# GAUSSIAN STIPPLING: EFFICIENT SORTING-FREE 3D GAUSSIAN RENDERING THROUGH HYBRID SAMPLING AND SPATIOTEMPORAL RECONSTRUCTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Conventional 3D Gaussian Splatting (3DGS) requires depth sorting and ordered alpha blending to correctly render overlapping Gaussian primitives. Stochastic transparency enables sorting-free rendering by replacing fractional alpha contributions with discrete stochastic visibility samples, but produces substantial spatial and temporal noise at low sample counts. We refer to this conversion from continuous Gaussian splats to discrete visibility samples as *Gaussian Stippling*. Based on this, we present an real-time order-independent rendering and reconstruction framework that operates directly on unmodified 3DGS assets. Our method alleviates the considerable instability of primitive-based sampling across views by introducing a primitive-wise router that adaptively integrates primitive-based and fragment-based stippling, significantly improving rendering throughput. To recover high-quality images from sparse stochastic samples, we further introduce a lightweight Gaussian-aware spatiotemporal reconstruction network. By exploiting the Gaussian attributes retained by each stipple, the network aggregates structured stochastic clues across both space and time, effectively suppressing stippling noise. Experiments show that our hybrid Gaussian stippling method, coupled with a spatiotemporal reconstruction network trained on diverse scenes, generalizes to unseen scenes and enables real-time, temporally stable, and visually plausible rendering on mobile devices without retraining or preprocessing. With scene-specific training, increased sampling and network capacity further improve reconstruction fidelity, enabling the recovery of complex lighting patterns and fine appearance details for quality-prioritized applications.

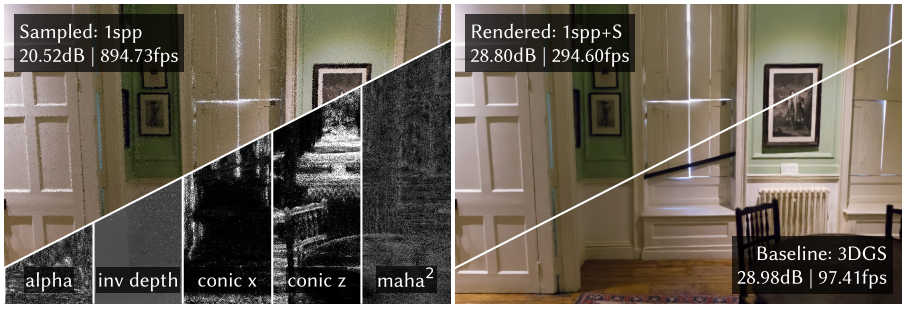


Figure 1: By rendering sorting-free stochastic samples, we achieve high rendering speed but suffer from severe spatial noise (left). We then extract informative Gaussian attributes and feed them into a lightweight spatiotemporal network for reconstruction (right).

## 1 INTRODUCTION

3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) enables real-time novel-view synthesis through efficient rasterization of explicit Gaussian primitives. However, it relies on depth sorting and ordered compositing of overlapping primitives, which limits its practicality in scenes with large primitive counts (Kerbl et al., 2024; Yang et al., 2025; Rijsdijk et al., 2026) and on resource-constrained devices (Du et al., 2026; Gao et al., 2026). Recent methods alleviate this overhead by reformulating the rendering paradigm (Hou et al., 2025; Müller et al., 2025; Hahlbohm et al., 2025; Du et al., 2026;



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

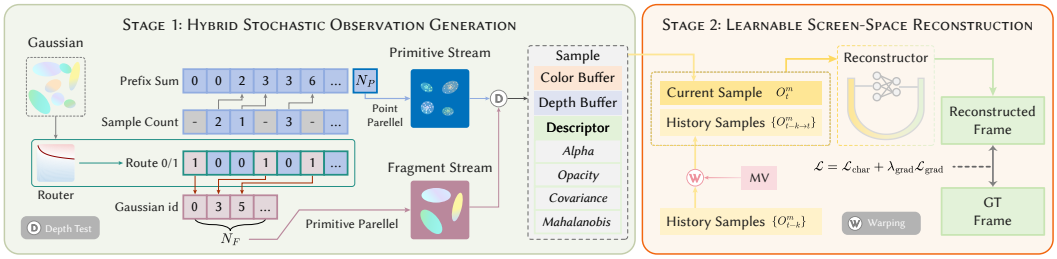


Figure 2: Gaussian stippling and reconstruction pipeline. A cost-aware router assigns Gaussians to primitive-based or fragment-based sampling, and a shared depth test produces Gaussian-local observations. The network reconstructs image from current and reprojected historical observations.

Ye et al., 2026) or streamlining the primitive representation (Fang & Wang, 2024; Niedermayr et al., 2024). Although effective within their respective settings, these approaches are not always directly applicable to existing, unmodified 3DGS assets.

In contrast, stochastic transparency (Enderton et al., 2010) offers a sorting-free mechanism while preserving the underlying Gaussian representation unchanged. Rather than treating opacity as a fractional contribution that must be composited in order, it interprets opacity as the probability of a stochastic visibility event whose expectation converges to the results of ordered alpha compositing. We refer to this transformation as *Gaussian stippling*. Conventional Gaussian splatting accumulates continuous Gaussian splats, whereas Gaussian stippling generates discrete, opaque Gaussian visibility samples, termed *stipples*, which compete through a standard depth test. Thus, Gaussian stippling is not merely a visual analogy but a different image formation process.

Gaussian Point Splatting (GPS) (Rijsdijk et al., 2026) efficiently realizes stochastic transparency via an opacity-corrected Poisson point process. GPS purportedly scales well to scenes with many Gaussian primitives, yet our experiments show that its rendering speed is highly variable across views and scenes (Figure 3). In these cases, we find that StochasticSplats (SS) (Kheradmand et al., 2025) renders faster and more stably, owing to its effective use of hardware acceleration. Through prototype experiments we identify the boundary between these two methods — the regime in which each is preferable. More importantly, we observe that GPS and SS follow the same formulation, which implies that switching between them per Gaussian preserves the unbiasedness of the Monte Carlo estimation. Motivated by these two findings, we propose a cost-aware hybrid renderer that routes each Gaussian to the more suitable stream (we refer to the *primitive-based stream* as GPS and the *fragment-based stream* as SS). Because the router operates at the level of individual primitives, it attains the fastest speed across scenes with varying numbers and sizes of Gaussian primitives, consistently outperforming either method used alone.

Efficient sampling alone, however, is insufficient to make low-sample stippling viable for display. Replacing deterministic compositing with stochastic estimation introduces spatial Monte Carlo noise, unstable occlusion boundaries, and temporal flickering. Although increasing the sample count can mitigate these artifacts, it also erodes the performance gained from sorting-free rendering. Image and video denoising methods achieve relatively high quality and generalizability but suffer from slow inference and are unsuitable for Monte Carlo noise. (Zamir et al., 2022; Tassano et al., 2020) Deep-learning-based Monte Carlo denoising methods mostly focus on clean GBuffers as cues to enhance denoising quality but fail to handle noisy GBuffers. (Bako et al., 2017; Gharbi et al., 2019) To tackle this challenge, we leverage the observation that every visible stipple remains associated with its originating Gaussian and thus carries structured information beyond RGB, including opacity, covariance, depth, etc. Moreover, preceding frames can provide additional stochastic samples without increasing the current-frame sampling rate. We therefore employ a lightweight spatiotemporal reconstruction network to gather spatial and temporal clues from Gaussian-aware observations of the current and reprojected historical frames, efficiently producing noise-free and temporally coherent results. As shown in Figure 1, this reconstruction turns the severely noisy sorting-free stochastic rendering into a clean and coherent image from the resulting Gaussian-aware observations.

Experimental results indicate that our hybrid Gaussian stippling framework provides a favorable balance between generalization, efficiency, and visual quality. Using a spatiotemporal reconstruction network trained from diverse scenes, the method can be directly applied to novel scenes without

per-scene optimization or preprocessing, while delivering real-time and temporally coherent rendering even on mobile hardware. When scene-specific training is permitted, allocating additional sampling resources and model capacity further enhances fidelity, allowing our method to outperform the baselines in high-fidelity rendering scenarios.

Our main contributions are summarized as follows:

- A Gaussian stippling framework that combines sorting-free sampling with a generalizable spatiotemporal reconstruction network to efficiently generate temporally stable and visually plausible renderings for unmodified 3DGS assets, without additional retraining.
- A cost-aware hybrid stippling renderer that routes Gaussian primitives between primitive-based and fragment-based stipple generation, alleviates the considerable instability of primitive-based sampling.
- A Gaussian-aware spatiotemporal reconstruction method that combines Gaussian-local descriptors with forward-reprojected stochastic observations to reconstruct images at low sampling budgets, supporting both scene-specific training and cross-scene deployment.

## 2 Related Work

### 2.1 Accelerating 3D Gaussian Splatting

**Representation-modified Rendering.** To circumvent the expensive depth sorting and alpha compositing in standard 3DGS (Kerbl et al., 2023), recent methods propose order-independent transparency approximations (Müller et al., 2025), depth peeling (Ye et al., 2026), or newly optimized Gaussian variants (Hahlbohm et al., 2025; Du et al., 2026; Hou et al., 2025). While these approaches successfully reduce rendering latency, their requirement for model retraining or parameter conversion makes them incompatible with off-the-shelf 3DGS assets.

**Stochastic Visibility Sampling.** Alternatively, render-time acceleration avoids modifying the assets by altering the rasterization process. Inspired by stochastic transparency (Enderton et al., 2010), StochasticSplat (Kheradmand et al., 2025) and Gaussian Point Splatting (Rijsdijk et al., 2026) apply Monte Carlo visibility sampling to Gaussians. Although they eliminate sorting, their low-spp outputs suffer from severe variance. While these works rely on simple temporal filters or heavy supersampling to mitigate noise, we propose to explicitly reconstruct the stochastic samples using a geometry-aware neural network, achieving high quality at extremely low spp budgets.

### 2.2 Neural Reconstruction from Stochastic Samples

**Neural Image-Based Rendering.** Synthesizing high-quality images from discrete, unstructured geometric proxies is a core challenge in image-based rendering. Neural Point-Based Graphics (NPBG) (Aliev et al., 2020; Rakhimov et al., 2022) and related deferred neural rendering frameworks (Thies et al., 2019; Rückert et al., 2022) have demonstrated that deep convolutional networks (e.g., U-Nets) can effectively reconstruct dense, photorealistic images from rasterized, sparse point features. While these methods excel at resolving holes and artifacts from incomplete point clouds, they typically assume deterministic point attributes and fixed visibility. In contrast, our pipeline is driven by extreme Monte Carlo variance and probabilistic visibility. We therefore adapt the neural deferred rendering paradigm to handle stochastic, semi-transparent volumetric primitives.

