and rendering under such conditions, as illustrated by the different Formula racing cars in Figure 1. Next, we consider an extreme scenario: two completely unrelated scenes. In this case, instGS still attempts to merge them into a shared template and processes them using the same pipeline. As shown in Figure 10, we treat the Truck and Train scenes from the Tanks and Temples Dataset [20] as two instances (despite their complete dissimilarity) and apply instGS to perform instanced rendering. The results show that, due to the large distribution discrepancy between the two

scenes, regions with sparse Gaussians cannot be reconstructed well. Although the opacity parameters attempt to learn near-zero values to make these regions fully transparent, ghosting artifacts still appear, which represents a typical failure case.

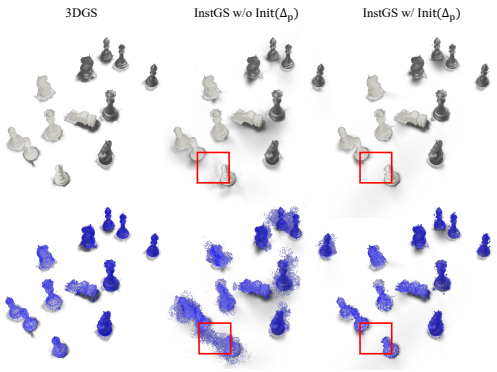

Fig. 8: Ablation study on the initialization of position offsets.

6 Conclusion

We have introduced InstGS, a Gaussian instancing framework that significantly improves the efficiency of 3D Gaussian Splatting for repetitive scenes. By leveraging a shared Gaussian template and learnable instance-specific offsets, InstGS reduces redundancy while preserving per-instance appearance variations. Our method enables high-quality, differentiable rendering at high frame rates and low memory footprint. Extensive experiments demonstrate that InstGS consistently outperforms existing baselines in both rendering efficiency and visual fidelity, making it particularly suitable for large-scale, editable 3D environments.

Limitation and future work. The advantage of our method scales positively with the number of repeated objects in the scene: the more repetitions and the larger their spatial coverage, the more pronounced our gains become. Consequently, our approach is inherently limited to such structured, repetitive scenes and is not a general acceleration technique. Nevertheless, we remain optimistic about the potential of instGS. for example, in scene editing and interaction, where users could populate large groups of objects by manipulating just a single instance. In the future, we will extend InstGS to scalable and editable customized instancing applications.

7 Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62502410) and the Guangdong Basic and Applied Basic Research Foundation (No. 2026A1515011138). All 3D models used in this work were obtained from publicly available model repositories for academic research purposes only.

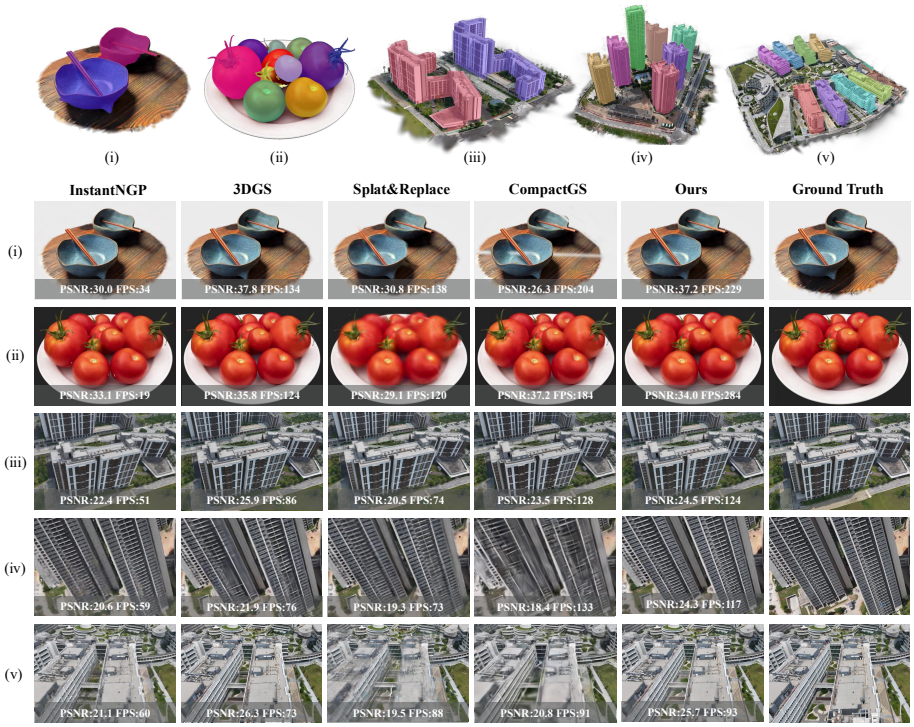

Fig. 9: Visualization comparison on the ReScene dataset. Our method consistently achieves the most impressive image quality and frame rates across diverse scenes.

