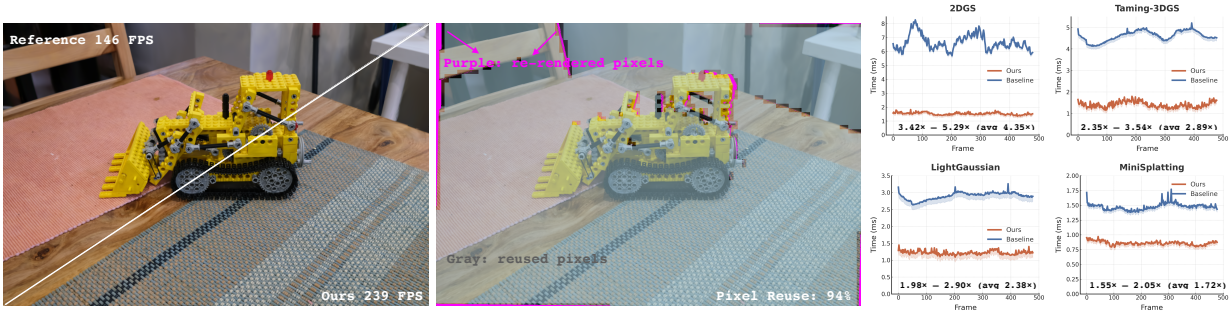
This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2026.3679889

# GSReuse: Temporally Adaptive Screen-Space Reuse for Accelerating 3D Gaussian Splatting

Chengzhi Tao, Yiyang Sun, Jie Guo, Tao Zhang, Letian Huang, Junqiu Zhu, Daoheng Wang and Yanwen Guo




(a) Full scene (b) Temporal Reuse Map (c) Runtime Comparison

Fig. 1: We propose *GSReuse*, a lightweight screen-space acceleration strategy that significantly improves the rendering speed of 3D Gaussian Splatting (3DGS) by adaptively reusing tiles from historical frames. (a) The baseline rendering without temporal reuse achieves only 146 FPS, while (b) our method attains 239 FPS by reusing previous-frame results in gray tiles and re-rendering only a small number of invalid pixels highlighted in purple. Since the strategy operates in screen space, it can be seamlessly integrated into existing 2D/3DGS pipelines and combined with various compression or acceleration techniques, such as LightGaussian, Taming-3DGS, and Mini-Splatting. (c) Runtime comparisons across representative pipelines demonstrate that our method (orange) consistently delivers stable and significant speedups over the baseline (blue) while maintaining comparable visual quality. These results highlight GSReuse as a general solution for accelerating real-time, high-quality rendering.

**Abstract**—Recent advances in 3D Gaussian Splatting (3DGS) have enabled real-time, high-fidelity novel view synthesis. However, rendering each frame independently in a video sequence leads to redundant computations, especially when adjacent frames share significant visual overlaps. This inefficiency is particularly problematic in VR applications, where high frame rates and stereoscopic rendering amplify the per-frame cost. Existing frame interpolation or reuse strategies typically rely on image-domain information and are thus not directly applicable to 3DGS rendering, which is fundamentally point-based. To address this runtime inefficiency, we propose GSReuse, a lightweight and drop-in accelerator that speeds up 3DGS rendering by reusing computations across consecutive frames. GSReuse operates in screen space and introduces only minimal modifications to existing 3DGS rendering pipelines. It also eliminates the need for retraining scene representations. Given the rendered image, depth map, and camera parameters of the current frame, GSReuse estimates reliable Gaussian splatting motion vectors for all pixels and warps reusable contents to the new view. A tile-based filtering and masking strategy is then applied to determine which regions can be safely reused, allowing the 3DGS renderer to skip redundant rendering operations. We evaluate GSReuse on multiple benchmark datasets, showing that GSReuse significantly improves rendering, while maintaining high visual fidelity. Compared to state-of-the-art video frame reuse/generation methods, GSReuse delivers better image quality with much lower latency, facilitating practical deployment of 3DGS in VR applications.

**Index Terms**—Rendering, gaussian splatting, acceleration

## 1 Introduction

Neural rendering has recently made significant strides, with Neural Radiance Fields (NeRF) [46] and 3DGS [29] notably showcasing the ability to generate high-quality, photo-realistic images from novel views. While NeRF demonstrates impressive quality in novel view synthesis, its limited training and rendering speed restricts its applicability in real-time scenarios. In contrast, 3DGS achieves a favorable balance between quality and efficiency by representing the scene with explicit 3D Gaussians and leveraging GPU rasterization, making it attractive for interactive and real-time applications.

Despite these advantages, achieving real-time performance with 3DGS remains challenging in many scenarios , especially for large-scale urban scenes [40,41]. A rapidly growing trend of research addresses this issue by aggressively reducing the number of Gaussians to accelerate rendering [8, 9, 16, 21, 34, 43, 47, 49, 50, 52]. While these techniques make great progress, they often struggle to preserve high-frequency details and necessitates additional re-training for existing 3DGS scenes. Meanwhile, 3DGS is being rapidly adopted in VR/XR for stereoscopic rendering and navigation, motivating a growing line of systems and rendering approaches [7,14,58,74]. In particular, VR rendering imposes strict requirements on latency and refresh rates (typically above 90 FPS), and demands stereo rendering for both eyes, effectively doubling the workload. Beyond rendering performance, 3DGS is also enabling a

- *Chengzhi Tao, Yiyang Sun, Jie Guo, Tao Zhang, Letian Huang, Daoheng Wang and Yanwen Guo are with the State Key Lab for Novel Software Technology, Nanjing University, Nanjing 210093, China. (E-mail: tcz_tao@smail.nju.edu.cn; 502023330052@smail.nju.edu.cn; guojie@nju.edu.cn; 191240073@smail.nju.edu.cn; lthuang@smail.nju.edu.cn; daohengwang@smail.nju.edu.cn; ywguo@nju.edu.cn). (Corresponding authors: Jie Guo and Yanwen Guo.)*
- *Junqiu Zhu is with Shandong University. E-mail: zhujunqiu@mail.sdu.edu.cn.*
- *Supported by the Fundamental Research Funds for the Central Universities.*

*Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx*

broad range of VR applications, including telepresence with lightweight transmission [37], environment stylization along the reality-virtuality continuum [72], interactive and physics-aware VR systems [27,44,56], and VR-guided content creation [17]. These trends make frame-to-frame reuse particularly appealing for 3DGS in VR, where consecutive views are strongly correlated.

In this paper, we propose *GSReuse*, a lightweight and drop-in accelerator that improves 3DGS rendering performance from an entirely different perspective, remarkably boosting the rendering frame rate for a wide range of 3DGS scenes. Inspired by the frame generation methods for videos [23,25,32] and temporal reuse methods for conventional mesh rendering [55,65], we exploit spatial and temporal coherence between frames in 3DGS rendering. Built upon existing 3DGS rendering pipelines, we develop a tile-based temporal reuse framework that facilitates the adaptive selection of reusable tiles for any 3DGS scene. These reusable tiles can safely reuse information from previous frames by warping with *Gaussian splatting motion vectors* (GS-MVs), while remaining tiles, which only constitute a minimal fraction of the frame, are re-rendered. To further improve the performance, we introduce a *mask generation* strategy to increase the number of reusable tiles and also design an *pixel filtering* technique to reduces artifacts in the filled pixels during frame warping. As a screen-space method, our GSReuse can be seamlessly integrated into existing 3DGS rendering pipelines, without requiring post-hoc training or auxiliary data.

It should be noted that our proposed acceleration method is orthogonal to existing approaches, such as LightGaussian [8], Taming-3DGS [43], and Mini-Splatting [9]. Moreover, since our method leverages inter-frame coherence, it not only enables efficient reuse between adjacent frames but can also be naturally extended to *multi-frame interpolation*, further enhancing both the smoothness and efficiency of rendering. We integrate our method into various 3DGS rendering pipelines and evaluate its effectiveness across multiple benchmark scenes. In summary, GSReuse introduces a temporal-reuse perspective that expands the paradigm of 3DGS acceleration, opening new possibilities for real-time neural rendering at larger scales and under more constrained hardware.

## 2 Related Work

In this section, we provide a concise overview of acceleration techniques for 3D Gaussian Splatting, followed by a discussion of temporal reuse in computer graphics and vision.

### 2.1 Acceleration for 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) [29] is a recent explicit 3D representation that has achieved real-time rendering of complex scenes. However, the method still suffers from unnecessary overhead and faces challenges in rendering speed, particularly in resource-constrained settings. Many existing methods [8,9,16,21,34,43,47,49,50,52] focus on reducing the overall model size, e.g., the number of 3D Gaussians, but these methods often trade off rendering quality for performance. Several methods for city-scale scenes [6,30,40] employ Level-of-Detail (LOD) [42] to mitigate runtime costs caused by the excessive number of Gaussians. Nevertheless, these approaches still improve rendering efficiency by decreasing the number of Gaussians in distant areas.

Some recent researches focus on optimizing the original rasterization pipeline of 3DGS while preserving the full Gaussian count. AdR-Gaussian [60] proposes early culling based on the axis-aligned bounding box (AABB) with adaptive radius, which accelerates 3DGS without rendering quality loss. Speedy-Splat [20] and FlashGS [10] further analyze the performance of each function, and extend AABB to precisely identify the set of tiles intersected by each Gaussian. Hou et al. [22] employ order-independent weighted-sum rendering to simplify volumetric rendering, thereby reducing the computational overhead associated with sorting. To meet the high-performance demands of virtual reality (VR) systems, Tu et al. [57] and Franke et al. [15] integrate 3DGS with foveated rendering [18,53]. And certain architectural approaches [33,38,68] enable real-time rendering on mobile devices through dedicated hardware-accelerated units.