**Reconstruction from Stochastic Samples.** Reconstructing expected radiance from highly variant Monte Carlo samples is extensively studied in path tracing (Zwicker et al., 2015; Schied et al., 2017; Huo & Yoon, 2021; Bako et al., 2017). Deep neural networks are widely used to filter these noisy observations (Vogels et al., 2018; Fan et al., 2021; Balint et al., 2023). Our approach is most conceptually aligned with sample-based reconstruction frameworks, such as SBMC (Gharbi et al., 2019), which preserve individual unmerged samples rather than relying on pixel-averaged statistics. However, traditional reconstructors depend on consistent surface G-buffers (e.g., clear depth or albedo), which are ill-defined in 3DGS where images are formed by probabilistically retained overlapping splats (Iglesias-Guitian et al., 2020). To address this, we propose extracting *Gaussian-local descrip-*

*tors* to explicitly encode the local footprint, opacity, and visibility state of each sampled primitive, guiding the network to reconstruct volumetric transparency.

**Spatiotemporal Alignment.** Provided with the extreme variance of 1-spp rendering, temporal reuse is indispensable (Scherzer et al., 2010; Chaitanya et al., 2017; Bitterli et al., 2020; Lin et al., 2021). Conventional real-time pipelines typically rely on backward reprojection using stable surface motion vectors (Yang et al., 2020). However, this fails under stochastic 3DGS: even with a static camera, stochastically sampled Gaussians yield inconsistent visibility events across frames. Directly blending historical pixel colors introduces severe ghosting and blurring. Instead, we utilize *forward reprojection at the sample level*, projecting historical stochastic sample descriptors into the current frame as candidate observations so that our network learns optimal temporal sample reuse and fusion without relying on fragile heuristic blending rules (Hofmann et al., 2021; Zhu et al., 2023). Instead, we forward-reproject historical pixel observations, formed by averaging stochastic stipple descriptors within each pixel. The network jointly processes these aligned observations and the current-frame input to learn temporal reuse and occlusion handling (Hofmann et al., 2021; Zhu et al., 2023).

# 3 METHOD

Given a fixed, trained 3D Gaussian representation $\mathcal{G}$ and a target camera pose $C_t$ at frame $t$, our goal is to reconstruct a high-quality image from stochastic Gaussian observations without any modification or optimization of the original scene representation. As illustrated in Figure 1, our method consists of two stages. First, a hybrid renderer routes each Gaussian primitive to either a primitive-based sampling stream (Rijsdijk et al., 2026) or a fragment-based sampling stream (Kheradmand et al., 2025) to generate a sparse stipple image, where each accepted sample is augmented with a local descriptor, dubbed an *observation*. Second, we generate $M$-spp (samples per pixel) averaged stochastic observations $\mathcal{O}_t^M$ for the current frame and the $K = 3$ preceding frames, and concatenate them to form $X_t$. A lightweight convolutional network $D_\theta$ then takes $X_t$ as input to reconstruct the final image $\hat{I}_t$, formulated as $\hat{I}_t = D_\theta(X_t)$.

## 3.1 HYBRID STOCHASTIC OBSERVATION GENERATION

**Stochastic visibility.** Considering a pixel sample location $\mathbf{x} \in \mathbb{R}^2$ covered by $N$ projected Gaussian primitives $\mathcal{G}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, o_i, \mathbf{c}_i, z_i\}$, standard 3D Gaussian splatting (Kerbl et al., 2023) computes front-to-back alpha blending as $I^{\mathrm{GS}}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) T_i(\mathbf{x}) \mathbf{c}_i$ ,where $T_i(\mathbf{x}) = \prod_{z_j < z_i} (1 - \alpha_j(\mathbf{x}))$ is the transmittance, $\alpha_i(\mathbf{x}) = o_i G(\mathbf{x}; \mu_i, \boldsymbol{\Sigma}_i)$ is the alpha value, $o_i$ is the base opacity, $\mathbf{c}_i$ is the view-dependent color, and $z_i$ is the depth.

Stochastic transparency (Enderton et al., 2010) interprets the effective alpha value as an acceptance probability. For each Gaussian primitive $i$ covering pixel $\mathbf{x}$, we independently draw $\hat{\alpha}_i(\mathbf{x}) \sim \mathrm{Bernoulli}(\alpha_i(\mathbf{x}))$ and retain it as an opaque candidate when $\hat{\alpha}_i(\mathbf{x}) = 1$. A depth test selects the nearest retained candidate. Let $\hat{i}(\mathbf{x})$ and $\hat{I}^{\mathrm{ST}}(\mathbf{x})$ denote the selected Gaussian primitive index and the pixel color, respectively. Therefore,

$$\Pr\left[\hat{i}(\mathbf{x}) = i\right] = \Pr\left[\hat{\alpha}_i(\mathbf{x}) = 1\right] \prod_{z_j < z_i} \Pr\left[\hat{\alpha}_j(\mathbf{x}) = 0\right] = \alpha_i(\mathbf{x}) \prod_{z_j < z_i} (1 - \alpha_j(\mathbf{x})) = \alpha_i(\mathbf{x}) T_i(\mathbf{x}).$$

Then we stipple the pixel location $\mathbf{x}$ with the color of the selected Gaussian primitive $i$:

$$\Pr\left[\hat{I}^{\mathrm{ST}}(\mathbf{x}) = \mathbf{c}_i\right] = \Pr\left[\hat{i}(\mathbf{x}) = i\right] = \alpha_i(\mathbf{x}) T_i(\mathbf{x}),$$

Since this process follows the Monte Carlo estimation principle, the expected value $\mathbb{E}[\hat{I}^{\mathrm{ST}}(\mathbf{x})] = I^{\mathrm{GS}}(\mathbf{x})$ is unbiased.

Gaussian Point Splatting (GPS) (Rijsdijk et al., 2026) follows this formulation using an opacity-corrected Poisson point process, which we refer to as *primitive-based sampling*. Specifically, for each sampling process, the number of generated sample points $n$ is drawn from poisson distribution,

$$n \sim \mathrm{Poisson}(\lambda), \qquad \lambda = 2A \,\mathrm{Li}_2(o), \qquad A = \pi\sqrt{|\boldsymbol{\Sigma}'|},$$

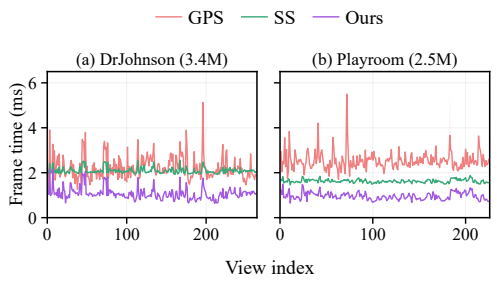


Figure 3: Per-view latency. Ours achieves the fastest speed and avoids the viewpoint-dependent fluctuation of Gaussian Point Splatting.

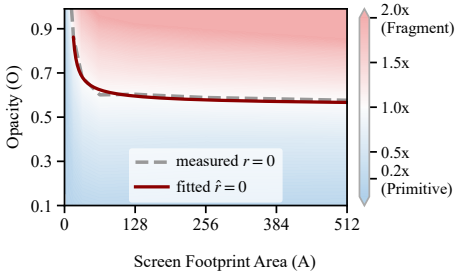


Figure 4: Time-cost ratio of two methods in the synthetic benchmark scene. The gray dotted line is the actual boundary $r = 0$ and the red solid line is the boundary of the fitting result $\hat{r} = 0$.

where $\mathrm{Li}_2$ denotes the dilogarithm function and $\mathbf{\Sigma}'$ is its projected 2D covariance. For each sample, let $LL^\top = \mathbf{\Sigma}$ denote the Cholesky decomposition of the covariance matrix. The pixel location is then sampled as $\mathbf{x} = \boldsymbol{\mu} + L\mathbf{u}$, where $\mathbf{u}$ is drawn from an opacity-corrected normal distribution using the corrected Box–Muller transform:

$$u_0, u_1 \sim \mathrm{Uniform}(0,1),\ r = \sqrt{-2\ln \frac{\mathrm{Li}_2^{-1}(u_0\,\mathrm{Li}_2(\alpha))}{\alpha}},\ \mathbf{u} = (r\cdot\cos(2\pi u_1), r\cdot\sin(2\pi u_1))^\top.$$

On the other hand, StochasticSplats (SS) (Kheradmand et al., 2025) treats the sampling process as *fragment-based sampling*. Each Gaussian primitive $\mathcal{G}$ is expanded into a finite ellipse $\Omega$ bounded by three standard deviations $3\sigma$ and then each fragment in $\Omega$ is accepted with probability $\alpha(\mathbf{x})$.

**Cost-aware routing.** Although GPS is claimed to scale well to scenes with many Gaussians (Rijsdijk et al., 2026), its rendering speed degrades drastically for certain views or scenes, as shown in Figure 3. SS, in contrast, is more consistent across views and can be faster than GPS in some scenes.

Considering the process of these two methods, both primitive-based and fragment-based sampling scale with the projected area $A$ and opacity $o$. In primitive-based sampling, the total number of sampling points $n$ depends on both $A$ and $o$, while higher-opacity gaussian primitives incur greater depth-testing costs. Similarly, in fragment-based sampling, the size of the expanded ellipse $\Omega$ depends on $A$, and higher-opacity gaussian primitives also increase the depth-testing cost. Therefore, we profile their relative time cost $r(o, A) = \ln(t_{\text{primitive}}/t_{\text{fragment}})$ using a synthetic benchmark scene under different $o$ and $A$ (Details of benchmark scene construction and profiling procedure are presented in Section 4.2). The profiling result in Figure 4 shows that the faster sampling method varies with the opacity $o$ and projected area $A$ of Gaussian primitives.

In practice, fragment-based sampling benefits from hardware acceleration to render Gaussian primitives with large $A$, but its cost increases with the number of rasterized fragments due to quad overdraw, particularly for small $A$. In contrast, primitive-based sampling avoids fragment rasterization and is more efficient for primitives with small $A$, but its cost grows rapidly with the total number of sampling points $n$, which depends on both $A$ and $o$.

We therefore use the borderline $r(o, A) = 0$ as the criterion to determine the faster method for each Gaussian primitive, forming a hybrid sampling router and treats the two sampling methods as two streams of the router. To approximate this criterion, we fit the following regression function:

$$\hat{r}(o, A) = \beta_0 + \beta_1 \log_2 A + \beta_2 o + \beta_3 o \log_2 A, \tag{1}$$

where $\beta_0$, $\beta_1$, $\beta_2$ and $\beta_3$ are the coefficients. We route Gaussian $\mathcal{G}$ to the fragment-based stream when $\hat{r}(o, A) > 0$, and to the primitive-based stream otherwise.

