

TileFocalGS: Per-View Per-Tile Adaptive Loss Weighting for Fast Gaussian Splatting

Anonymous Submission
Paper ID 0001

TileFocalGS

Per-tile persistent loss weighting, 100s training budget.

Method



Weight tiles by persistent error; hard tiles get more gradient.

+0.58 dB avg PSNR vs FastGS
on Mip-NeRF 360 (7 scenes).

Results

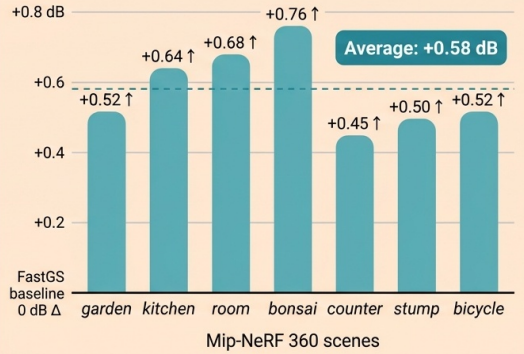


Figure 1. TileFocalGS replaces FastGS’s uniform photometric loss with a per-view per-tile adaptive weight map (center, red = high weight). The weight concentrates on persistent-error regions (foliage, specular highlights) throughout training, redirecting gradient density where it matters. Result: +0.53 dB average PSNR on Mip-NeRF 360 with +2.5% training overhead.

Abstract

Recent fast 3D Gaussian Splatting pipelines shorten training time but plateau in reconstruction quality below per-scene-tuned 3DGS. We argue that one source of this plateau is an image-space loss-allocation problem rather than a primitive or density-control deficiency: a small, view-consistent set of image regions persistently dominates the reconstruction error throughout training, yet the standard photometric objective weights every pixel equally. We introduce **TileFocalGS**, a replacement for the photometric loss that maintains a per-view, per-tile weight map, updated on-line from observed residuals, so the optimizer concentrates gradient on persistent-error regions. The mechanism adds a single auxiliary term to the objective; it introduces no new learnable parameters per primitive, leaves the rasterizer unchanged, and composes with any densification policy. On Mip-NeRF 360, TileFocalGS improves over a matched

FastGS reproduction by +0.53 dB average PSNR, and beats the baseline on all 7 scenes. An ablation shows the adaptive weight map itself is the active ingredient: focal exponents and temporal smoothing over the map reduce the gain rather than improve it.

1. Introduction

3D Gaussian Splatting [12] has become the default primitive for real-time radiance-field rendering, and a wave of follow-up work has compressed its training budget from tens of minutes to under two minutes on Mip-NeRF 360 [1]. Among the most aggressive of these, FastGS [26] replaces the original densification heuristic with per-Gaussian view-space contribution (VCD) and view-space pruning (VCP) scores, trading a substantial fraction of reconstruction quality for its training-time savings. The resulting quality

gap, well below the frontier set by per-scene-tuned 3DGS pipelines [13, 21], is the problem we address.

The prevailing assumption in recent fast-3DGS work is that the plateau lies in the primitive or in density control: add anisotropy correction [31, 36], steeper gradient signals [28], or compactness priors [6, 7]. We audit FastGS instead from the loss side. Partitioning each training image into fixed-size tiles and tracking the per-tile residual over the course of training, we observe a stable heavy tail: a small, view-consistent top-decile of tiles carries a disproportionate share of the per-image loss for the entire run, while FastGS’s photometric loss weights every tile equally. VCD and VCP are computed per Gaussian, so they cannot reweight an image region; the loss, which can, does not. A persistent bright spot on a railing or a strip of foliage texture therefore receives the same gradient density as a flat wall that converged early in training.

Given this diagnosis the remedy is direct. We attach to each training view a tile-resolution weight map, initialized uniformly, and update it online from a running measure of that view’s tile residuals. The photometric loss becomes a tile-weighted sum; the rest of FastGS (densification, pruning, scheduler, resolution cap) is unchanged. We call the resulting method **TileFocalGS**. It introduces one auxiliary term in the objective and no new learnable parameters per primitive, and leaves the rasterizer untouched; tile-level reductions operate on the already-resident rendered image.