While the above methods optimize the 3DGS pipeline via algorithmic or architectural approaches, some recent works have begun to explore temporal redundancy within the 3DGS framework. Lumina [12] proposes a system that exploits inter-frame coherence by sharing sorting results and caching radiance from significant Gaussians. Although effective, Lumina requires dedicated hardware support. While originally designed for NeRF acceleration, Cicero [13] introduces sparse radiance warping and memory-centric rendering strategies that can be adapted to 3DGS. However, its pipeline relies on precomputed meshes for accurate warping. In contrast, our method performs real-time screen-space extrapolation directly in the 3DGS pipeline without requiring mesh supervision or architectural changes.

### 2.2 Temporal Reuse for Rendering

Temporal reuse is a technique that leverages rendering results from previous frames to accelerate rendering [48,55,57,59]. This acceleration strategy is also widely used in the industry, such as DLSS [39, 51], FSR [1,2] and XeSS [26]. These methods typically rely on deep neural networks to reconstruct high-quality frames and are heavily optimized for modern GPUs with dedicated inference hardware. A few early methods [31, 35, 45, 66] use warping in real-time rendering. However, they struggle with disocclusion areas for the required information is unavailable. Reinert et al. [54] utilize geometric proxies to fill disocclusion regions, but they rely on pre-computed geometry and produce low-quality results due to the use of low-polygon meshes.

With the advent of deep learning, the trend in the frame interpolation methods has been spearheaded by networks. Briedis et al. [4,5] propose using optical flows to generate intermediate frames conditioned solely on given corresponding G-buffers. Some video interpolation methods in computer vision [25, 28, 32, 71] also generate plausible results. Nevertheless, these interpolation methods are limited by requiring both preceding and subsequent frames.

To overcome this limitation, frame extrapolation methods are developed to synthesize new frames exclusively from historical frames. ExtraNet [19] and Zeng et al. [70] propose motion vectors with neural networks to handle disocclusion areas. Wu et al. [63] extend these approaches by introducing learnable motion vectors. In parallel, Feng et al. [11] demonstrates that sparse but accurate motion flows, derived from LiDAR and IMU data, can be used for fast video enhancement via patch-based warping. Mob-FGSR [67] employs warping techniques for frame extrapolation, but still struggles with significant disocclusion areas. Wu et al. [61] present a G-buffer free extrapolation method for real-time rendering. However, these techniques are exclusively applicable to traditional mesh-based real-time rendering pipelines. Video extrapolation methods [23, 62], while not constrained by the scene representation, typically produce significantly inferior quality and performance, making them generally unsuitable for real-time rendering applications.

Our GSReuse effectively achieves Gaussian-based frame extrapolation, accelerating 3DGS while simultaneously handling disocclusion areas. It is a lightweight strategy that operates entirely in screen space and can be integrated directly into the 3DGS rendering pipeline without requiring neural inference or specialized hardware support. Unlike dynamic scene approaches like Zhao et al. [73], our method primarily focuses on performance enhancement.

## 3 Problem Formulation

3D Gaussian Splatting (3DGS) [29] is an efficient differentiable rendering framework that represents scenes as a collection of anisotropic 3D Gaussian primitives. Each Gaussian $\mathcal{G}_i$ is parameterized by a mean position $\mu_i \in \mathbb{R}^3$, and a covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$, which is constructed from a scaling matrix $\mathbf{S}_i \in \mathbb{R}^{3\times3}$ and a rotation matrix $\mathbf{R}_i \in \mathrm{SO}(3)$ as $\Sigma_i = \mathbf{R}_i\mathbf{S}_i\mathbf{S}_i^\top\mathbf{R}_i^\top$. In addition, each Gaussian carries an opacity parameter $\sigma_i \in [0,1]$, and a view-dependent appearance $\mathbf{c}_i(\mathbf{d}) : S^2 \to \mathbb{R}^3$, represented by spherical harmonics, where $\mathbf{d}$ denotes the viewing direction.

As illustrated in Figure 2, the forward rendering pipeline of 3DGS can be broadly subdivided into four sequential stages: *splatting*, *tiling*, *sorting*, and *compositing*.

- *Splatting*: Each visible 3D Gaussian primitives $\mathcal{G}_i^{3D}$ is transformed

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2026.3679889

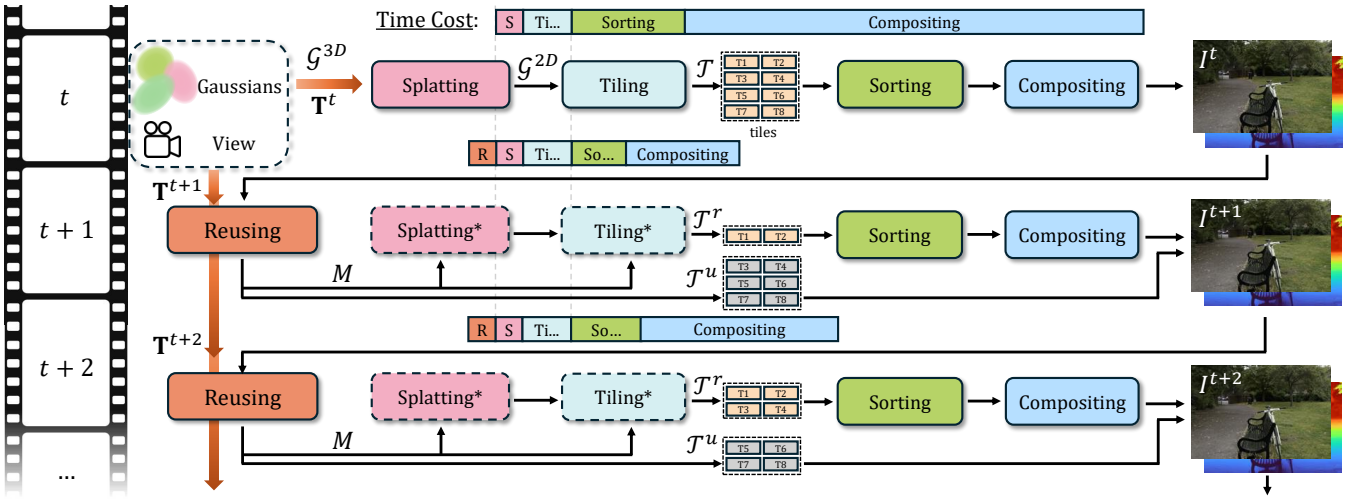



Fig. 2: Integration of our GSReuse method into an existing 3DGS rendering pipeline. Most tiles ($\mathcal{T}^u$) of frame $t+1$ are warped and reused from frame $t$, thus dramatically reducing the computational time cost of subsequent sorting and composition stages. To improve the quality of future reuse, pixels filtered during the generation of frame $t+1$ are excluded before producing frame $t+2$, resulting in a slightly increased number of tiles that need to be re-rendered.

into a screen-space 2D Gaussian $\mathcal{G}_i^{\text{2D}}$ according to its position, covariance, and the current camera parameters:

$$\mathcal{G}_i^{\text{3D}} \xrightarrow{\text{Proj}} \mathcal{G}_i^{\text{2D}}. \tag{1}$$

- *Tiling*: Each 2D Gaussian $\mathcal{G}_i^{\text{2D}}$ is assigned to a set of screen-space tiles $\mathcal{T}$ that it spatially overlaps:

$$\forall \mathcal{G}_i^{\text{2D}},\ \mathcal{T}(\mathcal{G}_i^{\text{2D}}) = \{\tau_k \mid \tau_k \cap \text{supp}(\mathcal{G}_i^{\text{2D}}) \neq \emptyset\}. \tag{2}$$

- *Sorting*: All 2D Gaussians within each tile are re-organized according to their depth values $z$, ensuring correct front-to-back blending:

$$\{\mathcal{G}_i^{\text{2D}}\} \xrightarrow{z_i} \{\mathcal{G}_{(1)}^{\text{2D}}, ..., \mathcal{G}_{(m)}^{\text{2D}}\} \text{ s.t. } z_{(1)} \leq \cdots \leq z_{(m)}. \tag{3}$$

- *Compositing*: The color of each pixel $C(\mathbf{p})$ is determined by the order-dependent blending of all 2D Gaussians covering this pixel:

$$C(\mathbf{p}) = \sum_{i \in O(\mathbf{p})} \mathbf{c}_i \sigma_i \mathcal{G}_i^{\text{2D}}(\mathbf{p}) \prod_{j=1}^{i-1} (1 - \sigma_j \mathcal{G}_j^{\text{2D}}(\mathbf{p})), \tag{4}$$

where $O(\mathbf{p})$ denotes the set of Gaussians that influence pixel $\mathbf{p}$. $\mathbf{c}_i$ and $\sigma_i$ are the color and opacity of the $i$-th Gaussian.

In the forward rendering pipeline of 3DGS, the sorting and compositing stages incur most of the computational cost, as demonstrated in the upper part of Figure 2. Our experiments show that splatting comprises roughly 2% of total rendering time, tiling 8%, sorting 20%, and compositing, the most time-consuming stage, accounts for nearly 70%. To optimize performance, our objective is to reduce the overall rendering time by minimizing the workload of the most expensive stages—sorting and compositing—through adaptively and automatically reusing previously rendered contents.

Mathematically, we formulate the problem as follows: Given a camera view at time step $t$ with its corresponding rendered frame $\mathbf{I}^t$ and depth map $\mathbf{Z}^t$, as well as a set of future camera views $\mathbf{T}^{t+n}$ for $n = 1, 2, \ldots$ that to be rendered, our method attempts to generate the corresponding new frames $\mathbf{I}^{t+n}$ and depth maps $\mathbf{Z}^{t+n}$ by reusing contents from previous frames (i.e., $\mathbf{I}^t$ and $\mathbf{Z}^t$). Here $n$ represents the number of intermediate frames to be generated after each fully rendered keyframe.