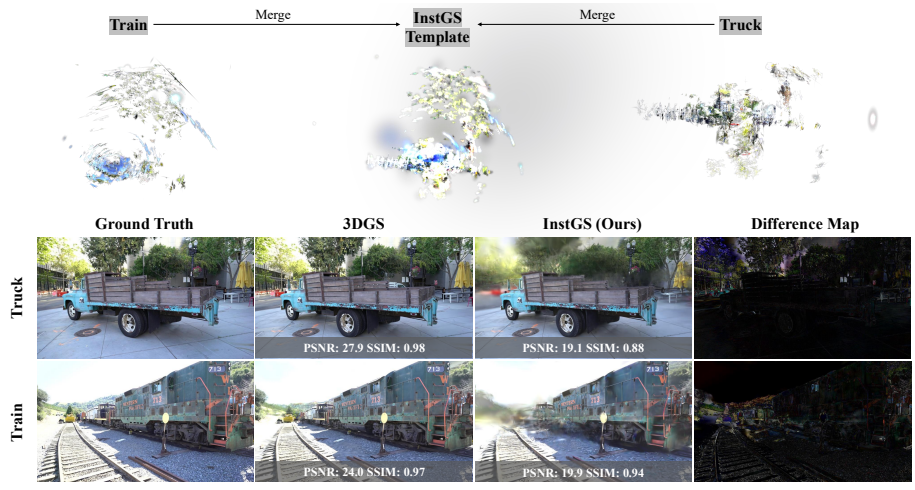

Fig. 10: Failure cases of InstGS. The difference map visualizes the discrepancy between the results of instGS and the ground truth.

References

1. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5855–5864 (2021)
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5470–5479 (2022)
3. Cen, J., Fang, J., Yang, C., Xie, L., Zhang, X., Shen, W., Tian, Q.: Segment Any 3D Gaussians. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 1971–1979 (2025)
4. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: HAC: Hash-Grid Assisted Context for 3D Gaussian Splatting Compression. In: European Conference on Computer Vision. pp. 422–438. Springer (2024)
5. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: HAC++: Towards 100x Compression of 3D Gaussian Splatting. Arxiv Preprint arxiv:2501.12255 (2025)
6. Cheng, K., Long, X., Yang, K., Yao, Y., Yin, W., Ma, Y., Wang, W., Chen, X.: GaussianPro: 3D Gaussian Splatting with Progressive Propagation. In: Forty-First International Conference on Machine Learning (2024)
7. Choy, J.G., Cha, G., Kee, H., Oh, S.: Unsupervised 3D Part Decomposition via Leveraged Gaussian Splatting. In: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2647–2652. IEEE (2024)
8. Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z.: LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. Advances in Neural Information Processing Systems **37**, 140138–140158 (2024)
9. Fang, G., Wang, B.: Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. In: European Conference on Computer Vision. pp. 165–181. Springer (2024)
10. Feng, K., Ma, Y., Wang, B., Qi, C., Chen, H., Chen, Q., Wang, Z.: Dit4edit: Diffusion Transformer for Image Editing. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 2969–2977 (2025)
11. Feng, K., Ma, Y., Zhang, X., Liu, B., Yuluo, Y., Zhang, Y., Liu, R., Liu, H., Qin, Z., Mo, S., Chen, Q., Wang, Z.: Follow-Your-Instruction: A Comprehensive MLLM Agent for World Data Synthesis (2025), `https://arxiv.org/abs/2508.05580`
12. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance Fields without Neural Networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5501–5510 (2022)
13. Guo, J., Ma, X., Fan, Y., Liu, H., Li, Q.: Semantic Gaussians: Open-Vocabulary Scene Understanding with 3D Gaussian Splatting (2024), `https://arxiv.org/Abs/2403.15624`
14. Haberl, J., Fleck, P., Arth, C.: Virtual Memory for 3D Gaussian Splatting. Arxiv Preprint arxiv:2506.19415 (2025)
15. Joseph, J., Amrutur, B., Bhatnagar, S.: Gradient-Driven 3D Segmentation and Affordance Transfer in Gaussian Splatting Using 2D Masks. Arxiv Preprint arxiv:2409.11681 (2024)
16. Kang, G., Nam, S., Sun, X., Khamis, S., Mohamed, A., Park, E.: iLRM: An Iterative Large 3D Reconstruction Model. Arxiv Preprint arxiv:2507.23277 (2025)