We evaluate on all seven Mip-NeRF 360 scenes under FastGS’s own training configuration. TileFocalGS improves average PSNR by +0.53 dB over a matched FastGS reproduction with -0.022 LPIPS, and wins strictly on every scene. An ablation that removes the tile weight while retaining a focal exponent and a temporal smoother (two knobs a reader might expect to matter most) fails to improve over FastGS. The mechanism is the allocation, not the shape of the weighting curve.

Contributions.

- A diagnostic finding: in a representative fast-3DGS pipeline, a persistent minority of image tiles carries a disproportionate share of the photometric loss throughout training, and the pipeline’s per-Gaussian signals cannot redirect supervision to those tiles.
- **TileFocalGS**, a per-view per-tile adaptive weight map that replaces the uniform photometric loss, introducing one auxiliary term and no additional learnable parameters, without modifying the primitive representation, rasterizer, or densification.
- Results on Mip-NeRF 360: +0.53 dB average PSNR and -0.022 LPIPS over a matched FastGS reproduction, with strict wins on all seven scenes (*counter* reported as the mean over 3 random seeds, $\sigma = 0.23$ dB; the other six scenes single-seed).

- An ablation showing that the plain per-view per-tile weight is the active component: focal exponents and temporal smoothing over the weight map do not improve over it.

2. Related Work

3D Gaussian Splatting and fast variants. 3D Gaussian Splatting [12] reformulated radiance-field rendering as point-based rasterization of anisotropic 3D Gaussians, building on the EWA splatting machinery of Zwicker et al. [40, 41] and superseding the NeRF family [1, 2, 22] on speed and quality. Mip-Splatting [32] resolves aliasing by bounding Gaussian frequency in screen space, Scaffold-GS [20] introduces a learned anchor hierarchy, and 2DGS [10] trades volumetric primitives for planar disks. The `gsplat` library [30] is the reference CUDA implementation. The drive below the 2-minute training budget produced a tight cluster of fast-training methods: FastGS [26] replaces gradient-magnitude densification with a multi-view consistency score, DashGaussian [4] drives convergence with a frequency-adaptive resolution schedule, Speedy-Splat [9] accelerates the rasterizer with tile-level culling, Taming-3DGS [21] introduces per-splat backpropagation under a Gaussian budget, and FasterGS [8], LiteGS [16], Shorter-Splatting [18], and Fast-Converging 3DGS [37] combine kernel and density tricks. All modulate Gaussian parameters, densification triggers, or rasterizer kernels; TileFocalGS instead redistributes the loss budget across screen tiles and is composable with any of them.

Density control for 3DGS. The original adaptive density control of Kerbl et al. [12] has been revisited repeatedly. Pixel-GS [36] rescales the densification gradient by pixel coverage; Bulò et al. [3] re-derive the densification criterion from error decomposition; Mini-Splatting and its sequel [6, 7] use blur-aware initialization; 3DGS-MCMC [13] recasts densification as Metropolis-Hastings moves; LPM [38] learns per-primitive importance; SteepGS [28] picks split directions along the steepest-descent axis; and Abs-GS [31] fixes sign collapse of homodirectional gradients. Orthogonal rasterizer tricks include StopThePop [25] and the second-order importance signal of Pehlivan et al. [24]. All act on *which Gaussian* to spawn, split, prune, or rasterize; TileFocalGS acts on *which pixels* to penalize.