Let $\mathcal{A}_P(\mathbf{x})$ and $\mathcal{A}_F(\mathbf{x})$ denote the accepted primitive-based and fragment-based stippling candidates at pixel $\mathbf{x}$, respectively. The hybrid renderer selects that with the smallest depth among all accepted candidates, and stipples its color to pixel $\mathbf{x}$, denoted by

$$\hat{i}(\mathbf{x}) = \underset{p\in\mathcal{A}_P(\mathbf{x})\cup\mathcal{A}_F(\mathbf{x})}{\arg\min} z_p, \qquad \mathbf{c}(\mathbf{x}) \triangleq \hat{I}^{\mathrm{ST}}(\mathbf{x}) = \mathbf{c}_{\hat{i}(\mathbf{x})}.$$

Since the two streams follow the same stochastic transparency formulation, this cross-stream merging driven by our proposed router remains unbiased.

### 3.2 Spatiotemporal Gaussian Observations

Suppose $\mathcal{G} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, o, \mathbf{c}, z\}$ is the final selected Gaussian primitive at pixel location $\mathbf{x}$ in frame $t$ for one observation. We associate it with a 10-channel descriptor

$$s_t(\mathbf{x}) = \left[\mathbf{c}(\mathbf{x}), \alpha(\mathbf{x}), \delta^2(\mathbf{x}), o, \mathbf{q}, z^{-1}\right],$$

where $\delta^2(\mathbf{x}) = (\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ is the squared Mahalanobis distance to the projected Gaussian center $\boldsymbol{\mu}$, $\mathbf{q} = (\boldsymbol{\Sigma}^{-1}_{0,0}, \boldsymbol{\Sigma}^{-1}_{0,1}, \boldsymbol{\Sigma}^{-1}_{1,1})^\top$ contains the independent coefficients of inverse of the projected 2D covariance matrix $\boldsymbol{\Sigma}$, and $z^{-1}$ denotes the inverse depth. Particularly, opacity and Mahalanobis distance help distinguish sampling variations from Gaussian footprint boundaries, while covariance and depth provide anisotropy and visibility cues.

For a fixed camera pose $C_t$, we independently sample $M$ observations and average their descriptors channel-wise to obtain a fixed-dimensional observation map $\mathcal{O}_t^M(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} s_t^{(m)}(\mathbf{x})$. Using the current frame and $K = 3$ historical frames through forward warping (Lin et al., 2021), the network input is $X_t(\mathbf{x}) = \mathrm{concat}_{k=0}^{K} \tilde{\mathcal{O}}^M_{t-k\to t}(\mathbf{x})$, where $\tilde{\mathcal{O}}^M_{\tau\to t}$ denotes the observation map from frame $\tau$ after forward reprojection to frame $t$, with $\tilde{\mathcal{O}}^M_{t\to t} \triangleq \mathcal{O}^M_t$.

### 3.3 Neural Reconstruction

Given the 40-channel spatiotemporal observation tensor $X_t$, we reconstruct the current RGB image using a fully convolutional network that combines a pixel-wise projection with a U-Net-style encoder–decoder (Ronneberger et al., 2015). The two branches provide complementary information: the pixel-wise branch retains a direct path from full-resolution observations, while the contextual branch aggregates neighboring observations across multiple spatial scales. The reconstruction is formulated as

$$\begin{aligned} F_t^{\mathrm{local}} = P_\theta(X_t), \qquad F_t^{\mathrm{context}} = U_\theta(X_t), \\ \hat{I}_t = H_\theta\big(\mathrm{concat}\left(F_t^{\mathrm{context}}, F_t^{\mathrm{local}}\right)\big). \end{aligned} \tag{2}$$

The pixel-wise branch $P_\theta$ consists of a $1 \times 1$ convolution followed by ReLU. It mixes color, Gaussian attributes, and reprojected temporal observations at each pixel without spatial downsampling. The contextual branch $U_\theta$ uses convolutional feature extraction and $2 \times 2$ max pooling to construct a spatial pyramid. Its decoder progressively restores resolution through nearest-neighbor upsampling and concatenation with encoder features at the corresponding resolutions. At full resolution, $H_\theta$ concatenates the contextual features with the pixel-wise features, applies two $3 \times 3$ convolutions with ReLU, and predicts RGB using a final $3 \times 3$ convolution. This fusion gives the output stage access to both spatial context and directly projected observations. The network processes each assembled input in a feed-forward pass; temporal information is supplied by the reprojected observation channels.

We supervise reconstruction at original dataset views where ground-truth images $I_t^{\mathrm{GT}}$ are available. The objective combines a Charbonnier reconstruction penalty (Charbonnier et al., 1997) with spatial gradient consistency:

$$\mathcal{L} = \mathcal{L}_{\mathrm{char}} + \lambda_{\mathrm{grad}}\mathcal{L}_{\mathrm{grad}}, \tag{3}$$

$$\mathcal{L}_{\mathrm{char}} = \frac{1}{3N}\sum_{p}\sum_{c=1}^{3}\sqrt{\left(\hat{I}_{t,c}(p) - I_{t,c}^{\mathrm{GT}}(p)\right)^2 + \epsilon^2}, \tag{4}$$

$$\mathcal{L}_{\mathrm{grad}} = \sum_{d\in\{x,y\}}\frac{1}{3|\Omega_d|}\sum_{p\in\Omega_d}\left\|\nabla_d \hat{I}_t(p) - \nabla_d I_t^{\mathrm{GT}}(p)\right\|_1. \tag{5}$$

Here, $N$ is the number of supervised pixels, $c$ indexes RGB channels, and $\nabla_d$ denotes a forward finite difference along direction $d$. The set $\Omega_d$ contains pixels with a valid neighbor in that direction. We use $\epsilon = 10^{-3}$ and $\lambda_{\mathrm{grad}} = 0.2$. The Charbonnier term penalizes color reconstruction errors, while the gradient term penalizes discrepancies in local image structure.

Table 1: Quantitative results of quality at the training resolution and throughput at 1080p. PSNR, SSIM, and LPIPS are measured against ground-truth images. * MobileGS fails to optimize Tanks&Temples due to NaN error. † MobileGS's own benchmark source code does not include its view-dependent MLP. We use TensorRT as the inference engine of its MLP since tiny-cuda-nn (Müller, 2021) is not applicable to its MLP architecture. The FPS is computed by MLP + raster.

| Method | Mip-NeRF360 | | | | Tanks&Temples | | | | Deep Blending | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR | SSIM | LPIPS | FPS | PSNR | SSIM | LPIPS | FPS | PSNR | SSIM | LPIPS | FPS |
| SS | **17.29** | **0.243** | **0.894** | 642.08 | **13.30** | **0.175** | **0.996** | 836.87 | **21.61** | **0.356** | **0.806** | 552.48 |
| GPS | 17.28 | 0.241 | 0.895 | 457.50 | **13.30** | 0.175 | **0.996** | 328.93 | 21.60 | 0.356 | 0.807 | 436.71 |
| Ours (1 spp) | 17.28 | 0.241 | 0.895 | **854.34** | 13.29 | 0.174 | 0.997 | **959.07** | **21.61** | 0.356 | 0.807 | **992.43** |
| 3DGS | 28.90 | **0.871** | **0.186** | 133.37 | 23.39 | **0.842** | **0.184** | 123.75 | 29.53 | 0.904 | **0.246** | 120.66 |
| MobileGS† | 28.20 | 0.857 | 0.210 | 234.97 | –* | –* | –* | –* | **30.07** | **0.911** | 0.249 | 292.26 |
| Mini. + Ours-S | 26.92 | 0.806 | 0.261 | **345.31** | 22.31 | 0.772 | 0.283 | **343.51** | 29.48 | 0.898 | 0.270 | **344.11** |
| Ours-S (1 spp) | 27.72 | 0.812 | 0.257 | 283.17 | 22.72 | 0.778 | 0.278 | 305.32 | 29.33 | 0.895 | 0.274 | 301.10 |
| Ours-S (4 spp) | 28.23 | 0.834 | 0.232 | 159.59 | 22.98 | 0.790 | 0.266 | 173.77 | 29.48 | 0.900 | 0.259 | 160.01 |
| Ours-L (16 spp) | **29.08** | 0.861 | 0.206 | 40.43 | **23.68** | 0.825 | 0.214 | 40.36 | 29.70 | 0.907 | 0.251 | 38.05 |




Figure 5: Qualitative results of 1-spp *Ours-S*. A lightweight network reconstructs visually plausible images from sparse stochastic inputs.

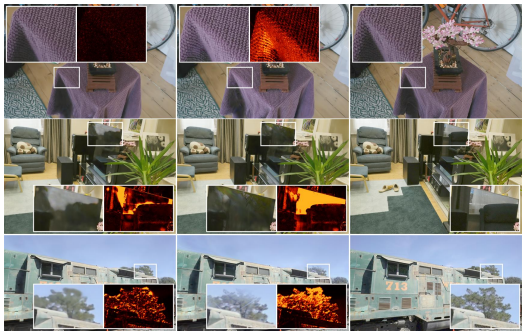


Figure 6: Qualitative results of 16-spp *Ours-L*. Increased sampling and model capacity better preserve lighting effects and fine details.

# 4 Experiments

## 4.1 Experimental Setup

**Datasets.** We evaluate our rendering and per-scene reconstruction performance on 11 scenes from Mip-NeRF 360 (Barron et al., 2022), Tanks and Temples (Knapitsch et al., 2017), and Deep Blending (Hedman et al., 2018), and tested the generalization quality on 5 scenes from the DL3DV dataset. We exclude the 'Flower' and 'Treehill' scenes from MipNeRF 360 (Barron et al., 2022), as these datasets are not publicly accessible. For each scene, we train a separate reconstruction network using the corresponding ground-truth training images and evaluate it on held-out views of the same scene. We insert at least three ($\geq K$) intermediate camera views between consecutive dataset views, constraining the adjacent camera-center displacement to $0.05$ reconstructed scene units and the relative rotation to $3°$. Ground-truth supervision is applied only to training views; interpolated views are rendered from the fixed Gaussian model and used solely as network inputs.

Table 2: Cross-scene generalization of the reconstruction network on five DL3DV scenes.

| Method | PSNR↑ | CVVDP↑ | tPSNR$_\Delta$↑ | FPS |
|---|---|---|---|---|
| 3DGS | 23.37 | 6.106 | 20.46 | - |
| 1-spp | 13.50 | 3.620 | 10.48 | - |
| SS (4-spp + TAA) | 18.64 | 4.914 | 15.76 | 353 |
| GPS (4-spp + TAA) | 19.07 | 3.793 | 16.68 | 283 |
| Ours (1-spp + S) | **22.11** | **5.284** | **19.27** | 353 |

**Baselines.** We compare our hybrid rendering pipeline and reconstruction network against several related state-of-the-art methods. StochasticSplats(SS) (Kheradmand et al., 2025) and Gaussian Point Splatting(GPS) (Rijsdijk et al., 2026) serve as our fragment-based and primitive-based streams, respectively. MobileGS (Du et al., 2026) also eliminate sorting by adopting order-independent transparency (OIT), but requires additional training to the original Gaussian assets. To demonstrate the generality of our approach, we further evaluate it on MiniSplatting (Fang & Wang, 2024).