To achieve our goal and accelerate 3DGS rendering, we introduce an additional *reusing* stage prior to the standard rendering pipeline at time $t+1$ (see the lower part of Figure 2). This stage takes the RGB and depth maps from the $t$-th frame as input and produces a set of reusable tiles $\mathcal{T}^u$, along with a tile-level mask $\mathbf{M}$. The mask $\mathbf{M}$ is then propagated into the splatting and tiling stages, where it influences the resulting tile organization. Consequently, during the sorting and compositing stages, the renderer only needs to process the remaining non-reused tiles $\mathcal{T}^r$.

Unlike most previous 3DGS acceleration approaches, our method require little modification to 3D Gaussian representations. For instance, splatting-stage optimizations frequently involve filtering [69] or replacing Gaussians with negligible projected influence [15], while tiling-stage improvements often rely on approximating the screen coverage of individual 2D Gaussians [10]. These object-space methods inherently require per-Gaussian processing, and their computational cost scales with the number of Gaussians in the scene. In contrast, our method operates entirely in screen space, making its performance improvement independent of scene complexity. This makes our technique particularly attractive for real-time applications or large-scale scenes where object-space optimizations become increasingly expensive.

In summary, our acceleration method offers the following key advantages:

- *Plug-and-play*: The method offers a purely inference-time solution, eliminating the need for retraining or fine-tuning the 3DGS scene.
- *Scene-agnostic efficiency*: It is largely independent of scene complexity and the number of Gaussians, exhibiting consistent performance improvement across a wide range of tested scenarios.
- *Incremental rendering*: It enables the generation of multiple reused frames from a single fully rendered keyframe.
- *Non-intrusive integration*: It incurs little modification to the internal rendering pipeline of 3DGS, making it easy to integrate into existing 3DGS rendering frameworks.

## 4 Method

### 4.1 Overview

Given a 3DGS scene, our goal is to quickly generate the frame from time $t+1$ to $t+n$ by adaptively reusing information from the rendered frame at time $t$. First, we compute a set of Gaussian splatting motion vectors based on the $t$-th frame's depth map and the relative camera transformation between the two frames (Section 4.2). Then, we design a tile-based reusing scheme with a filtering process and a tile-wise mask

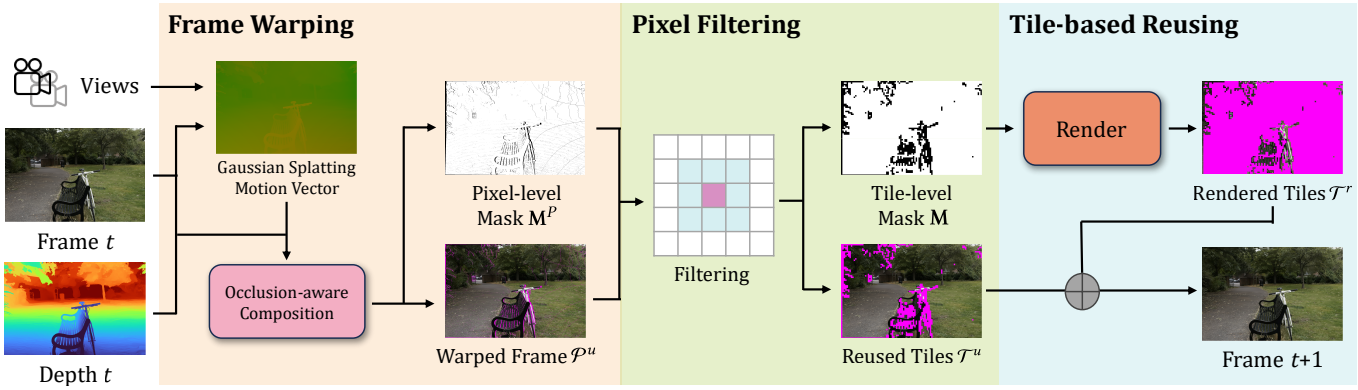



Fig. 3: Overall pipeline of our method. It involves frame warping based on Gaussian splatting motion vectors, occlusion-aware composition, pixel filtering, and tile-based reusing.

to identify and safely skip unnecessary tiles (Section 4.3). We further incorporate an occlusion-aware composition strategy into our method to address occlusion conflicts (Section 4.3.4). The overall pipeline of our method is illustrated in Figure 3.

## 4.2 Gaussian Splatting Motion Vector

*Depth Extraction.* Our approach for depth estimation aggregates contributions from overlapping Gaussians through alpha-weighted averaging:

$$\bar{z}(\mathbf{p}) = \frac{\sum_{i\in O(\mathbf{p})} \alpha_i(\mathbf{p}) z_i}{\sum_{i\in O(\mathbf{p})} \alpha_i(\mathbf{p})}, \tag{5}$$

where $z_i$ denotes the camera-space depth of the $i$-th Gaussian's center and $O(\mathbf{p})$ denotes the set of Gaussians overlapping pixel $\mathbf{p}$ sorted by depth.

*Gaussian Splatting Motion Vector.* We compute per-pixel GS-MV through geometric transformations between consecutive frames. Given the depth map $\mathbf{Z}^t$ from the $t$-th frame, we first reconstruct the 3D position $\mathbf{P}^t_w$ in world coordinates of each pixel $\mathbf{p}^t_j = (x^t_j, y^t_j)^\top$ by back-projecting it using the $t$-th frame's camera parameters: intrinsics $\mathbf{K}^t$ and extrinsics $\mathbf{T}^t$:

$$\mathbf{P}^t_w = \text{Proj}^{-1}(\mathbf{p}^t_j, z^t_j; \mathbf{K}^t, \mathbf{T}^t), \tag{6}$$

where $\text{Proj}^{-1}(\cdot)$ denotes the inverse projection operator parameterized by the $t$-th frame's camera intrinsics and extrinsics. Subsequently, we reproject this world-space position into the $t+1$-th frame's camera space using the $t+1$-th camera pose $(\mathbf{K}^{t+1}, \mathbf{T}^{t+1})$ and project it back to screen coordinates:

$$\mathbf{p}^{t+1}_j = \text{Proj}(\mathbf{P}^t_w; \mathbf{K}^{t+1}, \mathbf{T}^{t+1}), \tag{7}$$

where $\text{Proj}(\cdot)$ represents the perspective projection using $t+1$-th frame parameters. The GS-MV for pixel $\mathbf{p}_j$ is then computed as the difference between the reprojected screen-space position and the original pixel location:

$$\mathbf{m}_j = \mathbf{p}^t_j - \mathbf{p}^{t+1}_j \in \mathbb{R}^2. \tag{8}$$

The GS-MV describes the displacement of pixel $\mathbf{p}^t_j$ from its position in the previous frame to its corresponding location in the new frame.

## 4.3 Tile-based Temporal Reuse

Current rendering pipelines of 3DGS are mostly organized in a tile-based manner. Due to the *Tiling* stage mentioned previously, only a limited number of Gaussians are considered for each tile. This eliminates the need for sorting and processing all Gaussians once only a single pixel is required to render. This restricts the rendering to a tile level, significantly reducing the efficiency of reuse for our method. The design of our tile-based temporal reuse method aligns with the rendering configuration of existing 3DGS pipelines and avoids unnecessary computation. The details of this approach is summarized in Algorithm 1.

### 4.3.1 Tile-based Warping

Using GS-MVs, we can directly warp corresponding pixels from the $t$-th frame to the $t+1$-th frame. These pixels can potentially be reused without rendering, resulting in a warped image $\mathcal{P}^u$, along with a pixel-level mask $\mathbf{M}^P$ that indicates which regions are eligible for reuse. For pixels in the $t+1$-th frame lacking valid information warped from the $t$-th frame, traditional 3DGS pipelines are still triggered to perform the rendering.

For most 3DGS pipelines, rendering is organized at the tile level: all Gaussian primitives are initially assigned and sorted within each $K \times K$ tile (typically $K = 16$). This sorting process occurs irrespective of the number of pixels within the tile that actually require rendering. To mitigate the computational overhead, we propose a tile-based warping strategy. By performing the re-rendering operation at the tile level, rather than assessing rendering needs on a per-pixel basis, we reduce the number of tiles containing only a few invalid pixels (i.e., those requiring rendering). This, in turn, minimizes unnecessary sorting and compositing related to 3D Gaussians within those tiles.

### 4.3.2 Mask Generation

As depicted in Figure 3, the warped image $\mathcal{P}^u$ exhibits numerous scattered invalid pixels. Consequently, the corresponding tiles would require rendering, thereby substantially diminishing the proportion of reusable tiles. Given a smooth scene, the presence of scattered invalid pixels is unlikely to correlate with significant photometric discontinuities. After warping the previous frame to the current view, the resulting image often contains scattered empty pixels due to disocclusion or depth mismatch. Considering this two facts, we propose a pixel filtering method that utilizes the neighboring pixel information for inpainting invalid pixels. This method maximizes the proportion of reusable tiles, thereby enhancing the overall efficiency of the system.