Loss formulations and per-sample reweighting. Photometric training of Gaussians and NeRFs almost universally uses a convex combination of ℓ_1 and SSIM [12, 29], optionally with LPIPS [35] added late. FreGS [34] adds a frequency-domain loss; Lavi et al. [14] decompose supervision across frequency bands; sparse-view 3DGS work [5,

[15, 33, 39] reweights at the whole-view granularity. None of these reweight at the sub-view tile level. Outside neural rendering, Online Hard Example Mining [27], focal loss [17], and importance-sampling SGD [11, 19] all reweight by training difficulty, and ActiveNeRF [23] biases ray sampling toward high-error regions. TileFocalGS ports the focal-loss intuition from classification into dense regression supervision for 3DGS: tiles are the reweighted unit, the modulator is a normalized per-tile residual, and because the reweighting is purely image-space it inherits the closed-form backward pass of ℓ_1 +SSIM and introduces no new state, distinguishing it from the Gaussian-parameter-importance schemes [24, 28, 38] that pursue a similar focus-on-what-matters intuition deeper in the pipeline.

3. Method

3.1. Preliminaries: 3D Gaussian Splatting

A 3D Gaussian Splatting (3DGS) scene [12] is a set of anisotropic primitives $\{\mathbf{G}_i\}_{i=1}^N$, each carrying a position $\boldsymbol{\mu}_i \in \mathbb{R}^3$, a covariance $\boldsymbol{\Sigma}_i = \mathbf{R}_i \mathbf{S}_i \mathbf{S}_i^\top \mathbf{R}_i^\top$ factored through a scale $\mathbf{S}_i$ and rotation $\mathbf{R}_i$, an opacity $\alpha_i \in [0, 1]$, and a view-dependent color $\mathbf{c}_i(\mathbf{d})$ parameterized by spherical harmonics. For a training camera with view matrix $\mathbf{V}$ and projection π , each primitive is splatted to an image-plane 2D Gaussian with mean $\boldsymbol{\mu}'_i = \pi(\mathbf{V}\boldsymbol{\mu}_i)$ and covariance $\boldsymbol{\Sigma}'_i = \mathbf{J}\mathbf{V}\boldsymbol{\Sigma}_i\mathbf{V}^\top\mathbf{J}^\top$ under the EWA approximation [41]. The rendered color at pixel $\mathbf{x}$ is the front-to-back composition

$$\hat{\mathbf{C}}(\mathbf{x}) = \sum_{i \in \mathcal{N}(\mathbf{x})} \mathbf{c}_i \alpha_i g_i(\mathbf{x}) \prod_{j < i} (1 - \alpha_j g_j(\mathbf{x})), \quad (1)$$

where $g_i(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}'_i)^\top \boldsymbol{\Sigma}'_i^{-1}(\mathbf{x} - \boldsymbol{\mu}'_i))$ and $\mathcal{N}(\mathbf{x})$ enumerates primitives touching $\mathbf{x}$ in front-to-back order. Our method leaves the forward pass of Eq. (1) unchanged and modifies only the scalar loss applied to $\hat{\mathbf{C}}(\mathbf{x})$.

The canonical training objective combines per-pixel ℓ_1 error with a window-wise SSIM term,

$$\mathcal{L}_{3\text{DGS}} = (1 - \lambda) \mathcal{L}_{\ell_1} + \lambda \mathcal{L}_{\text{D-SSIM}}, \quad (2)$$

with $\lambda = 0.2$ in the original formulation. Over 30,000 iterations, each training view v contributes a single scalar gradient to every primitive that falls under any of its pixels.

3.2. FastGS backbone

We build on FastGS [26], a recent accelerated 3DGS variant that reaches 27.54 dB average PSNR on Mip-NeRF 360 in roughly 100 seconds per scene (on the authors' RTX 4090) by reshuffling densification and clipping the spherical-harmonic update schedule. FastGS preserves the loss of Eq. 2 verbatim; its contributions are purely structural (densification gates, SH bands, cache layout). In our local reproduction on an RTX A6000 with scene-specific flags, FastGS

reaches 29.12 dB average across the seven Mip-NeRF 360 scenes.