**Implementation and Training details.** We implement the reconstruction training pipeline in PyTorch and the hybrid rendering and inference pipeline using Vulkan and GLSL, with TensorRT (NVIDIA, 2026) as the neural inference backend. All training and inference experiments are conducted on a GPU with computational performance comparable to an NVIDIA RTX 4090. The scene-specific network is optimized for 2,000 epochs with AdamW (Loshchilov & Hutter, 2017). In each epoch, seven random $256 \times 256$ crops are sampled from every training image. After a 100-epoch linear warmup, the learning rate follows a cosine decay schedule from $1 \times 10^{-4}$ to $5 \times 10^{-6}$. Our compact *Ours-S* and larger *Ours-L* variants both use the same structure formulated in Equation (2), but differ in depth, channel widths, and output activation function. Their detailed structures are provided in the supplementary material.

### 4.2 Hybrid Stochastic Rendering

**Renderer throughput.** The first three lines of Table 1 report renderer-only throughput at 1080p. Ours provides a 7.2–9.2$\times$ speedup over vanilla 3DGS and also outperforms both single-stream sampling baselines, owing to our primitive-wise strategy. Figure 3 further shows that, thanks to our proposed router, ours avoids the severe instability of primitive-based sampling across views.

**Routing Criterion Estimation.** We first describe construction of the synthetic benchmark used to profile the relative time cost $r(o, A) = \ln(t_{\text{primitive}}/t_{\text{fragment}})$. The synthetic scene contains $L$ aligned layers, with a regular $1000 \times 1000$ grid of Gaussians per layer and $10^6 L$ Gaussians in total. All Gaussians share an opacity $o \in \{0.1, 0.2, 0.4, 0.6, 0.8, 0.9, 0.99\}$ and a projected footprint of $A \in \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$ pixels. Under a fixed camera view, for each configuration we vary $L \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, record the median core GPU time at 1080p of the primitive-based and fragment-based streams, $t_{\text{primitive}}$ and $t_{\text{fragment}}$, and compute the relative time cost $r$. The coefficients in Equation (1) are then fitted via ordinary least squares, yielding the fitted function $\hat{r}$. Figure 4 shows the averaged measured and fitted costs; detailed results for each $L$ are provided in the supplementary material. For a fixed $L$, the two streams exhibit substantial runtime discrepancies under extreme combinations of opacity and footprint. Across different values of $L$, the relative cost pattern and boundary remain similar, indicating that inter-layer occlusion has a minimal effect on their relative performance.

### 4.3 End-to-End Reconstruction

**Quality and throughput results.** All but the first three rows of Table 1 compares operating points obtained by varying the observation budget and network capacity. The reported time includes rendering and reconstruction. At 1 spp, *Ours-S* achieves a 2.3–2.7$\times$ speedup over 3DGS while approaching closely its rendering quality. When increasing the observation budget to 16 spp and using larger reconstruction model *Ours-L*, ours achieves comparable or even superior results, with average PSNR above 3DGS by 0.18–0.29 dB across all three benchmarks. In detail, the reconstruction time of *Ours-S* and *Ours-L* is 1.78 ms and 11.31 ms, respectively. Since the reconstruction is performed in screen space, its time remains stable across various scenes and depends on image resolution, making it particularly attractive for 3DGS assets with a large number of primitives.

Figure 5 demonstrates the qualitative result of 1-spp *Ours-S*. Despite using only a single stochastic sample per pixel and a lightweight reconstruction network, our method produces images that closely match the visual quality of standard 3DGS. Figure 6 show 16-spp *Ours-L*. Compared with depth-sorted 3DGS, *Ours-L* reconstructs high-frequency appearance more accurately, yielding finer geometric details.

**Generalization and Temporal Stability** To demonstrate the generalizability of our reconstructor, we train *Ours-S* on 11 scenes from Mip-NeRF360, Tanks and Temples, and Deep Blending, and

Table 3: **Ablation of temporal history and descriptors**.

| Variant | Avg. PSNR | $\Delta$ |
|---|---|---|
| Full descriptor | **26.820** | – |
| RGB only | 26.529 | -0.291 |
| w/o $\alpha$ | 26.808 | -0.012 |
| w/o conic | 26.741 | -0.079 |
| w/o opacity | 26.809 | -0.011 |
| w/o $d_M^2$ | 26.809 | -0.011 |
| w/o history | 26.347 | -0.473 |
| 1 history frame | 26.618 | -0.202 |

Table 4: **Mobile throughput (FPS)** on vivo X300 Pro with Dimensity 9500. *Ours-S* uses FP16 NPU inference.

| Method | Truck | Room | DrJ. | Avg. |
|---|---|---|---|---|
| 3DGS-GL | 5.46 | 12.80 | 3.20 | 7.15 |
| SS | 24.80 | 65.80 | 69.40 | 53.30 |
| GPS | 41.30 | 52.80 | 53.00 | 49.03 |
| MobileGS | – | 26.98 | 25.61 | 26.29 |
| Ours (1 spp) | **78.80** | **112.70** | **86.30** | **92.60** |
| Ours-S | 64.60 | 86.70 | 68.30 | 73.20 |

evaluate the frozen model on 5 unseen DL3DV scenes. We compare ours with GPS and SS, each using its proposed TAA strategy. Specifically, we use 4 spp for GPS and SS but 1 spp for ours, so that all methods run at comparable rendering speeds for a fair quality comparison. All methods use the same Gaussian assets and camera sequences in chronological order.

We further show the temporal stability on these scenes using the ColorVideoVDP (CVVDP) metric (Mantiuk et al., 2024) and reference-relative temporal fidelity measured by $\mathrm{tPSNR}_\Delta$. Table 2 reports scene-averaged PSNR, CVVDP, and $\mathrm{tPSNR}_\Delta$. Without additional temporal accumulation after our reconstruction, *Ours-S* outperforms both evaluated stochastic baselines with TAA across all three metrics, while remaining below standard 3DGS.

**Ablation studies.** All ablations use 1-spp and *Ours-S* with the same training settings and 3DGS-based evaluation on Bonsai, Counter, and Train. Table 3 jointly evaluates temporal history and local Gaussian descriptors. We report mean PSNR against the standard 3DGS target. Per-scene results are provided in the supplementary material. The results show that removing historical observations or retaining only RGB reduces reconstruction quality, which demonstrates that historical observations and Gaussian attributes provide complementary information. Individual descriptor removals produce subtle but consistent degradations. Notably, effective alpha remains beneficial even when opacity and Mahalanobis distance are provided, suggesting that explicitly exposing their nonlinear interaction helps the compact network.

### 4.4 Application: Mobile Deployment and Simultaneous Graphics and Compute

We further implement our method on a mobile platform to evaluate its applicability on resource-constrained devices. At 540p, our renderer averages 92.6 FPS, versus 53.3 FPS (SS) and 49.0 FPS (GPS). With our compact reconstruction network, the full system reaches 73.2 FPS end-to-end on mobile NPUs while maintaining temporal stability. This corresponds to about $10.24\times$ the throughput of 3DGS-GL based on Splatapult (Thibault., 2023). As dedicated AI accelerators, such as NPUs and Tensor Cores, become increasingly common across modern computing platforms, we expect concurrent graphics rendering and neural inference to become more prevalent, further improving the practicality and scalability of our approach.

## 5 Conclusion

We presented a Gaussian stippling framework for efficient sorting-free 3D Gaussian rendering. By converting continuous Gaussian splats into discrete stochastic visibility samples, our hybrid sampler routes each Gaussian to primitive- or fragment-driven sampling without modifying the underlying 3DGS assets. Gaussian-aware spatiotemporal reconstruction then aggregates current and historical stochastic evidence to recover high-quality images from limited samples. The framework supports both direct cross-scene deployment through a scene-agnostic reconstruction model and a scene-specific configuration that prioritizes visual fidelity.

### AI USE STATEMENT

We used generative AI tools to assist with software development for mobile NPU deployment and to improve the clarity and readability of the English manuscript. The research methodology and design, the core hybrid rendering framework, and the implementation and validation of the research prototype were completed by the authors without generative AI assistance. We reviewed the AI-assisted text and code and verified the mobile NPU implementation through testing on the target devices. We take full responsibility for the final manuscript, including all AI-assisted content and code.

### REPRODUCIBILITY STATEMENT

To support reproducibility, the Method section describes our hybrid sampling procedure, spatiotemporal observations, reconstruction networks, and training objective. The Experiments section specifies the datasets, scene selection, temporal sequence generation, optimization settings, and evaluation resolutions, as well as the calibration procedure for the routing criterion. The supplementary material provides detailed network architectures, the DL3DV scene identifiers used in our evaluation, and additional implementation details and experimental results.

## REFERENCES

Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *European conference on computer vision*, pp. 696–712. Springer, 2020.

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4):97–1, 2017.

Martin Balint, Krzysztof Wolski, Karol Myszkowski, Hans-Peter Seidel, and Rafał Mantiuk. Neural partitioning pyramids for denoising monte carlo renderings. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–11, 2023.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 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.

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)*, 39(4):148–1, 2020.

Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36 (4):1–12, 2017.

Pierre Charbonnier, Laure Blanc-Féraud, Gilles Aubert, and Michel Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on image processing*, 6(2): 298–311, 1997.

Xiaobiao Du. Mobile-gs github source code, 2026. URL `https://github.com/xiaobiaodu/mobile-gs`.

Xiaobiao Du, Yida Wang, Kun Zhan, and Xin Yu. Mobile-gs: Real-time gaussian splatting for mobile devices. 2026, 2026.

Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. Stochastic transparency. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp. 157–164, 2010.

Hangming Fan, Rui Wang, Yuchi Huo, and Hujun Bao. Real-time monte carlo denoising with weight sharing kernel prediction network. In *Computer Graphics Forum*, volume 40, pp. 15–27. Wiley Online Library, 2021.

Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European conference on computer vision*, pp. 165–181. Springer, 2024.

Fan Gao, YIbo Zhao, Changhao Song, Jiarui Wen, Yuxuan Song, Youcheng Cai, and Ligang Liu. Mobile3dgs3: Accelerate mobile 3dgs rendering via gradient-aware super-sampling and frame interpolation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–11, 2026.

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. Sample-based monte carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics (ToG)*, 38(4):1–12, 2019.

Amirhosein Ghasemabadi, Muhammad K Janjua, Mohammad Salameh, and Di Niu. Learning truncated causal history model for video restoration. *Advances in Neural Information Processing Systems*, 37:27584–27615, 2024.

Florian Hahlbohm, Fabian Friederichs, Tim Weyrich, Linus Franke, Moritz Kappel, Susana Castillo, Marc Stamminger, Martin Eisemann, and Marcus Magnor. Efficient perspective-correct 3d gaussian splatting using hybrid transparency. In *Computer Graphics Forum*, volume 44, pp. e70014. Wiley Online Library, 2025.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018.

Nikolai Hofmann, Jon Hasselgren, Petrik Clarberg, and Jacob Munkberg. Interactive path tracing and reconstruction of sparse volumes. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 4(1):1–19, 2021.