After warping, we have the pixel-level mask of valid pixels $\mathbf{M}^P$, where $\mathbf{M}^P(\mathbf{p}) = 1$ if the pixel at location $\mathbf{p}$ is valid (i.e., has a defined color and depth), and $\mathbf{M}^P(\mathbf{p}) = 0$ otherwise. This mask captures the distribution of valid and missing pixels after warping. To identify scattered invalid pixels that are sufficiently surrounded by valid regions and are therefore likely to be safely filled, we apply a morphological closing operation to the mask:

$$\mathbf{M}^P_{\text{fill}} = (\mathbf{M}^P \oplus \mathcal{S}) \ominus \mathcal{S} - \mathbf{M}^P, \tag{9}$$

where $\oplus$ and $\ominus$ denote dilation and erosion, respectively. Instead of using a conventional $3 \times 3$ square structuring element, we adopt a cross-shaped kernel $\mathcal{S}$ (i.e., a 4-connected pattern). This more conservative design reduces the risk of falsely marking isolated invalid pixels as fillable and helps suppress spurious artifacts caused by overly permissive neighborhood aggregation. The resulting difference mask $\mathbf{M}^P_{\text{fill}}$ highlights sparse invalid pixels that have sufficient surrounding context to be reliably reconstructed.

With the updated mask $\mathbf{M}^P_{\text{closed}} = \mathbf{M}^P + \mathbf{M}^P_{\text{fill}}$, we further construct a tile-level validity mask to guide reuse decisions in the 3DGS rendering pipeline. For each tile, we inspect the validity of all pixels within its region. If all pixels inside the tile satisfy $\mathbf{M}^P_{\text{closed}}(\mathbf{p}) = 1$, we mark the tile as valid, indicating that it can be skipped safely during rendering. Conversely, if any pixel within the tile remains invalid, its mask value is set to 0 (invalid), and the tile will be rendered as usual. The generated mask $\mathbf{M}$ is passed to the original 3DGS rendering pipeline, specifically affecting the splatting and tiling stages (Figure 2). In the splatting stage, after determining the number of tiles each Gaussian covers ($N_{\text{cover}}$), the algorithm calculates the number of reusable tiles ($N_{\text{reuse}}$) and forwards $N_{\text{cover}} - N_{\text{reuse}}$ to the prefix sum computation for subsequent rendering steps. In the tiling stage, reusable tiles are skipped during key-value pair generation, so only tiles with mask value 0 undergo the sorting and compositing stages, thereby significantly reducing the workload of these time-consuming steps.

#### 4.3.3 Pixel Filtering

The above mask generation step identifies which pixels can be reliably reused, but their depth and color values are still undefined. To complete these pixels, we perform a two-step filtering process that ensures foreground-background awareness. We first complete the depth map for the invalid pixels. For each newly added pixel $\mathbf{p}$ where $\mathbf{M}^P_{\text{fill}}(\mathbf{p}) = 1$, we estimate a new depth value $z(\mathbf{p})$ by selecting the maximum depth among its valid neighbors:

$$z(\mathbf{p}) = \max_{\mathbf{q}\in\mathcal{N}(\mathbf{p})} z(\mathbf{q}), \quad \text{where } \mathbf{M}^P(\mathbf{q}) = 1, \tag{10}$$

where $\mathcal{N}(\mathbf{p})$ denotes the set of neighboring pixels around $\mathbf{p}$. This choice preserves a conservative estimate of scene geometry, assuming that the newly filled pixel belongs to a background surface previously occluded by foreground content.

After hole filling in the depth map as described above, we perform foreground-aware color completion to avoid blending foreground and background colors when inpainting invalid pixels. For each such invalid pixel $\mathbf{p}$, we apply a depth-guided bilateral filter over its spatial neighborhood $\mathcal{N}(\mathbf{p})$, using both spatial and depth similarity to compute the filled color $\mathbf{C}(\mathbf{p})$:

$$\mathbf{C}(\mathbf{p}) = \frac{1}{W(\mathbf{p})} \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{q}) \cdot \mathbf{C}(\mathbf{q}) \cdot \mathbf{M}^P(\mathbf{q}) \tag{11}$$

where $w(\mathbf{p},\mathbf{q}) = \exp\left(-\frac{\|\mathbf{p}-\mathbf{q}\|^2}{2\sigma_s^2} - \frac{|z(\mathbf{p})-z(\mathbf{q})|^2}{2\sigma_d^2}\right)$ is the bilateral weights, $\mathbf{C}(\cdot)$ denotes the color of a pixel, $\sigma_s$ and $\sigma_d$ control the spatial and depth sensitivity, respectively. $\mathbf{M}^P(\mathbf{q}) = 1$ ensures that only valid source pixels are used. $W(\mathbf{p})$ is the normalization term: $W(\mathbf{p}) = \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{q}) \cdot \mathbf{M}^P(\mathbf{q})$. This bilateral filter respects both spatial proximity and depth continuity, effectively preventing color bleeding across depth discontinuities (e.g., object boundaries). As a result, invalid pixels near foreground objects are filled using only nearby pixels with similar depth, preserving sharp edges and avoiding background contamination.

By treating these scattered pixels as reusable, we prevent the unnecessary rendering of entire tiles, thereby significantly increasing the number of reusable tiles (as shown in Figure 6). This filtering method exploits the spatial coherence inherent in 3DGS and reduces the overhead associated with tile-level sorting within the rendering pipeline.

#### 4.3.4 Occlusion-aware Composition

When the view changes, multiple pixels $\{\mathbf{p}^t_j\}$ of the $i$-th frame may map to the same pixel position $\mathbf{p}^{t+1}$ of the $i+1$-th frame, leading to occlusion conflicts where background content may overwrite foreground. To resolve this, we implement an occlusion-aware depth testing mechanism that prioritizes pixels closer to the camera. For each reprojected pixel, we compute its depth in the current view and maintain a depth buffer $\mathbf{Z}_{\text{rep}} \in \mathbb{R}^{H\times W}$ to track the minimum depth observed at each location. For each target pixel location $\mathbf{p}^{t+1}$, we update the buffer incrementally as: $\mathbf{Z}_{\text{rep}}(\mathbf{p}^{t+1}) = \min\left(z^{t+1}_j\right)$. Finally, the color value at each pixel is resolved by selecting the contribution from the candidate pixel with the smallest depth value. Formally, the final color is given by:

$$C_{\text{final}}(\mathbf{p}^{t+1}) = C^t(\mathbf{p}^t_k), \quad k = \arg\min_j z^{t+1}_j. \tag{12}$$

To ensure correctness in a parallel rendering environment, we leverage CUDA atomic operations. This prevents race conditions during composition and guarantees accurate occlusion relationships.

**Algorithm 1** Tile-Based Warping for 3DGS Reuse

**Input:** Initial rendered image $\mathbf{I}^t$, depth image $\mathbf{Z}$
**Output:** $t+n$-th frame image $\mathbf{I}^{t+n}$

**for** $i$ in $1 \ldots n$ **do**
  Warp $\mathbf{I}^{t+i-1}$ to form partial valid image $\mathcal{P}^u$ and depth image $\mathbf{Z}^u$, pixel-level mask $\mathbf{M}^P$;
  Prepare tile-level mask $\mathbf{M}$;
  $\mathbf{I}^{t+i} \leftarrow$ empty;
  $\mathbf{M}^P \leftarrow$ zeros;
  $\mathbf{M} \leftarrow$ zeros;
  $\mathbf{M}^P_{\text{fill}} = (\mathbf{M}^P \oplus \mathcal{S}) \ominus \mathcal{S} - \mathbf{M}^P$; {Perform morphological closing}
  $\mathbf{M}^P \leftarrow \mathbf{M}^P + \mathbf{M}^P_{\text{fill}}$; {Update the pixel-level mask}
  **for** pixel $\mathbf{p}$ in $\mathcal{P}^u$ **do**
    **if** $\mathbf{M}^P_{\text{fill}}(\mathbf{p}) == 1$ **then**
      $\mathbf{Z}^u(\mathbf{p}) = \max_{\mathbf{q}\in\mathcal{N}(\mathbf{p})} \mathbf{Z}^u(\mathbf{q})$, where $\mathbf{M}^P(\mathbf{q}) = 1$; {Fill depth}
    **end if**
  **end for**
  **for** pixel $\mathbf{p}$ in $\mathcal{P}^u$ **do**
    **if** $\mathbf{M}^P_{\text{fill}}(\mathbf{p}) == 1$ **then**
      Perform depth-guided bilateral filtering at $\mathbf{p}$;
    **end if**
  **end for**
  **for** tile $\tau$ in $\mathcal{P}^u$ **do**
    **if** all pixels in $\tau$ have $\mathbf{M}^P(\mathbf{p}) == 1$ **then**
      $\mathbf{M}(\tau) \leftarrow 1$; {Set tile-level mask}
      Add tile $\tau$ to reused tiles $\mathcal{T}^u$;
    **end if**
  **end for**
  Pass tile mask $\mathbf{M}$ to 3DGS render pipeline;
  Render remaining tiles $\mathcal{T}^r$;
  Combine $\mathcal{T}^u$ and $\mathcal{T}^r$ to form $\mathbf{I}^{t+i}$;
**end for**

### 4.4 Incremental Rendering

Once the frame $t+1$ is generated from frame $t$, it can further serve as the source for generating frame $t+2$. This process can be recursively applied to produce continuous frames $t+3, t+4, \ldots$, enabling efficient incremental rendering without full rendering at each step. This increases the fraction of reused frames and improves rendering speed performance. However, directly applying this multi-frame reuse strategy leads to accumulated errors due to inaccuracies in depth estimation and numerical imprecision during repeated projection and warping. To address this issue, we remove the most error-prone pixels—specifically, those inpainted during the generation of frame $t+1$—before proceeding to generate frame $t+2$. By excluding these unreliable regions, we improve the quality of the reusable content and maintain higher temporal consistency across frames. However, this filtering inevitably increases the number of tiles that must be re-rendered, resulting in a trade-off between rendering quality and performance.