17. Kerbl, B., Kopanas, G., Leimkuehler, T., Drettakis, G.: 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Trans. Graph. **42**(4) (Jul 2023). https://doi.org/10.1145/3592433
18. Kerbl, B., Meuleman, A., Kopanas, G., Wimmer, M., Lanvin, A., Drettakis, G.: A Hierarchical 3D Gaussian Representation for Real-Time Rendering of Very Large Datasets. ACM Transactions on Graphics (TOG) **43**(4), 1–15 (2024)
19. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., Dollár, P., Girshick, R.: Segment Anything (2023), https://arxiv.org/Abs/2304.02643
20. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. ACM Transactions on Graphics **36**(4) (2017)
21. Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3D Gaussian Representation for Radiance Field. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21719–21728 (2024)
22. Li, C., Zhu, H., Chen, H., Zhang, J., Chen, T., Yang, S., Shao, S., Dong, W., Zhang, B.: HRGS: Hierarchical Gaussian Splatting for Memory-Efficient High-Resolution 3D Reconstruction. Arxiv Preprint arxiv:2506.14229 (2025)
23. Li, H., Wu, Y., Meng, J., Gao, Q., Zhang, Z., Wang, R., Zhang, J.: InstanceGaussian: Appearance-Semantic Joint Gaussian Representation for 3D Instance-Level Perception. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 14078–14088 (2025)
24. Li, Y., Ran, X., Xu, L., Lu, T., Yu, M., Wang, Z., Xiangli, Y., Lin, D., Dai, B.: Proc-GS: Procedural Building Generation for City Assembly with 3D Gaussians (2024), https://arxiv.org/abs/2412.07660
25. Lin, J., Li, Z., Tang, X., Liu, J., Liu, S., Liu, J., Lu, Y., Wu, X., Xu, S., Yan, Y.: VastGaussian: Vast 3D Gaussians for Large Scene Reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5166–5175 (2024)
26. Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Jiang, Q., Li, C., Yang, J., Su, H.: Grounding Dino: Marrying Dino with Grounded Pre-Training for Open-Set Object Detection. In: European Conference on Computer Vision. pp. 38–55. Springer (2024)
27. Liu, X., Wu, X., Wang, S., Li, Z., Kwong, S.: CompGS++: Compressed Gaussian Splatting for Static and Dynamic Scene Representation. Arxiv Preprint arxiv:2504.13022 (2025)
28. Liu, Y., Luo, C., Fan, L., Wang, N., Peng, J., Zhang, Z.: CityGaussian: Real-Time High-Quality Large-Scale Scene Rendering with Gaussians. In: European Conference on Computer Vision. pp. 265–282. Springer (2024)
29. Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 20654–20664 (2024)
30. Ma, Y., Feng, K., Hu, Z., Wang, X., Wang, Y., Zheng, M., He, X., Zhu, C., Liu, H., He, Y., et al.: Controllable Video Generation: A Survey. arXiv preprint arXiv:2507.16869 (2025)
31. Ma, Y., Feng, K., Zhang, X., Liu, H., Zhang, D.J., Xing, J., Zhang, Y., Yang, A., Wang, Z., Chen, Q.: Follow-Your-Creation: Empowering 4D Creation through Video Inpainting. arXiv preprint arXiv:2506.04590 (2025)
32. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. Communications of the ACM **65**(1), 99–106 (2021)