Qiqi Hou, Randall Rauwendaal, Zifeng Li, Hoang Le, Farzad Farhadzadeh, Fatih Porikli, Alexei Bourd, and Amir Said. Sort-free gaussian splatting via weighted sum rendering. In *International Conference on Learning Representations*, volume 2025, pp. 5655–5671, 2025.

Yuchi Huo and Sung-eui Yoon. A survey on deep learning-based monte carlo denoising. *Computational visual media*, 7(2):169–185, 2021.

Jose A Iglesias-Guitian, Prajita Mane, and Bochang Moon. Real-time denoising of volumetric path tracing for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2734–2747, 2020.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. doi: 10.1145/3592433.

Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions On Graphics (TOG)*, 43(4):1–15, 2024.

Shakiba Kheradmand, Delio Vicini, George Kopanas, Dmitry Lagun, Kwang Moo Yi, Mark Matthews, and Andrea Tagliasacchi. Stochasticsplats: Stochastic rasterization for sorting-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 26326–26335, 2025.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

Daqi Lin, Chris Wyman, and Cem Yuksel. Fast volume rendering with spatiotemporal reservoir resampling. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021.

Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, et al. Dl3dv-10k: A large-scale scene dataset for deep learning-based 3d vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22160–22169, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Rafal K Mantiuk, Param Hanji, Maliha Ashraf, Yuta Asano, and Alexandre Chapiro. Colorvideovdp: A visual difference predictor for image, video and display distortions. *arXiv preprint arXiv:2401.11485*, 2024.

Jan U Müller, Robin Tim Landsgesell, Leif Van Holland, Patrick Stotko, and Reinhard Klein. Moment-based 3d gaussian splatting: Resolving volumetric occlusion with order-independent transmittance. *arXiv preprint arXiv:2512.11800*, 2025.

Thomas Müller. tiny-cuda-nn, 4 2021. URL `https://github.com/NVlabs/tiny-cuda-nn`.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger 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.

NVIDIA. Tensorrt, 2026. URL `https://developer.nvidia.com/tensorrt/`.

Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15969–15979, 2022.

Joris Rijsdijk, CHRISTOPH Peters, MICHAEL Weinmann, and Ricardo Marroquim. Gaussian point splatting. *ACM Transactions on Graphics (TOG)*, 45(4):1–11, 2026.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):1–14, 2022.

Daniel Scherzer, Lei Yang, and Oliver Mattausch. Exploiting temporal coherence in real-time rendering. In *ACM SIGGRAPH ASIA 2010 Courses*, pp. 1–26. 2010.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of high performance graphics*, pp. 1–12. 2017.

Matias Tassano, Julie Delon, and Thomas Veit. Fastdvdnet: Towards real-time deep video denoising without flow estimation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1351–1360. IEEE, 2020.

Anthony J. Thibault. Splatapult, 2023. URL `https://github.com/hyperlogic/splatapult`.

Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *Acm Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (TOG)*, 37(4):1–15, 2018.

Lei Yang, Shiqiu Liu, and Marco Salvi. A survey of temporal antialiasing techniques. In *Computer graphics forum*, volume 39, pp. 607–621. Wiley Online Library, 2020.

Xijie Yang, Linning Xu, Lihan Jiang, Dahua Lin, and Bo Dai. Virtualized 3d gaussians: Flexible cluster-based level-of-detail system for real-time rendering of composed scenes. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–11, 2025.

Keyang Ye, Hongzhi Wu, and Kun Zhou. Depth peeling for high-fidelity gaussian-enhanced surfel rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2026.

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5728–5739, 2022.

Shilin Zhu, Xianyao Zhang, Gerhard Röthlin, Marios Papas, and Mark Meyer. Denoising production volumetric rendering. In *ACM SIGGRAPH 2023 Talks*, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3587421.3595432.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer graphics forum*, volume 34, pp. 667–681. Wiley Online Library, 2015.

# A LIMITATIONS

Our main limitation is that reconstruction from sparse stochastic observations can over-smooth high-frequency appearance, particularly around thin structures, sharp visibility transitions, and view-dependent details. This behavior reflects the limited evidence available at low sampling rates. A promising direction is unified end-to-end per-scene training that jointly optimizes the Gaussian representation, sampling strategy, and reconstruction network for the final image objective. Such a quality-oriented mode could better preserve fine detail, at the cost of generalizability and low deployment overhead that motivate our scene-agnostic setting.

# B MOBILE BASELINE AND DEPLOYMENT

Despite recent years of development, efficient 3D Gaussian Splatting (3DGS) on mobile devices remains an open challenge, primarily bottlenecked by massive Gaussian primitive counts and view-dependent sorting time. While methods like SortFreeGS Hou et al. (2025) and MobileGS Du et al. (2026) attempt to address this, they remain closed-source. To overcome these computational bottlenecks, we propose a GPU-NPU parallel framework. Evaluated on a Vivo device equipped with a MediaTek NPU, our approach achieves state-of-the-art (SOTA) rendering speeds (see supplementary video). Table 4 shows the quantitative result of Mobile 540p throughput (FPS) on vivo x300pro with MediaTek tianji9500.

## B.1 MOBILEGS REPRODUCTION

We faithfully reproduced the optimizations in the released Mobile-GS source code Du (2026) , including its view-dependent MLP and sort-free rasterizer. Unlike original 3DGS, Mobile-GS avoids depth sorting by constructing unsorted per-tile Gaussian lists and using order-independent weighted color and transmittance accumulation; we preserve its preprocessing, culling, weighting, and compositing equations.

For mobile deployment, we additionally execute the full MLP in FP16 on the MediaTek NPU and the rasterizer in Vulkan compute on the GPU, using zero-copy buffers and overlapping NPU inference for frame $N+1$ with GPU rendering of frame $N$. We introduce no temporal caching, skipped MLP evaluations, or internal resolution scaling.

## B.2 FUTURE WORK

We also test our pipeline on IOS devices using Metal shaders. We find that on a 5-year-old iPhone 13 Pro, our rasterization and neural reconstruction can fully overlap, resulting in $\sim$30 FPS under 720P. And on iPhone 17 Pro, the NPU inference time reduces $3\times$ while GPU raster time stays relatively stable. This leaves us the headroom of using a more powerful neural reconstruction network. In

general, this rendering pipeline can be used on most modern devices, either mobile phone, desktop CPUs (with intregrated graphics and NPUs), or graphic cards with graphic and neural units. We believe that our pipeline will be more favorable as computing devices ship with stronger NPUs.

Table 5: Mobile 540p throughput (FPS) on vivo X300 Pro with Dimensity 9500, using one scene per benchmark. Rendering uses Vulkan and *Ours-S* runs in FP16 on the NPU.

| Method | Truck | Room | DrJohnson | Avg. |
|---|---|---|---|---|
| 3DGS-GL | 5.46 | 12.8 | 3.2 | 7.15 |
| SS | 24.8 | 65.8 | 69.4 | 53.3 |
| GPS | 41.3 | 52.8 | 53.0 | 49.03 |
| MobileGS | – | 26.98 | 25.61 | 26.29 |
| Ours (1-spp) | **78.8** | **112.7** | **86.3** | **92.6** |
| Ours (1-spp + S) | 64.6 | 86.7 | 68.3 | 73.2 |

## C Stochastic Gaussian Stippling

For completeness, we summarize the stochastic sampling used by StochasticSplats Kheradmand et al. (2025) serving as our fragment-based stream, and the opacity-corrected Poisson point process introduced by Gaussian Point Splatting Rijsdijk et al. (2026) serving as our primitive-based stream. This derivation also shows why routing Gaussian primitives between the fragment and primitive streams preserves the expected depth-sorted 3DGS image.

Consider a pixel sample location $\mathbf{x} \in \mathbb{R}^2$ and a projected 2D Gaussian $\mathcal{G}$ with base opacity $o$, mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$, and

$$\begin{aligned} G(\mathbf{x}) &= \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \\ \alpha(\mathbf{x}) &= oG(\mathbf{x}). \end{aligned} \tag{6}$$

If we want to approximate the blending process

$$\Pr\left[\hat{I}^{\mathrm{ST}}(\mathbf{x}) = \mathbf{c}_i\right] = \alpha_i(\mathbf{x})T_i(\mathbf{x}), \tag{7}$$

we can estimate $\alpha(\mathbf{x})$ using a random variable $\hat{\alpha}(\mathbf{x})$, and then perform a depth test to estimate transmittance $T(\mathbf{x})$. Considering that the depth-testing process, which retains only the closest sample, if $\alpha(\mathbf{x})$ follows a Bernoulli distribution,

$$\hat{\alpha}(\mathbf{x}) \sim \mathrm{Bernoulli}(\alpha(\mathbf{x}))\,, \tag{8}$$

then we can guarantee unbiasedness of $\hat{I}^{\mathrm{ST}}$.

For the fragment-based stream proposed in StochasticSplats, each fragment is generated by a Gaussian primitive following the vanilla 3D Gaussian splatting process. The stochasticity is introduced during the fragment shading stage. Specifically, for a fragment located at pixel $\mathbf{x}$, a binary random variable $\hat{\alpha}(\mathbf{x})$ indicates whether the fragment is retained, where the fragment is discarded with probability $1-\alpha(\mathbf{x})$. By this construction, $\hat{\alpha}(\mathbf{x}) \sim \mathrm{Bernoulli}(\alpha(\mathbf{x}))$.

For the primitive-based stream proposed in Gaussian Point Splatting, we first describe an important property of inhomogeneous Poisson point process. Suppose that the number of points $N$ in a bounded study region $W \subset \mathbb{R}^2$ follows a Poisson distribution with parameter $\lambda_0$, i.e., $N \sim \mathrm{Poisson}(\lambda_0)$. Conditional on $N$, the point locations $\Phi = X_{ii=1}^{N}$ are assumed to be independently and identically distributed according to a probability distribution $P$ with density $p$ on $W$. The resulting point process forms an inhomogeneous Poisson point process. Specifically, for any bounded measurable region $A \subset W$, we have

$$N(A) = \#(\Phi \cap A) \sim \mathrm{Poisson}\left(\lambda_0 \int_A p(x)\,\mathrm{d}x\right). \tag{9}$$

Neglecting the aliasing within pixel region $A$, we have

$$P(N(A) = 0) = e^{-\lambda|A|p(\mathbf{x})} \approx 1 - \alpha(\mathbf{x}), \tag{10}$$

where $\mathbf{x}$ denotes the pixel center of $A$.Further assuming the pixel size $|A| = 1$, we obtain

$$p(\mathbf{x}) = -\frac{1}{\lambda} \ln(1 - \alpha(\mathbf{x})). \tag{11}$$

Recall the construction of Box–Muller transform, and let $LL^\top = \mathbf{\Sigma}$ and $L^{-1}(\mathbf{x} - \boldsymbol{\mu}) \triangleq (r\cos\theta, r\sin\theta)$, we have

$$p(\mathbf{x}) = p(r, \theta) = -\frac{|L|}{\lambda} r \ln\left(1 - \alpha e^{-\frac{r^2}{2}}\right). \tag{12}$$

If $\theta \triangleq u_0 \sim \mathrm{Uniform}(0, 2\pi)$, then the cumulative distribution function (c.d.f) of $r$ is

$$\begin{aligned} F(r) &= 2\pi \int_0^r -\frac{|L|}{\lambda} \ln\left(1 - \alpha e^{-\frac{u^2}{2}}\right) r \, \mathrm{d}u \\ &= \frac{2\pi|L|}{\lambda} \left(\mathrm{Li}_2(\alpha) - \mathrm{Li}_2\left(\alpha e^{-\frac{r^2}{2}}\right)\right). \end{aligned} \tag{13}$$

Since $F(\infty) = 1$, solving for $\lambda$ gives

$$\lambda = 2\pi\sqrt{|\mathbf{\Sigma}|}\,\mathrm{Li}_2(\alpha), \tag{14}$$

and therefore

$$F(r) = 1 - \mathrm{Li}_2\left(\alpha e^{-\frac{r^2}{2}}\right) / \mathrm{Li}_2(\alpha). \tag{15}$$

Sampling an independent random variable $u_1 \sim \mathrm{Uniform}(0, 1)$ and applying inverse transform sampling with $F(r) = u_1$, we have

$$r = \sqrt{-2\ln\frac{\mathrm{Li}_2^{-1}((1 - u_1)\,\mathrm{Li}_2(\alpha))}{\alpha}} \tag{16}$$

Because the sampling function for $r$ differs from that of the original Box–Muller transform, we refer to this procedure as the opacity-corrected Box–Muller transform.