To better understand this trade-off, we analyze the overall rendering time in the context of incremental frame generation, as shown in Figure 4. Let $a \in [0,1]$ denote the proportion of pixels that need to be re-rendered in each generated frame, and let $n$ denote the number of

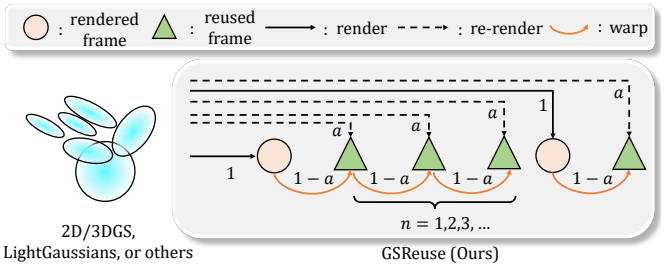



Fig. 4: Theoretical analysis of performance and trade offs between $a$ and $n$. $a$ denotes the proportion of re-rendered pixels within a reused frame. Increasing $a$ improves quality, and $a = 1$ corresponds to the lossless case, but it reduces the speedup ratio. $n$ denotes the number of reused frames. A larger $n$ improves the speedup ratio but may also lead to error accumulation, where the per-frame accumulated error depends on the value of $1 - a$.

consecutively generated frames following a fully rendered one. We divide the total rendering time per frame into two parts: the time $t_r$ required for reusing (warping) a frame, and the time $t_u$ for re-rendering the invalid pixels. Then, the average rendering time per frame is:

$$t = \frac{(1 + a \cdot n) \cdot t_\mathrm{r} + (1 - a) \cdot n \cdot t_\mathrm{u}}{1 + n}, \tag{13}$$

Given that re-rendeing operations dominate the computation cost $t_\mathrm{r} \gg t_\mathrm{u}$, we can approximate the total time as: $t \approx \frac{(1+a\cdot n)\cdot t_\mathrm{r}}{1+n}$. The corresponding speedup $S$, when compared to fully rendering every frame using $t_\mathrm{r}$, is: $S = \frac{t_\mathrm{r}}{t} \approx \frac{1+n}{1+a\cdot n}$. This formulation reveals two key properties. First, the speedup decreases as the re-rendering ratio $a$ increases, as shown by the derivative $\frac{\partial S}{\partial a} = -\frac{n(n+1)}{(an+1)^2} \leq 0$. In the ideal case where $a \to 0$, i.e., no re-rendering is required, the maximum speedup is $\lim_{a\to 0} S = \lim_{a\to 0} \frac{1+n}{1+a\cdot n} = 1 + n$. Second, the speedup increases with the number of generated frames, as shown by $\frac{\partial S}{\partial n} = \frac{1-a}{(an+1)^2} \geq 0$. As $n \to \infty$, the theoretical upper bound of speedup approaches $\lim_{n\to\infty} S = \frac{1}{a}$. These results highlight a fundamental trade-off between the re-rendering ratio $a$ and the number of generated frames $n$: reducing $a$ improves quality but reduces speedup, while increasing $n$ boosts performance but risks accumulating artifacts. Our method addresses this trade-off by eliminating the most error-prone pixels before reuse, achieving a practical balance between efficiency and rendering fidelity over long sequences, as shown in Figure 11.

## 5 Results

### 5.1 Experiment Settings

To evaluate the effectiveness of our method, we conduct experiments on all 9 scenes from the Mip-NeRF360 dataset [3]. Additionally, to show the benefits of our method for improving the frame rate of large-scale scenes, we also test it on the GauU-Scene [64] and MatrixCity [36] datasets. The image quality metrics we choose are PSNR, SSIM, and LPIPS. All experiments are conducted on a single NVIDIA RTX 4090 GPU, using the same setting of hyper-parameters. Unless otherwise noted, we only reuse one frame (i.e., $n = 1$) for each scene.

### 5.2 Evaluation

#### 5.2.1 Comparison with Video Frame Generation Methods

Our method draws inspiration from video frame generation techniques that create new frames by reusing contents from preceding (and subsequent) frames. In this section, we make comparisons with state-of-the-art video frame interpolation (IFRNet [32] and RIFE [25]) and extrapolation (DMVFN [23]) approaches. To ensure fair comparison, we first render reference video sequences using the original 3DGS rendering pipeline. Then, we render the same video sequences using our proposed method with temporal reuse enabled. For video frame generation methods, we exclude the reused frames and regenerate them using the interpolation or extrapolation methods. Quality metrics are then computed between the generated frames and the reference, as summarized in Table 1. Except for the garden scene, our method consistently achieves superior rendering quality compared to existing video frame generation methods. Even on the garden scene, our results are comparable to RIFE, which achieves the best performance among the baselines.

In addition to high visual quality, our method also significantly outperforms all frame interpolation/extrapolation baselines in terms of rendering speed, as shown in Table 2. This substantial speed advantage stems from the fact that, unlike video frame generation methods that rely on neural networks, our approach is non-learning-based and operates directly on the rendered outputs—allowing us to achieve extremely low latency.

Visual comparisons with video frame generation methods are presented in Figure 8. As can be observed, IFR-Net tends to produce overly blurred results and fails to capture thin geometric structures, often leading to their disappearance. For example, in the bicycle scene, a part of the rear fork of the bicycle is missing. Both IFRNet and RIFE suffer from inaccurate optical flow estimation, resulting in ghosting artifacts—multiple dislocated versions of the same structure appear in a single frame (e.g., the front fork of the bicycle in the bonsai scene). Additionally, RIFE and DMVFN tend to distort structural geometry, causing visible stretching or compression. This is particularly evident in the counter scene, where the knobs appear unnaturally elongated. In contrast, our method reliably estimates pixel motion and preserves structural integrity. The reused frames generated by our approach remain consistent with the original 3DGS-rendered outputs, free from ghosting or distortion.

#### 5.2.2 Compatibility with Other Acceleration Methods

Figure 9 compares per-frame rendering times with and without our method when combined with existing 3DGS acceleration techniques. As shown, despite variations in method and scene complexity, combining our pipeline with other techniques consistently yields significant performance gains. For the original 3DGS pipeline, our method achieves a 2.24× speedup, reducing average rendering time from 4.5 ms to 2.0 ms. For other acceleration methods, the per-frame rendering time is already reduced, so the relative speedup is slightly smaller. Nevertheless, combinations with our method still achieves around 1.8× acceleration. Moreover, since our method operates entirely in screen space, it helps mitigate frame time fluctuations caused by varying Gaussian densities across different views.

#### 5.2.3 Evaluation on Large-Scale Scenes

We also evaluate our method on large-scale 3DGS scenes by integrating it with CityGaussian-v2 [41], and conduct experiments on two large-scene datasets: GauU and MatrixCity. Results on selected scenes are shown in Figure 10. Our method achieves an average 1.6× speedup, significantly improving rendering efficiency on complex scenes. In particular, on the MC-Aerial scene, our approach maintains an average rendering speed of 100 FPS. Furthermore, the error maps reveal that our method introduces negligible quality degradation, with an average RMSE of only 0.02. This demonstrates that our approach can effectively accelerate large-scale scene rendering while preserving high visual fidelity.

#### 5.2.4 Stage-wise Runtime Breakdown

we provide a breakdown of the stage-wise runtime comparison between our method and the original 3D Gaussian Splatting (3DGS), as shown in Figure 5. The results are generally consistent with our expectations: the most substantial reductions in runtime occur in the sorting and compositing stages, which are known to be the main bottlenecks in the original pipeline.

### 5.3 Ablation Study

In this section, we analyze the effectiveness and necessity of key component in our method.

Table 1: Quantitative comparison between our method and video frame generation methods on all 9 scenes of the Mip-NeRF360 dataset. Bold numbers indicate the best performance, while underlined values denote the second-best performance in our experiments.

| Method | bicycle | | | bonsai | | | counter | | | flowers | | | garden | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| *IFRNet* | 25.90 | 0.845 | 0.169 | 26.02 | 0.874 | 0.161 | 29.48 | 0.940 | 0.071 | 22.79 | 0.791 | 0.183 | 29.29 | 0.891 | 0.086 |
| *RIFE* | 30.79 | 0.939 | 0.055 | 31.93 | 0.961 | 0.036 | 34.40 | 0.978 | **0.022** | 27.22 | 0.915 | 0.073 | **33.44** | **0.962** | **0.031** |
| *DMVFN* | 23.47 | 0.752 | 0.171 | 25.72 | 0.866 | 0.113 | 27.55 | 0.901 | 0.076 | 21.25 | 0.699 | 0.177 | 25.66 | 0.795 | 0.111 |
| *Ours* | **31.83** | **0.956** | **0.043** | **36.91** | **0.985** | **0.021** | **36.25** | **0.980** | 0.026 | **29.81** | **0.946** | **0.047** | 32.52 | 0.948 | 0.032 |

| Method | kitchen | | | room | | | stump | | | treehill | | | average | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| *IFRNet* | 26.42 | 0.840 | 0.092 | 32.96 | 0.914 | 0.124 | 27.93 | 0.848 | 0.189 | 26.45 | 0.910 | 0.097 | 27.47 | 0.873 | 0.130 |
| *RIFE* | 32.34 | 0.953 | 0.026 | 36.65 | 0.960 | 0.046 | 35.07 | 0.964 | 0.041 | 28.52 | 0.944 | 0.055 | 32.26 | 0.953 | 0.043 |
| *DMVFN* | 27.21 | 0.848 | 0.080 | 30.39 | 0.909 | 0.096 | 27.78 | 0.827 | 0.143 | 23.30 | 0.840 | 0.116 | 25.81 | 0.826 | 0.120 |
| *Ours* | **36.26** | **0.977** | **0.021** | **39.30** | **0.986** | **0.021** | **37.09** | **0.973** | **0.033** | **31.05** | **0.954** | **0.041** | **34.56** | **0.967** | **0.031** |