33. Morgenstern, W., Barthel, F., Hilsmann, A., Eisert, P.: Compact 3D Scene Representation via Self-Organizing Gaussian Grids. In: European Conference on Computer Vision. pp. 18–34. Springer (2024)
34. Müller, T., Evans, A., Schied, C., Keller, A.: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. ACM Transactions on Graphics (TOG) **41**(4), 1–15 (2022)
35. Navaneet, K., Meibodi, K.P., Koohpayegani, S.A., Pirsiavash, H.: Compact3D: Compressing Gaussian Splat Radiance Field Models with Vector Quantization. Arxiv Preprint arxiv:2311.18159 **2**(3) (2023)
36. Niedermayr, S., Stumpfegger, J., Westermann, R.: Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10349–10358 (2024)
37. Papantonakis, P., Kopanas, G., Kerbl, B., Lanvin, A., Drettakis, G.: Reducing the Memory Footprint of 3D Gaussian Splatting. Proceedings of the ACM on Computer Graphics and Interactive Techniques **7**(1), 1–17 (2024)
38. Piekenbrinck, J., Schmidt, C., Hermans, A., Vaskevicius, N., Linder, T., Leibe, B.: OpenSplat3D: Open-Vocabulary 3D Instance Segmentation Using Gaussian Splatting. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 5246–5255 (2025)
39. Qin, M., Li, W., Zhou, J., Wang, H., Pfister, H.: LangSplat: 3D Language Gaussian Splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 20051–20060 (2024)
40. Rematas, K., Liu, A., Srinivasan, P., Barron, J., Tagliasacchi, A., Funkhouser, T., Ferrari, V.: Urban Radiance Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12932–12942 (2022)
41. Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., Dai, B.: Octree-GS: Towards Consistent Real-Time Rendering with LOD-Structured 3D Gaussians. Arxiv Preprint arxiv:2403.17898 (2024)
42. Rusinkiewicz, S., Levoy, M.: Efficient Variants of the ICP Algorithm. In: Proceedings Third International Conference on 3D Digital Imaging and Modeling. pp. 145–152. IEEE (2001)
43. Rusu, R.B., Blodow, N., Beetz, M.: Fast Point Feature Histograms (FPFH) for 3D Registration. In: 2009 IEEE International Conference on Robotics and Automation. pp. 3212–3217. IEEE (2009)
44. Sha, J., Zhang, H., Huang, Q., Kou, G.: Modular Gaussian Splatting: Instance Decomposable Learning and Adaptive Rendering of 3D Scenes via Mixture of Experts. In: ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (Icassp). pp. 1–5. IEEE (2025)
45. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., Clarkson, A., Yan, M., Budge, B., Yan, Y., Pan, X., Yon, J., Zou, Y., Leon, K., Carter, N., Briales, J., Gillingham, T., Mueggler, E., Pesqueira, L., Savva, M., Batra, D., Strasdat, H.M., de Nardi, R., Goesele, M., Lovegrove, S., Newcombe, R.: The Replica Dataset: A Digital Replica of Indoor Spaces (2019), `https://arxiv.org/Abs/1906.05797`
46. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-NeRF: Scalable Large Scene Neural View Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8248–8258 (2022)
47. Turki, H., Ramanan, D., Satyanarayanan, M.: Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12922–12931 (2022)

48. Violante, N., Meuleman, A., Gauthier, A., Durand, F., Groueix, T., Drettakis, G.: Splat and Replace: 3D Reconstruction with Repetitive Elements. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. pp. 1–12 (2025)
49. Wang, R., Lohmeyer, Q., Meboldt, M., Tang, S.: DeGauss: Dynamic-Static Decomposition with Gaussian Splatting for Distractor-Free 3D Reconstruction. In: TCCV 2025 Workshop on Wild 3D: 3D Modeling, Reconstruction, and Generation in the Wild (2025)
50. Wang, T., Yu, Z., Xu, Y.: TC-GS: Tri-Plane Based Compression for 3D Gaussian Splatting. Arxiv Preprint arxiv:2503.20221 (2025)
51. Wang, Z., Su, M., Au, H., Li, Y., Cao, X., Pan, C., Chen, Y., Wang, G.: HUG: Hierarchical Urban Gaussian Splatting with Block-Based Reconstruction. Arxiv Preprint arxiv:2504.16606 (2025)
52. Wu, Y., Meng, J., Li, H., Wu, C., Shi, Y., Cheng, X., Zhao, C., Feng, H., Ding, E., Wang, J.: OpenGaussian: Towards Point-Level 3D Gaussian-Based Open Vocabulary Understanding. Advances in Neural Information Processing Systems **37**, 19114–19138 (2024)
53. Yan, M., Zhang, J., Zhu, Y., Wang, H.: MaskClustering: View Consensus Based Mask Graph Clustering for Open-Vocabulary 3D Instance Segmentation (2024), `https://arxiv.org/Abs/2401.07745`
54. Yang, Q., Feng, M., Wu, Z., Dong, W., Wu, F., Wang, Y., Mian, A.: Hierarchical Gaussian Mixture Model Splatting for Efficient and Part Controllable 3D Generation. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 11104–11114 (2025)
55. Yang, Z., Yang, B., Dong, W., Cao, C., Cui, L., Ma, Y., Cui, Z., Bao, H.: InstaScene: Towards Complete 3D Instance Decomposition and Reconstruction from Cluttered Scenes (2025), `https://arxiv.org/Abs/2507.08416`
56. Yang, Z., Gong, B., Chen, K.: LOD-GS: Level-of-Detail-Sensitive 3D Gaussian Splatting for Detail Conserved Anti-Aliasing. Arxiv Preprint arxiv:2507.00554 (2025)
57. Ye, M., Danelljan, M., Yu, F., Ke, L.: Gaussian Grouping: Segment and Edit Anything in 3D Scenes. In: European Conference on Computer Vision. pp. 162–179. Springer (2024)
58. Zhang, X., Ge, X., Xu, T., He, D., Wang, Y., Qin, H., Lu, G., Geng, J., Zhang, J.: GaussianImage: 1000 FPS Image Representation and Compression by 2D Gaussian Splatting. In: European Conference on Computer Vision. pp. 327–345. Springer (2024)