In summary, sampling the total number of points as $N \sim \mathrm{Poisson}(\lambda)$, followed by opacity-corrected Box–Muller sampling for each point location, constructs an inhomogeneous Poisson point process satisfying $\hat{\alpha}(\mathbf{x}) \sim \mathrm{Bernoulli}(\alpha(\mathbf{x}))$.

## D Route Criterion Estimation

As discussed in the main text, we evaluate the relative time cost of the primitive-based and fragment-based streams under varying layer counts $L \in \{1, \ldots, 8\}$ to understand the impact of inter-layer occlusion.

In Figure 7, we visualize the crossover boundaries—the zero-ratio contours where $t_{\mathrm{primitive}} = t_{\mathrm{fragment}}$ (or $S(o, A) = 0$)—for $L$ ranging from 1 to 8 on a single opacity-area coordinate system. As observed, these eight boundary curves almost overlap. This demonstrates that depth complexity and occlusion relationships have a minimal impact on the relative execution time of the two rendering streams. Consequently, our routing criterion is robust to varying occlusion conditions, justifying the use of a unified routing rule regardless of the scene's depth complexity.

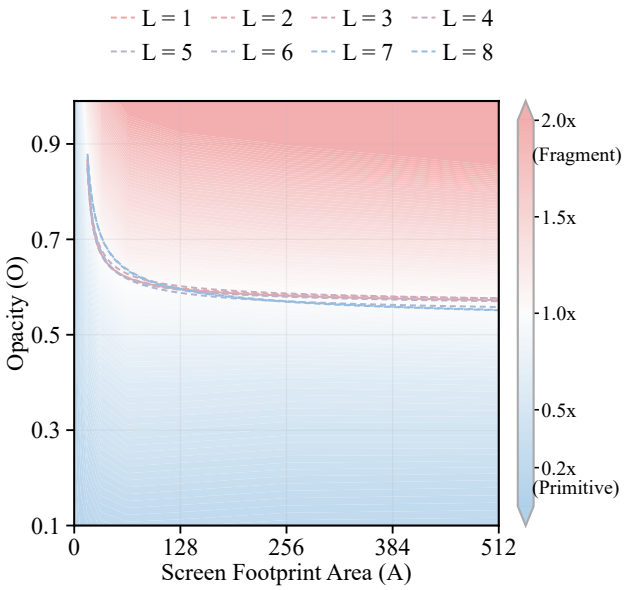


Figure 7: Crossover boundaries ($t_{\text{primitive}} = t_{\text{fragment}}$) for varying layer counts $L \in \{1, \ldots, 8\}$ plotted over the opacity-area plane. The almost perfectly overlapping curves indicate that depth complexity has a negligible effect on the relative performance of the two streams.

Table 6: Scene names and corresponding dataset hash values in DL3DV.

| Scene | Hash Value |
|---|---|
| Pavil. | `d3812aad538261e7f73c75762ff55f23b468bcc76f376d52ac86ca6cf3c44b4b` |
| Stat. | `c37109a55effe0000f8e40652ca935376e75bcb2a0b56de8eabd20a26e2a0f68` |
| Flow. | `1d6a9ed47cce39fd1c4d18f776bcc97e507b81bde921ca596bd91b0b02b5e414` |
| Plant | `dafa9c7cbda9d1ddaa8a2b51fc8c54f4eb44161f5e5c53685dc744580ca77751` |
| House | `6d81c5ab0d480fd43d78b75ff372a8113ad38e2c03f1d69627c009883054d4c2` |

# E Implement Details

## E.1 Dataset

We use custom scene names for clarity when presenting the DL3DV Ling et al. (2024) scenes. The corresponding hash identifiers in the dataset are listed in Table 6.

## E.2 Metrics

We details the measurement of CVVDP and $\text{tPSNR}_\Delta$ as follows. CVVDP measures joint spatial and temporal perceptual quality at an assigned playback rate of 30 fps. With RGB values normalized to $[0, 1]$, we define $e_t = \hat{I}_t - I_t^{\text{GT}}$ and $\text{tPSNR}_\Delta = -10 \log_{10} \text{mean}_{t,\mathbf{x},c}[(e_t - e_{t-1})^2]$, where the mean covers adjacent frames, pixels, and color channels within each scene. This metric measures errors in inter-frame changes without motion compensation.

## E.3 Compare with Gaussian Point Splatting

Our runtime comparison with GPS strictly follows their original implementation, with one exception: we disabled its upper limit on single-Gaussian sampling. As illustrated in the Figure 8, artificially capping this limit leads to severe artifacts from certain viewpoints. As a concurrent work, the only architectural difference between our Fragment Stream and GPS lies in the retrieval of Gaussian

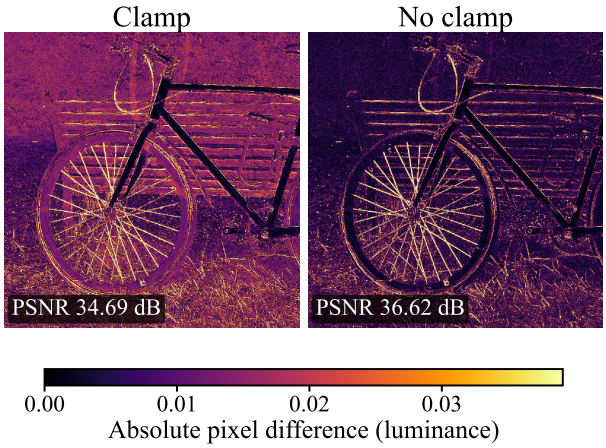


Figure 8: After 1024spp rendering using GPS's source code, the result was compared with that of Gaussian rendering, and there was a loss. The only modification to Gaussian Point Splatting is that we removed the upper limit on individual Gaussian sampling.




Figure 9: Gaussian Points Platting's workload distribution will lost a lot of detail while aligning 3DGS bounding box rendering conventional resolution.

IDs. Specifically, we launch $N_p$ threads and perform a binary search over a prefix-sum array to locate the corresponding Gaussian IDs, whereas they utilize a pre-allocated fixed array. Theoretically, this binary search introduces global memory queries with a time complexity of $\mathcal{O}(N_{\text{sample}} \log N_{\text{Gaussian}})$. In practice, however, the observed runtime difference is negligible. We attribute this to the fact that the rendering pipeline is primarily compute-bound rather than memory I/O-bound. Furthermore, GPS imposes a hard hyperparameter limit of $2.5 \times 10^8$ total sampling points, which triggers forced culling and ultimately results in rendering failures, as depicted in Figure 9. While increasing this hyperparameter can alleviate the issue, we typically lack prior knowledge of novel scene characteristics and arbitrary user viewpoints. Consequently, it is intractable to determine an optimal amount of VRAM for pre-allocation (e.g., a cap of $2.5 \times 10^8$ sampling points consumes approximately 1 GB of memory).

### E.4 Network Architecture

**Network architecture.** We propose two variants *Ours-S* and *Ours-L* to test the limit of the reconstructor. Both variants take the current frame and $K = 3$ reprojected history frames as input. Each pixel $\mathbf{x}$ contributes 10 channels:

- RGB color $\mathbf{c}(\mathbf{x})$.
- alpha $\alpha(\mathbf{x})$.

- squared Mahalanobis distance
$$\delta^2(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}).$$
- opacity $o$ of the selected Gaussian primitive.
- independent projected 2D covariance coefficients of the selected Gaussian primitive
$$\mathbf{q} = (\boldsymbol{\Sigma}^{-\mathbf{1}}_{0,0}, \boldsymbol{\Sigma}^{-\mathbf{1}}_{0,1}, \boldsymbol{\Sigma}^{-\mathbf{1}}_{1,1})^\top.$$
- inverse depth $z^{-1}$ of the selected Gaussian primitive.

These concatenated features result in $(K+1) \times 10 = 40$ input channels.

$$F_t^{\text{local}} = P(X_t), \qquad F_t^{\text{context}} = U_\theta(X_t),$$
$$\hat{I}_t = H_\theta\big(\text{concat}\left(F_t^{\text{local}}, F_t^{\text{context}}\right)\big),$$

Both variants use encoder–decoder U-Net architectures without normalization. Hidden convolutions are followed by ReLU activations. Downsampling uses $2 \times 2$ max pooling, while each decoder stage uses nearest-neighbor upsampling, concatenation with the corresponding encoder feature, and two $3 \times 3$ convolutions.

*Ours-S* uses three downsampling stages with a constant feature width of 16 channels. Its 40-channel input is first projected to 16 channels using a $1 \times 1$ convolution. A separate $1 \times 1$ convolution projects the full-resolution 40-channel input to a 16-channel skip feature.

*Ours-L* uses four downsampling stages with encoder widths $64, 96, 128, 192$, and $256$. At full resolution, its 40-channel input is projected to a 32-channel skip feature using a $1 \times 1$ convolution.

A final $3 \times 3$ convolution maps the decoder features to three RGB channels. *Ours-S* uses a linear output, whereas *Ours-L* applies ReLU after the final convolution. Both variants predict RGB directly and do not use residual output addition.

The renderer-supervised generalization model uses the same *Ours-S* architecture and loss as the scene-specific model.

Table 7 shows the network details. Table 8,Table 9 and Table 10 shows the reconstruction quality on differrent input noise levels and network sizes.

### E.5 Temporal Accumulation

In our interactive GUI app, we use a simple reprojected temporal accumulation of reconstructed images to further improve the temporal stability.