Table 2: Average frame generation time (ms) in 1600x1000 resolution over all 9 scenes of the Mip-NeRF360 dataset.

| IFRNet | RIFE | DMVFN | Ours |
|---|---|---|---|
| 27.22 | 9.89 | 20.78 | **2.64** |

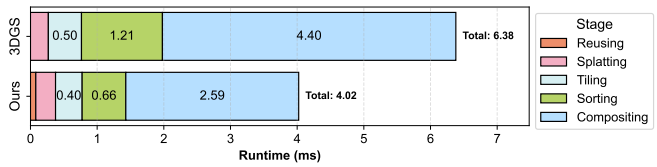
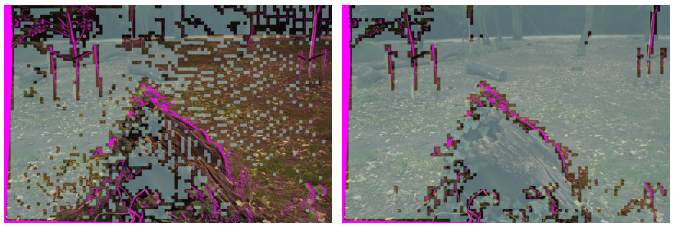



Fig. 5: Stage-wise runtime breakdown for rendering on the Kitchen scene from the Mip-NeRF 360 dataset [3]. Each bar represents the average per-frame runtime, with different colors indicating stages in the rendering pipeline.



(a) w/o filtering (b) w/ filtering

Fig. 6: Comparison of reusable tile coverage with and without pixel filtering. Reusable tiles are marked in gray, while invalid pixels after warping are shown in purple.

We evaluate the impact of the pixel filtering module. As shown in Figure 6, we compare the reusable tile coverage with and without pixel filtering. It can be clearly observed that the right-hand side (with pixel filtering) achieves significantly larger tile coverage than the left-hand side (without filtering). From the invalid pixel distribution, many tiles contain only a small number of invalid pixels. The filtering step effectively fills in these sparse invalid pixels without sacrificing visual quality, thereby increasing the number of tiles that can be reused.

## 6 Limitation

While our method demonstrates consistent improvements across various scenes, it also comes with several limitations. Our method relies on depth maps generated by 3DGS to perform warping, and is therefore sensitive to the accuracy of depth estimation. Inaccurate depth predictions can lead to incorrect occlusion handling, where parts of a foreground object may be mistakenly overwritten by warped content from other regions due to erroneous depth values. Since 2DGS provides more accurate depth maps than 3DGS, we perform a comparison to evaluate its impact on reuse quality. As shown in Figure 7, using 2DGS depth leads to noticeable improvements, indicating that depth accuracy has a direct influence on the effectiveness of our method.

Our method, like all reuse-based approaches, assumes temporal redundancy between consecutive frames. In cases of fast camera motion or strong parallax, the overlap between neighboring frames significantly decreases. This reduces the effectiveness of reuse, leading to performance closer to the original 3DGS.

Figure 5 shows that, despite significant improvements in the sorting and compositing stages, the overall runtime reduction does not scale proportionally with the reuse ratio observed in Figure 1. This is primarily due to the spatial distribution of complexity across scenes. In particular, tiles containing dense Gaussian populations tend to correspond to important regions that are less likely to be reused due to view-dependent visibility changes, which partially offsets the benefits of reuse.

## 7 Conclusion and Future Work

We propose GSReuse, a practical and general screen-space solution for accelerating Gaussian Splatting-based neural rendering. By reusing previously rendered frames through tile-based warping, our method enables the generation of high-quality frames with significantly reduced computation, effectively reusing previously rendered contents across consecutive frames. Through extensive experiments, we demonstrate that GSReuse substantially improves rendering efficiency while maintaining high visual fidelity, and can be seamlessly integrated with a variety of existing 3DGS methods.

In future work, we would like to addressing limitations of our method by developing more robust depth estimation techniques, motion prediction, or uncertainty-aware reuse strategies.

## References

[1] AMD. Amd fidelityfx™ super resolution, 2021. Accessed: 2023-05-23. 2

[2] AMD. Amd fidelityfx™ super resolution 3, 2022. https://gpuopen.com/fidelityfx-super-resolution-3/ [Accessed: 2024-01-24]. 2

[3] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5470–5479, June 2022. 6, 7, 11

[4] K. M. Briedis, A. Djelouah, M. Meyer, I. McGonigal, M. Gross, and C. Schroers. Neural frame interpolation for rendered content. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021. 2

[5] K. M. Briedis, A. Djelouah, R. Ortiz, M. Meyer, M. Gross, and C. Schroers. Kernel-based frame interpolation for spatio-temporally adaptive rendering. In *ACM SIGGRAPH 2023 Conference Proceedings*, pp. 1–11, 2023. 2

[6] J. Cui, J. Cao, F. Zhao, Z. He, Y. Chen, Y. Zhong, L. Xu, Y. Shi, Y. Zhang, and J. Yu. Letsgo: Large-scale garage modeling and rendering via lidar-assisted gaussian primitives. *ACM Transactions on Graphics (TOG)*, 43(6):1–18, 2024. 2

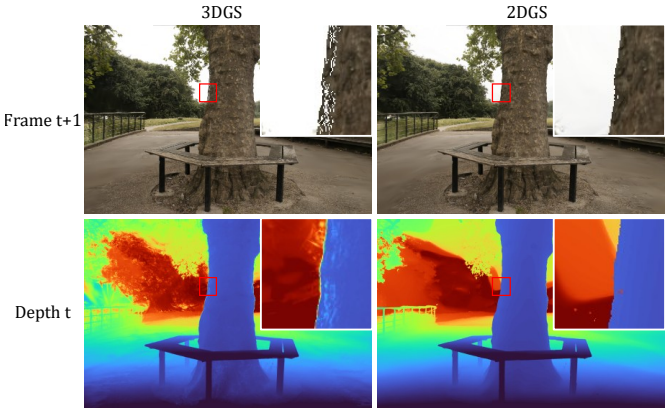



Fig. 7: Visual comparison of rendering and depth quality using 3DGS and 2DGS depth maps within our reuse pipeline. The top row shows rendered images of the frame t+1, and the bottom row shows the corresponding depth maps from frame t used for warping. Replacing 3DGS depth with more accurate 2DGS depth leads to visibly improved results.


[7] Z. Ding, C.-T. Lee, M. Zhu, T. Guan, Y.-C. Sun, C.-H. Hsu, and Y. Liu. Eyenavgs: A 6-dof navigation dataset and record-n-replay software for real-world 3dgs scenes in vr. MM '25, 7 pages, p. 13126–13132. Association for Computing Machinery, New York, NY, USA, 2025. doi: 10.1145/3746027.3758265 1

[8] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. In *Advances in Neural Information Processing Systems*, vol. 37, pp. 140138–140158. Curran Associates, Inc., 2024. 1, 2, 11

[9] G. Fang and B. Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, pp. 165–181. Springer, 2024. 1, 2, 11

[10] G. Feng, S. Chen, R. Fu, Z. Liao, Y. Wang, T. Liu, Z. Pei, H. Li, X. Zhang, and B. Dai. Flashgs: Efficient 3d gaussian splatting for large-scale and high-resolution rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 2, 3

[11] Y. Feng, P. Hansen, P. N. Whatmough, G. Lu, and Y. Zhu. Fast and accurate: Video enhancement using sparse depth. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 4492–4500, January 2023. 2

[12] Y. Feng, W. Lin, Y. Cheng, Z. Liu, J. Leng, M. Guo, and Y. Zhu. Lumina: Real-time neural rendering by exploiting computational redundancy. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*, 2025. doi: 10.1145/3695053.3731003 2

[13] Y. Feng, Z. Liu, J. Leng, M. Guo, and Y. Zhu. Cicero: Addressing algorithmic and architectural bottlenecks in neural rendering by radiance warping and memory optimizations. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024. doi: 10.1109/ISCA59077.2024.00096 2

[14] L. Franke, L. Fink, and M. Stamminger. Vr-splatting: Foveated radiance field rendering via 3d gaussian splatting and neural points. *Proc. ACM Comput. Graph. Interact. Tech.*, 8(1), article no. 18, 21 pages, May 2025. doi: 10.1145/3728302 1

[15] L. Franke, L. Fink, and M. Stamminger. Vr-splatting: Foveated radiance field rendering via 3d gaussian splatting and neural points. *Proc. ACM Comput. Graph. Interact. Tech.*, 8(1), article no. 18, May 2025. doi: 10.1145/3728302 2, 3

[16] S. Girish, K. Gupta, and A. Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*, pp. 54–71. Springer, 2024. 1, 2

[17] S. Gu, H. Song, B. Liu, Q. Yu, S. Zhang, H. Jiang, J. Huang, and F. Tian. Vrsketch2gaussian: 3d vr sketch guided 3d object generation with gaussian splatting, 2025. 2

[18] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. Foveated 3d graphics. *ACM transactions on Graphics (TOG)*, 31(6):1–10, 2012. 2

[19] J. Guo, X. Fu, L. Lin, H. Ma, Y. Guo, S. Liu, and L.-Q. Yan. Extranet: Real-time extrapolated rendering for low-latency temporal supersampling. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021. 2

[20] A. Hanson, A. Tu, G. Lin, V. Singla, M. Zwicker, and T. Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 2

[21] A. Hanson, A. Tu, V. Singla, M. Jayawardhana, M. Zwicker, and T. Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 1, 2