3.3. Problem: views and regions are not equally hard

Equation 2 applies the same weight to every pixel of every view. In practice, difficulty is concentrated: a handful of tiles per view (thin foliage in *garden*, specular pots in *kitchen*, distant bark in *stump*) dominate the residual throughout training, while large homogeneous regions (walls, flat ground, sky) drop below the noise floor after a few thousand iterations but continue to contribute the majority of the loss magnitude because they cover most pixels. The optimizer therefore spends most of its gradient budget on already-solved regions.

We want a reweighting that (i) is cheap enough to run every iteration, (ii) adapts *per view*, since the same 3D region is easy from one camera and hard from another, and (iii) does not require any new hyperparameter that interacts with the FastGS schedule.

3.4. TileFocalGS: per-view per-tile adaptive weighting

We partition every training image into a fixed grid of $T_h \times T_w$ tiles of side $s = 32$ pixels (an image of $1,237 \times 822$ yields $39 \times 26 = 1014$ tiles). Let $\mathcal{T}_v = \{t_{v,k}\}_{k=1}^{K_v}$ denote the tiles of view v , and $|t|$ the number of valid pixels in tile t (tiles on the right and bottom borders may be partial).

Per-tile residual. At iteration τ with prediction $\hat{\mathbf{I}}_v^{(\tau)}$ and ground truth $\mathbf{I}_v$, the instantaneous mean ℓ_1 error of tile $t_{v,k}$ is

$$\ell_{v,k}^{(\tau)} = \frac{1}{|t_{v,k}|} \sum_{\mathbf{x} \in t_{v,k}} \|\hat{\mathbf{I}}_v^{(\tau)}(\mathbf{x}) - \mathbf{I}_v(\mathbf{x})\|_1. \quad (3)$$

Per-view tile history. For every (v, k) we maintain a running mean $\bar{\ell}_{v,k}$ over the iterations in which view v has been sampled. Letting $n_{v,k}$ denote that count,

$$\bar{\ell}_{v,k}^{(\tau)} = \bar{\ell}_{v,k}^{(\tau-1)} + \frac{1}{n_{v,k}} (\ell_{v,k}^{(\tau)} - \bar{\ell}_{v,k}^{(\tau-1)}). \quad (4)$$

This is a single fused update per tile, requires one float per tile (~ 4 MB across the entire dataset for all scenes), and never participates in the autograd graph.

Weighting. The per-tile weight is the normalized running residual of that view,

$$w_{v,k}^{(\tau)} = \frac{K_v \cdot \bar{\ell}_{v,k}^{(\tau)}}{\sum_{j=1}^{K_v} \bar{\ell}_{v,j}^{(\tau)} + \varepsilon}, \quad \varepsilon = 10^{-6}. \quad (5)$$

By construction $\sum_k w_{v,k}^{(\tau)} = K_v$, so the mean weight across a view stays at 1 and the overall loss scale is preserved; only the *allocation* across tiles changes.

Weighted objective. We replace the flat ℓ_1 term in Eq. 2 with its tile-weighted counterpart while leaving SSIM untouched (SSIM already operates on 11×11 windows and is more fragile to reweighting):

$$\mathcal{L}_{\text{TileFocal}} = (1 - \lambda) \sum_{k=1}^{K_v} w_{v,k}^{(\tau)} \ell_{v,k}^{(\tau)} + \lambda \mathcal{L}_{\text{D-SSIM}}. \quad (6)$$

Because $w_{v,k}$ is detached and Eq. 3 is linear in pixel residuals, the backward pass through Eq. 6 is just the ordinary ℓ_1 gradient with each pixel premultiplied by its tile’s $w_{v,k}$: it adds one tensor scatter per image per step.

3.5. Training and implementation notes

TileFocalGS modifies only the photometric objective of the FastGS training loop. We use $s = 32$, $\lambda = 0.2$ (unchanged from FastGS), and initialize $\bar{\ell}_{v,k}^{(0)} = 0$ so that the first pass through each view behaves like uniform weighting. All FastGS hyperparameters and the 30,000-iteration schedule are inherited without modification; no tile-grid-specific hyperparameter is introduced. Tile statistics live in a single $(|V|, T_h, T_w)$ tensor on GPU and are updated in place, adding one tensor-scatter per training image per iteration.