The accumulated result from the previous frame is forward-reprojected into the current camera, with depth competition used to retain the closest history sample. Valid history is combined with the current denoised output using an incremental average with a capped history length:

$$A_t(\mathbf{x}) = \left(1 - \frac{1}{N_t}\right) \widetilde{A}_{t-1}(\mathbf{x}) + \frac{1}{N_t} D_t(\mathbf{x}),$$

where $D_t$ is the current denoised frame and $\widetilde{A}_{t-1}$ is the forward-reprojected accumulated result. We use $N_t = 12$ for a stationary camera and $N_t = 2$ during camera motion. Invalid history is replaced by $D_t$.

## F Extended Ablations

### F.1 Temporal Ablations

We evaluated various temporal reprojection strategies, including forward and backward reprojection, combined with nearest-neighbor and bilinear sampling methods Table 11. Our comprehensive comparisons indicate that forward nearest-neighbor reprojection achieves the optimal performance.

| Configuration | Ours-S | Ours-L |
|---|---|---|
| Input / output channels | $40/3$ | $40/3$ |
| Down-/upsampling stages | $3/3$ | $4/4$ |
| Resolution pyramid | $H$ to $H/8$ | $H$ to $H/16$ |
| Encoder widths, fine to coarse | $16, 16, 16, 16$ | $64, 96, 128, 192, 256$ |
| Encoder/bottleneck convolutions per resolution | $2, 1, 1, 2$ (first stage: $1\times1$, then $3\times3$) | $2, 2, 2, 2, 2$ (all $3\times3$) |
| Decoder transitions, coarse to fine | $32\to16\to16$ at $H/4$; $32\to16\to16$ at $H/2$; $32\to16\to16$ at $H$ | $384\to192\to192$ at $H/8$; $288\to128\to128$ at $H/4$; $192\to96\to96$ at $H/2$; $128\to64\to64$ at $H$ |
| Convolutions per decoder stage | two $3\times3$ | two $3\times3$ |
| Full-resolution skip projection | $40\to16$ ($1\times1$) | $40\to32$ ($1\times1$) |
| Output head | $16\to3$ ($3\times3$) | $64\to3$ ($3\times3$) |
| Activation / normalization | hidden ReLU, linear output / none | ReLU including output / none |
| Output formulation | direct RGB | direct RGB |
| Downsampling / upsampling | max pool / nearest | max pool / nearest |
| Total convolutional layers | 14 (12 conv-3 + 2 conv-1) | 20 (19 conv-3 + 1 conv-1) |
| Parameters | 34,179 | 3,880,803 |
| FP16 kernel-weight size | 0.0648 MiB | 7.397 MiB |

Table 7: Detailed reconstruction-network architectures. A decoder transition $a \to b \to c$ denotes the channel count after skip concatenation and the outputs of the two subsequent $3 \times 3$ convolutions. Both variants use concatenative encoder skips, a projected full-resolution input skip, max-pool downsampling, nearest-neighbor upsampling, and direct RGB prediction. Parameter counts include convolutional biases. FP16 kernel-weight sizes exclude biases, activations, and runtime workspace.

Table 8: Per-scene and per-dataset reconstruction quality at 1 spp, measured against ground-truth images. The Ours-S results use the floating-point RGB evaluation protocol of Table 1, with VGG-based LPIPS and the same SSIM implementation. Dataset means are computed before rounding; the final row averages all 11 scenes equally.

| Scene | Ours-S (1 spp) | | | Ours-L (1 spp) | | |
|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Bicycle | 24.0400 | 0.6683 | 0.3132 | 24.6107 | 0.6996 | 0.2745 |
| Bonsai | 30.7497 | 0.9142 | 0.2469 | 32.1253 | 0.9284 | 0.2303 |
| Counter | 28.2665 | 0.8761 | 0.2540 | 28.9801 | 0.8927 | 0.2317 |
| Garden | 25.3956 | 0.7410 | 0.2352 | 26.0137 | 0.7769 | 0.1891 |
| Kitchen | 29.2107 | 0.8765 | 0.1960 | 30.0836 | 0.8948 | 0.1703 |
| Room | 30.5952 | 0.8924 | 0.2740 | 31.3050 | 0.9048 | 0.2527 |
| Stump | 25.7882 | 0.7155 | 0.2822 | 26.1589 | 0.7371 | 0.2509 |
| **Mip-NeRF 360 Avg.** | **27.7209** | **0.8120** | **0.2573** | **28.4682** | **0.8335** | **0.2285** |
| Train | 21.4527 | 0.7471 | 0.2948 | 22.1772 | 0.7800 | 0.2476 |
| Truck | 23.9940 | 0.8089 | 0.2617 | 24.7056 | 0.8376 | 0.2081 |
| **Tanks & Temples Avg.** | **22.7234** | **0.7780** | **0.2782** | **23.4414** | **0.8088** | **0.2279** |
| DrJohnson | 28.7965 | 0.8925 | 0.2738 | 29.1139 | 0.9010 | 0.2565 |
| Playroom | 29.8633 | 0.8979 | 0.2748 | 30.1277 | 0.9085 | 0.2550 |
| **Deep Blending Avg.** | **29.3299** | **0.8952** | **0.2743** | **29.6208** | **0.9048** | **0.2558** |
| **All Scenes Avg.** | **27.1048** | **0.8209** | **0.2642** | **27.7638** | **0.8419** | **0.2333** |

Traditional temporal information reuse typically relies on either forward or backward reprojection, both of which exhibit inherent limitations. Specifically, forward reprojection necessitates hole-filling to handle missing information (e.g., disocclusions), whereas backward reprojection inevitably introduces motion-blur artifacts (as illustrated in Figure 10). Based on our numerical analysis, addressing the hole-filling problem using a neural network of a fixed capacity is significantly more tractable than mitigating the artifacts induced by backward reprojection.

Table 9: Per-scene and per-dataset reconstruction quality of the Small and Large variants at 4 spp, measured against ground-truth images.

| Scene | Small (4 spp) | | | Large (4 spp) | | |
|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Bicycle | 24.5220 | 0.7018 | 0.2799 | 24.9213 | 0.7212 | 0.2614 |
| Bonsai | 31.2651 | 0.9232 | 0.2346 | 32.4673 | 0.9317 | 0.2279 |
| Counter | 28.5755 | 0.8876 | 0.2357 | 29.1298 | 0.8981 | 0.2263 |
| Garden | 26.1494 | 0.7916 | 0.1903 | 26.5915 | 0.8104 | 0.1662 |
| Kitchen | 29.9029 | 0.8948 | 0.1701 | 30.5050 | 0.9057 | 0.1595 |
| Room | 30.9545 | 0.9004 | 0.2581 | 31.5079 | 0.9085 | 0.2485 |
| Stump | 26.2412 | 0.7415 | 0.2563 | 26.4615 | 0.7535 | 0.2404 |
| **Mip-NeRF 360 Avg.** | **28.2301** | **0.8344** | **0.2321** | **28.7978** | **0.8470** | **0.2186** |
| Train | 21.5588 | 0.7535 | 0.2913 | 22.1426 | 0.7808 | 0.2489 |
| Truck | 24.3956 | 0.8274 | 0.2398 | 25.0175 | 0.8451 | 0.2083 |
| **Tanks & Temples Avg.** | **22.9772** | **0.7904** | **0.2656** | **23.5800** | **0.8130** | **0.2286** |
| DrJohnson | 28.9455 | 0.8978 | 0.2654 | 29.1475 | 0.9029 | 0.2562 |
| Playroom | 30.0227 | 0.9019 | 0.2528 | 30.2009 | 0.9086 | 0.2529 |
| **Deep Blending Avg.** | **29.4841** | **0.8999** | **0.2591** | **29.6742** | **0.9058** | **0.2545** |
| **All Scenes Avg.** | **27.5030** | **0.8383** | **0.2431** | **28.0084** | **0.8515** | **0.2270** |

Table 10: Per-scene and per-dataset reconstruction quality at 16 spp, measured against ground-truth images. Ours-L PSNR and VGG-based LPIPS use the floating-point RGB evaluation protocol of Table 1. Ours-L per-scene and all-scene SSIM values are retained from the earlier evaluation; its dataset-level SSIM values follow Table 1. Dataset means for PSNR and LPIPS are computed before rounding; their final-row values average all 11 scenes equally.

| Scene | Ours-S (16 spp) | | | Ours-L (16 spp) | | |
|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Bicycle | 24.9336 | 0.7303 | 0.2550 | 25.1837 | 0.7470 | 0.2442 |
| Bonsai | 31.7702 | 0.9304 | 0.2228 | 32.8727 | 0.9350 | 0.2205 |
| Counter | 28.7965 | 0.8974 | 0.2180 | 29.2864 | 0.9000 | 0.2175 |
| Garden | 26.7682 | 0.8296 | 0.1487 | 27.0923 | 0.8450 | 0.1432 |
| Kitchen | 30.4227 | 0.9098 | 0.1461 | 30.7830 | 0.9150 | 0.1480 |
| Room | 31.2549 | 0.9094 | 0.2407 | 31.7049 | 0.9110 | 0.2414 |
| Stump | 26.4946 | 0.7595 | 0.2347 | 26.6540 | 0.7660 | 0.2300 |
| **Mip-NeRF 360 Avg.** | **28.6344** | **0.8523** | **0.2094** | **29.0824** | **0.861** | **0.2064** |
| Train | 21.7615 | 0.7754 | 0.2633 | 22.1796 | 0.7880 | 0.2392 |
| Truck | 24.7443 | 0.8469 | 0.2036 | 25.1851 | 0.8560 | 0.1887 |
| **Tanks & Temples Avg.** | **23.2529** | **0.8112** | **0.2335** | **23.6823** | **0.825** | **0.2139** |
| DrJohnson | 29.0140 | 0.8998 | 0.2544 | 29.1605 | 0.8940 | 0.2524 |
| Playroom | 30.1152 | 0.9062 | 0.2419 | 30.2362 | 0.9030 | 0.2500 |
| **Deep Blending Avg.** | **29.5646** | **0.9030** | **0.2482** | **29.6983** | **0.907** | **0.2512** |
| **All Scenes Avg.** | **27.8251** | **0.8541** | **0.2208** | **28.2126** | **0.8605** | **0.2159** |

Table 11: Temporal transport under stochastic visibility. Backward gathering produces structured artifacts; bilinear transport spreads samples, whereas nearest-forward transport preserves discrete observations. The best and second-best PSNR (dB) values are shown in **bold** and underlined.

| Variant | Bonsai | Counter | Train | Avg. |
|---|---|---|---|---|
| Forward Nearest. | **30.75** | <u>28.23</u> | **21.50** | **26.83** |
| Forward Bilinear. | 30.60 | 28.19 | 21.38 | 26.72 |
| Backward Nearest. | 30.59 | 28.21 | 21.40 | 26.73 |
| Backward Bilinear. | <u>30.68</u> | **28.25** | <u>21.40</u> | <u>26.78</u> |

Furthermore, regarding the processing of historical samples, we investigated both nearest-neighbor and bilinear interpolation. Under the forward reprojection framework, nearest-neighbor sampling




Figure 10: Temporal Ablations. Backward reprojection reveals artifacts, while forward reprojection results in black holes.

consistently outperforms the bilinear approach. We attribute this to the fact that nearest-neighbor sampling better preserves the spatial independence and structural integrity of the original samples.