[22] Q. Hou, R. Rauwendaal, Z. Li, H. Le, F. Farhadzadeh, F. Porikli, A. Bourd, and A. Said. Sort-free gaussian splatting via weighted sum rendering. In *The Thirteenth International Conference on Learning Representations*, 2025. 2

[23] X. Hu, Z. Huang, A. Huang, J. Xu, and S. Zhou. A dynamic multi-scale voxel flow network for video prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6121–6131, 2023. 2, 6, 10

[24] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. doi: 10.1145/3641519.3657428 11

[25] Z. Huang, T. Zhang, W. Heng, B. Shi, and S. Zhou. Real-time intermediate flow estimation for video frame interpolation. In *European Conference on Computer Vision*, pp. 624–642. Springer, 2022. 2, 6, 10

[26] Intel. Intel® arc™- xe super sampling, 2022. Accessed: 2023-05-23. 2

[27] Y. Jiang, C. Yu, T. Xie, X. Li, Y. Feng, H. Wang, M. Li, H. Lau, F. Gao, Y. Yang, and C. Jiang. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, article no. 78, 1 pages. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3641519.3657448 2

[28] X. Jin, L. Wu, J. Chen, Y. Chen, J. Koo, and C.-h. Hahm. A unified pyramid recurrent network for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1578–1587, 2023. 2

[29] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. 1, 2, 11

[30] B. Kerbl, A. Meuleman, G. Kopanas, M. Wimmer, A. Lanvin, and G. Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024. 2

[31] J. Kim and S. Lee. Potentially visible hidden-volume rendering for multi-view warping. *ACM Transactions on Graphics (TOG)*, 42(4):1–11, 2023. 2

[32] L. Kong, B. Jiang, D. Luo, W. Chu, X. Huang, Y. Tai, C. Wang, and J. Yang. Ifrnet: Intermediate feature refine network for efficient frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1969–1978, 2022. 2, 6, 10

[33] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim. Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 497–511, 2024. 2

[34] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21719–21728, 2024. 1, 2

[35] S. Lee, Y. Kim, and E. Eisemann. Iterative depth warping. *ACM Transactions on Graphics (TOG)*, 37(5):1–13, 2018. 2

[36] Y. Li, L. Jiang, L. Xu, Y. Xiangli, Z. Wang, D. Lin, and B. Dai. Matrixcity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3205–3215, 2023. 6

[37] F. Lin, Y. Hu, Z. Liu, Y. Zhuang, Z. Lin, and J. Zhang. Mon3tr: Monocular 3d telepresence with pre-built gaussian avatars as amortization, 2026. 2

[38] W. Lin, Y. Feng, and Y. Zhu. Metasapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 669–682, 2025. 2

[39] E. Liu. Dlss 2.0-image reconstruction for real-time rendering with deep learning. In *Nvidia GPU Tech. Conf. (GTC)*, 2020. 2

[40] Y. Liu, C. Luo, L. Fan, N. Wang, J. Peng, and Z. Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In

*European Conference on Computer Vision*, pp. 265–282. Springer, 2024. 1, 2

[41] Y. Liu, C. Luo, Z. Mao, J. Peng, and Z. Zhang. Citygaussianv2: Efficient and geometrically accurate reconstruction for large-scale scenes, 2025. 1, 6, 11

[42] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of detail for 3D graphics*. Elsevier, 2002. 2

[43] S. S. Mallick, R. Goel, B. Kerbl, M. Steinberger, F. V. Carrasco, and F. De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, pp. 1–11, 2024. 1, 2, 11

[44] H. Mao, Z. Xu, S. Wei, Y. Quan, N. Deng, and X. Yang. Live-gs: Llm powers interactive vr by enhancing gaussian splatting. In *2025 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 1234–1235, 2025. doi: 10.1109/VRW66409.2025.00263 2

[45] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 7–ff, 1997. 2

[46] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[47] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *European Conference on Computer Vision*, pp. 18–34. Springer, 2024. 1, 2

[48] J. H. Mueller, T. Neff, P. Voglreiter, M. Steinberger, and D. Schmalstieg. Temporally adaptive shading reuse for real-time rendering and virtual reality. *ACM Trans. Graph.*, 40(2), article no. 11, 14 pages, Apr. 2021. 2

[49] K. Navaneet, K. Pourahmadi Meibodi, S. Abbasi Koohpayegani, and H. Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. In *European Conference on Computer Vision*, pp. 330–349. Springer, 2024. 1, 2

[50] S. Niedermayr, J. Stumpfegger, and R. Westermann. 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. 1, 2

[51] NVIDIA. Nvidia dlss 3: Ai-powered performance multiplier boosts frame rates by up to 4x, 2022. https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/ [Accessed: 2024-01-24]. 2

[52] P. Papantonakis, G. Kopanas, B. Kerbl, A. Lanvin, and G. Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1), May 2024. 1, 2

[53] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions On Graphics (TOG)*, 35(6):1–12, 2016. 2

[54] B. Reinert, J. Kopf, T. Ritschel, E. Cuervo, D. Chu, and H.-P. Seidel. Proxy-guided image-based rendering for mobile devices. In *Computer Graphics Forum*, vol. 35, pp. 353–362. Wiley Online Library, 2016. 2

[55] D. Scherzer, L. Yang, O. Mattausch, D. Nehab, P. V. Sander, M. Wimmer, and E. Eisemann. Temporal coherence methods in real-time rendering. In *Computer Graphics Forum*, vol. 31, pp. 2378–2408. Wiley Online Library, 2012. 2

[56] D. Taniguchi. Gaussmr: Interactive gaussian splatting sandbox with gpu particles and signed distance fields. In *ACM SIGGRAPH 2024 Immersive Pavilion*, SIGGRAPH '24, article no. 6, 2 pages. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3641521.3664405 2

[57] X. Tu, B. Kerbl, and F. de la Torre. Fast and robust 3d gaussian splatting for virtual reality. In *SIGGRAPH Asia 2024 Posters*, pp. 1–3. 2024. 2

[58] X. Tu, L. Radl, M. Steiner, M. Steinberger, B. Kerbl, and F. de la Torre. Vrsplat: Fast and robust gaussian splatting for virtual reality. *Proc. ACM Comput. Graph. Interact. Tech.*, 8(1), article no. 1, 22 pages, May 2025. doi: 10.1145/3728311 1

[59] B. Walter, G. Drettakis, and S. Parker. Interactive rendering using the render cache. In *Rendering Techniques' 99: Proceedings of the Eurographics Workshop in Granada, Spain, June 21–23, 1999 10*, pp. 19–30. Springer, 1999. 2

[60] X. Wang, R. Yi, and L. Ma. Adr-gaussian: Accelerating gaussian splatting with adaptive radius. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, article no. 73, 10 pages. Association for Computing Machinery, New York, NY, USA, 2024. 2

[61] S. Wu, D. Vembar, A. Sochenov, S. Panneer, S. Kim, A. Kaplanyan, and L.-Q. Yan. Gffe: G-buffer free frame extrapolation for low-latency real-time rendering. *ACM Transactions on Graphics (TOG)*, 43(6):1–15, 2024. 2

[62] Y. Wu, Q. Wen, and Q. Chen. Optimizing video prediction via video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17814–17823, 2022. 2

[63] Z. Wu, C. Zuo, Y. Huo, Y. Yuan, Y. Peng, G. Pu, R. Wang, and H. Bao. Adaptive recurrent frame prediction with learnable motion vectors. In *SIGGRAPH Asia 2023 Conference Papers*, pp. 1–11, 2023. 2

[64] B. Xiong, N. Zheng, J. Liu, and Z. Li. Gauu-scene v2: Assessing the reliability of image-based metrics with expansive lidar image dataset using 3dgs and nerf, 2024. 6

[65] L. Yang, S. Liu, and M. Salvi. A survey of temporal antialiasing techniques. *Computer Graphics Forum*, 39(2):607–621, July 2020. doi: 10.1111/cgf.14018 2

[66] L. Yang, Y.-C. Tse, P. V. Sander, J. Lawrence, D. Nehab, H. Hoppe, and C. L. Wilkins. Image-based bidirectional scene reprojection. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pp. 1–10, 2011. 2

[67] S. Yang, Q. Zhu, J. Zhuge, Q. Qiu, C. Li, Y. Yan, H. Xu, L.-Q. Yan, and X. Jin. Mob-fgsr: Frame generation and super resolution for mobile real-time rendering. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024. 2

[68] Z. Ye, Y. Fu, J. Zhang, L. Li, Y. Zhang, S. Li, C. Wan, C. Wan, C. Li, S. Prathipati, et al. Gaussian blending unit: An edge gpu plug-in for real-time gaussian-based rendering in ar/vr. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 353–365. IEEE, 2025. 2

[69] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19447–19456, June 2024. 3

[70] Z. Zeng, S. Liu, J. Yang, L. Wang, and L.-Q. Yan. Temporally reliable motion vectors for real-time ray tracing. In *Computer Graphics Forum*, vol. 40, pp. 79–90. Wiley Online Library, 2021. 2

[71] G. Zhang, C. Liu, Y. Cui, X. Zhao, K. Ma, and L. Wang. Vfimamba: Video frame interpolation with state space models. *Advances in Neural Information Processing Systems*, 37:107225–107248, 2024. 2

[72] X. Zhang, K. Liu, Y. Liu, F. Li, J. Ma, and Y. Li. East: Environment-aware stylized transition along the reality-virtuality continuum, 2025. 2

[73] B. Zhao, Y. Li, Z. Sun, L. Zeng, Y. Shen, R. Ma, Y. Zhang, H. Bao, and Z. Cui. Gaussianprediction: Dynamic 3d gaussian prediction for motion extrapolation and free view synthesis. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–12, 2024. 2