### F.2 Obsevation Time and Network Size Ablations

We comprehensively evaluated the rendering quality across various scenes using combinations of different sampling rates (1, 4, and 16 spp) and network capacities. Our findings indicate that the compact network *Ours-S* demonstrates remarkable capabilities in denoising, significantly mitigating noise artifacts. On the other hand, the high-capacity network *Ours-L* excels in capturing complex illumination effects and intricate local geometric details. Qualitative comparisons supporting these observations are provided in Figures 5 and 6 of the main text, as well as in Appendix Section G. Furthermore, as the temporal observation window extends, the temporally aggregated samples yield a more accurate representation of view-dependent information, consistently resulting in an increase in PSNR.

In terms of practical implementation, we align our framework with traditional rendering pipelines by integrating Temporal Anti-Aliasing (TAA) subsequent to the neural denoising module. As demonstrated in our supplementary video, this integration effectively resolves the flickering issues typically encountered in the extreme 1 spp + *Ours-S* configuration. Notably, operating at a speed three times faster than Vanilla 3DGS, our method not only requires less rendering time but also outperforms the TAA modules proposed by StochasticSplat and GaussianPointSplatting, respectively, achieving a PSNR improvement of approximately 4 dB on the truck scene.

## G Additional Qualitative Results

Figure 11 shows large networks can reconstruct complex lighting and local details.

## H Comparison with a Causal Video Denoising Baseline

We examine whether a general-purpose causal video restoration network can replace our compact Gaussian-aware reconstruction module. We train the official TURTLE-T1 architecture (Ghasemabadi et al., 2024) on source-scene 1-spp observations and evaluate the frozen model on the same five chronological DL3DV sequences as Table 2. TURTLE and Ours-S achieve similar mean PSNR (22.040 versus 22.113 dB), while their current-frame model inference takes 700.56 and 3.81 ms, respectively, under matched FP32 hardware settings. These results support a practical quality–cost trade-off, rather than statistically significant quality superiority or a controlled architectural comparison.

**Training and model selection.** TURTLE-T1 is trained from random initialization on 11 source scenes using 2,328 four-frame training targets and 338 held-out source-validation targets. Only

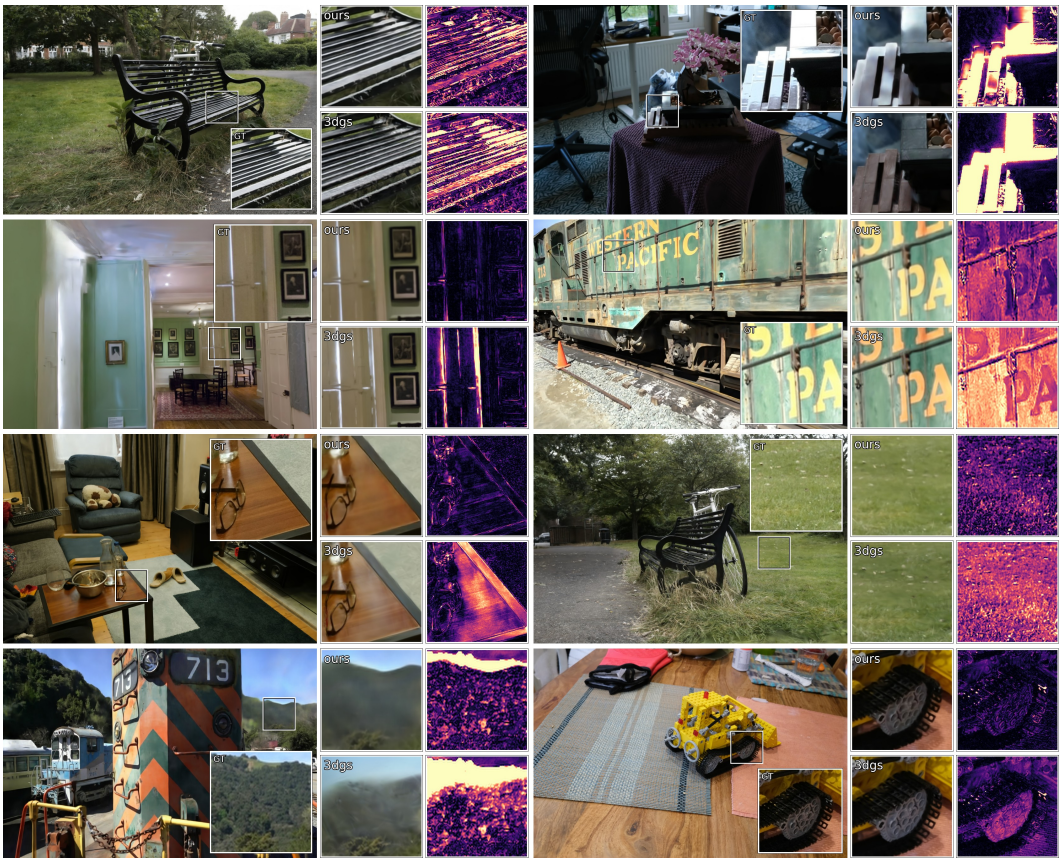


Figure 11: Qualitative results of additional scenes of 16spp + Large network showing neural reconstruction quality

Table 12: Frozen cross-scene reconstruction on the complete chronological DL3DV sequences. All references are photographs; resolution is $960 \times 540$. The final rows average scene scores equally. LPIPS uses AlexNet.

| Scene (frames) | Method | PSNR↑ | SSIM↑ | LPIPS↓ | CVVDP↑ | tPSNR$_\Delta$↑ |
|---|---|---|---|---|---|---|
| Flowers (315) | TURTLE | 21.634 | .6474 | .3645 | 5.792 | 18.746 |
| | Ours-S | 22.049 | .6807 | .2875 | 6.115 | 19.173 |
| House (436) | TURTLE | 23.416 | .7578 | .3283 | 4.898 | 20.946 |
| | Ours-S | 23.099 | .7226 | .3778 | 4.697 | 20.588 |
| Pavilion (401) | TURTLE | 22.487 | .7665 | .3461 | 5.328 | 19.518 |
| | Ours-S | 22.212 | .7177 | .4211 | 5.034 | 19.268 |
| Plant (344) | TURTLE | 21.540 | .6643 | .2742 | 5.499 | 18.651 |
| | Ours-S | 21.807 | .6722 | .2707 | 5.594 | 18.932 |
| Statue (345) | TURTLE | 21.124 | .5643 | .4220 | 4.745 | 18.119 |
| | Ours-S | 21.397 | .5848 | .3523 | 4.978 | 18.391 |
| Scene mean | TURTLE | 22.040 | .6801 | .3470 | 5.252 | 19.196 |
| | Ours-S | 22.113 | .6756 | .3419 | 5.284 | 19.270 |

photographic ground truth supervises the last frame of each training group. The model receives noisy RGB observations without added synthetic noise, Gaussian descriptors, or future frames. Training uses $192 \times 192$ crops and a total batch size of four across two GPUs. The run takes 13.1 hours for 50,000 updates. This is the prescribed optimization budget, not evidence of full convergence. The checkpoint is selected exclusively using source-validation PSNR; no DL3DV scene is used for training, hyperparameter selection, or checkpoint selection.

Table 13: Model-only current-frame inference on one RTX A6000: FP32, TF32 disabled, $960 \times 540$, batch size one. Mean and median latencies are averaged equally across scenes, using 32 targets per scene and three timing repetitions.

| Method | Parameters | Mean (ms) | Median (ms) | Peak increment (MiB) |
|---|---|---|---|---|
| TURTLE-T1 | 59,079,548 | 700.56 | 700.51 | 5,820.8 |
| Ours-S | 34,179 | 3.81 | 3.80 | 220.5 |

The memory column is the maximum additional PyTorch-allocated memory during the timed forward, above the live pre-forward allocation. Inputs, parameters and already-resident history are excluded from this increment; it is not total deployment memory. Both models remain resident during measurement. Across all scene/repetition means, latency ranges are 699.80–701.00 ms for TURTLE and 3.786–3.824 ms for Ours-S. These repetitions measure runtime variation, not quality uncertainty.

**Continuous-sequence evaluation.** We reuse the exact stochastic inputs, original camera order, and photographic references of Table 2. The first three frames of each scene initialize history; all subsequent frames are scored, yielding 1,841 targets at $960 \times 540$. TURTLE processes frames chronologically and resets its recurrent state only between scenes. Its cached features may summarize observations older than three frames; our reconstruction network instead receives three explicitly reprojected historical observations. Neither method uses future frames. We preserve the main-text scoring protocol, including RGB8 output quantization, scene-equal aggregation, and the spatial and temporal metrics. The defined 30-fps playback rate is a scoring convention, not a measured inference rate or an established acquisition rate.

**Timing protocol.** We distinguish continuous cache-primed inference from a paired model-only microbenchmark. The latter evaluates both models on the same physical RTX A6000 at $960 \times 540$, batch size one, FP32, and with TF32 disabled. We use 32 uniformly spaced targets per scene and three timing repetitions. TURTLE's history is primed before timing its current-frame forward; both methods receive preassembled GPU-resident inputs. CUDA-event timings exclude rendering, geometric reprojection, host–device transfers, disk access, and metric computation. Consequently, these measurements must not be interpreted as complete rendering-pipeline latency. The separate RTX 4090/TensorRT deployment throughput reported elsewhere uses a different precision and implementation and is not used to calculate a direct speedup over this baseline.

**Results and limitations.** TURTLE attains higher PSNR on House and Pavilion, whereas Ours-S scores higher on the other three scenes. Ours-S has slightly better mean LPIPS, CVVDP, and temporal error-difference PSNR, while TURTLE has higher mean SSIM (Table 12). The PSNR difference is only 0.072 dB; a single training run does not establish statistical significance. The measured model-only mean latency ratio is approximately $184\times$ in these implementations, not a complete-pipeline speedup. Continuous TURTLE inference independently averages 700.47 ms per scored frame across scenes. Processing its first three initialization frames takes 1.954–1.958 s for four scenes and 3.608 s for the first scene; these startup costs are excluded from the cache-primed latency comparison.

**Scope of the comparison.** Our frozen generalized checkpoint uses Gaussian descriptors and reprojected history and was originally trained with deterministic rendered targets, whereas the TURTLE checkpoint uses RGB-only inputs and photographic supervision. All evaluation references in this experiment are photographs. These input and supervision differences are disclosed rather than treated as a controlled architectural ablation. Repeated timing runs characterize runtime variation; they do not establish statistical significance for reconstruction quality from a single training run.