[74] H. Zhu, Z. Liu, X. Li, A. Wu, J. Zhao, F. Liu, Y. Gan, J. Leng, and Y. Feng. Nebula: Enable city-scale 3d gaussian splatting in virtual reality via collaborative rendering and accelerated stereo rasterization, 2025. 1

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2026.3679889

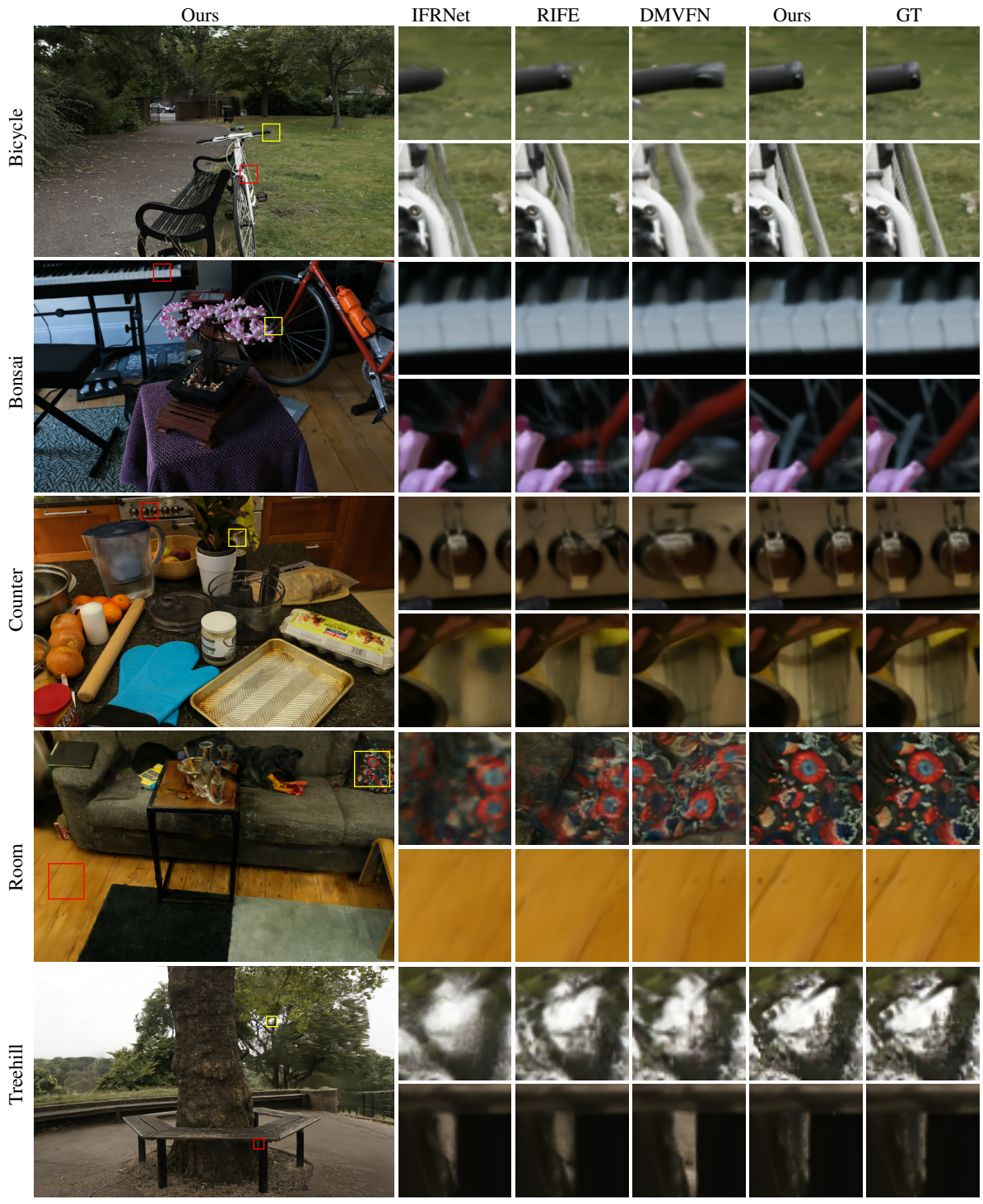



Fig. 8: Visual comparison to state-of-the-art video frame interpolation (IFR-Net [32], RIFE [25]) and extrapolation (DMVFN [23]) methods. Close-ups highlight the artifacts (over blurriness, broken structures, and ghosting) generated by these deep learning-based methods, while our method is free from these artifacts.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2026.3679889

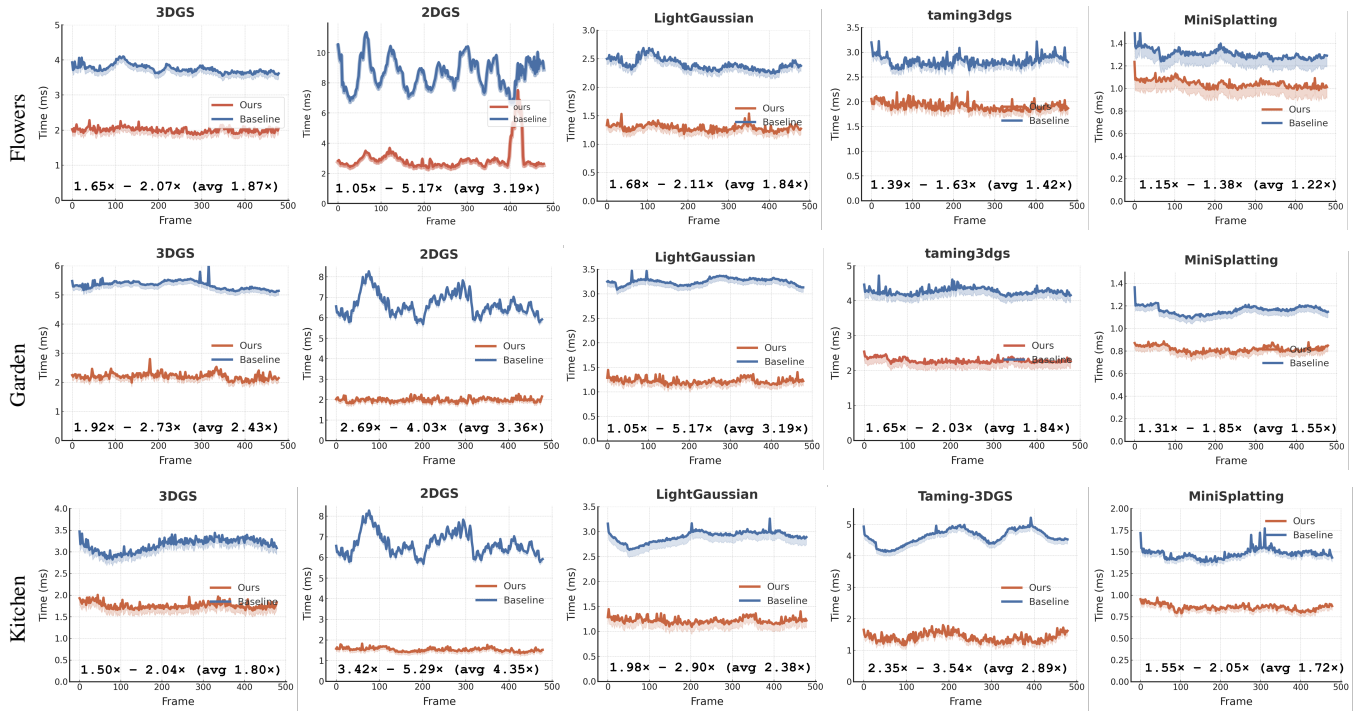



Fig. 9: Per-frame rendering time comparison on three typical scenes from the Mip-NeRF360 dataset [3]. We show the acceleration effect on five baseline methods: 3DGS [29], 2DGS [24], LightGaussian [8], Taming-3DGS [43], and Mini-Splatting [9].

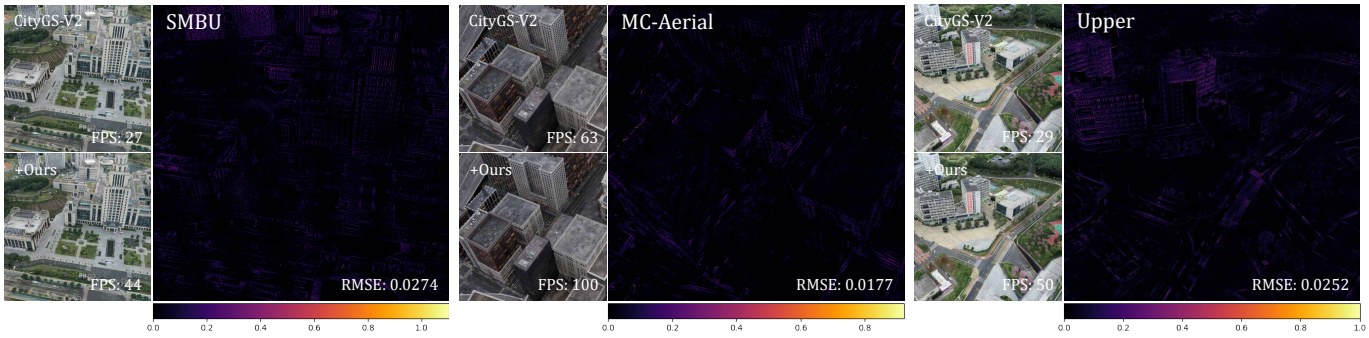



Fig. 10: Our method significantly improves the rendering performance (FPS) of the large-scale CityGaussianV2 scenes [41], while maintaining high visual quality, as evidenced by error maps (RMSE).


Fig. 11: Evaluation of different reuse lengths $n$ after each fully rendered frame. Only the last frame among reused frames is shown for each setting.