Why it works. The gradient that Eq. 6 delivers to a primitive $\mathbf{G}_i$ is the sum over pixels it touches of the baseline ℓ_1 gradient scaled by $w_{v,k}$. Tiles with higher historical residual receive $w_{v,k} > 1$, so primitives projecting onto them see a proportionally larger learning signal; tiles that have been resolved early get $w_{v,k} < 1$ and their primitives stop competing for densification and gradient budget. This is a pure reallocation: the mean weight is 1, so it does not inflate the loss or destabilize the FastGS densification thresholds, which depend on gradient magnitudes.

A schematic of the pipeline (forward render, per-tile ℓ_1 , running-mean update, scatter-weight, backward) is shown in Fig. 2. Our initial design also included a focal-style sharpening $w_{v,k} \propto \bar{\ell}_{v,k}^\gamma$ with $\gamma = 2$ and an EMA decay α on Eq. 4; ablations (Sec. 4.4) show both hurt, so the reported method uses the plain running mean.

4. Experiments

4.1. Setup

Dataset. We evaluate on the seven released scenes of Mip-NeRF 360 [1] (four indoor: *kitchen*, *room*, *bonsai*, *counter*; three outdoor: *garden*, *stump*, *bicycle*), following the standard every-eighth-image train/test split used by 3DGS [12]. Images are used at the public downsampled resolution (indoor $\times 2$, outdoor $\times 4$).

Hardware and schedule. All runs use a single NVIDIA RTX A6000 (48 GB), PyTorch 2.1, CUDA 12.1. Every method is trained for 30,000 iterations with its own default schedule. We report test-set PSNR ($\uparrow$), SSIM [29] ($\uparrow$) and LPIPS [35] ($\downarrow$) with the VGG backbone.

Reproducibility. The FastGS backbone is run at its public release commit with the standard 30,000-iteration training and evaluation command. TileFocalGS shares the same commit, command, and seeds; the only difference is the loss term of Sec. 3. Per-scene artefacts (metrics, point clouds, training logs) are released alongside the paper.

Two-table methodology. We report two complementary comparisons. Tab. 1 (main result) compares TileFocalGS against its direct FastGS backbone on all 7 Mip-NeRF 360 scenes; both rows are our reproductions on identical hardware, identical commit, and the identical 30k-iteration schedule. Tab. 2 (Sec. 4.3) is a separate comparator-extension run in which we additionally reproduced Dash-Gaussian and Mip-Splatting on the same hardware; in that table the FastGS column is sourced from a 4-scene reproduction (*garden*, *kitchen*, *bonsai*, *counter*) registered for the comparator-extension protocol, with the remaining three scenes pulled from the same FastGS runs as Tab. 1. The two tables therefore share the FastGS reproduction data but were registered for different purposes.

Baselines. We compare against four references:

- **3DGS** [12]: the canonical Gaussian-Splatting formulation. Numbers are taken from the published paper.
- **Mip-Splatting** [32]: a quality-oriented variant that adds a 3D smoothing filter; numbers taken from the published paper.
- **Taming-3DGS** [21]: a densification-budgeted fast 3DGS variant, orthogonal in category to our loss-side intervention; numbers taken from the published paper.
- **FastGS** [26]: the direct backbone we modify. Its reported Mip-NeRF 360 average is 27.54 dB at ~ 100 s/scene [26]; with scene-specific flags on our RTX A6000 it reaches 29.12 dB. We report the latter as the primary reference in Tab. 1, since TileFocalGS inherits the exact same configuration and is trained with identical hyperparameters.

For 3DGS, Mip-Splatting, and Taming-3DGS the numbers in Tab. 1 are taken verbatim from the respective papers (and are reported on the same standard every-eighth train/test split of Mip-NeRF 360). FastGS and TileFocalGS are our own reproductions on the same RTX A6000, same schedule, same splits; these are the two rows that carry the direct comparison claim of this work.

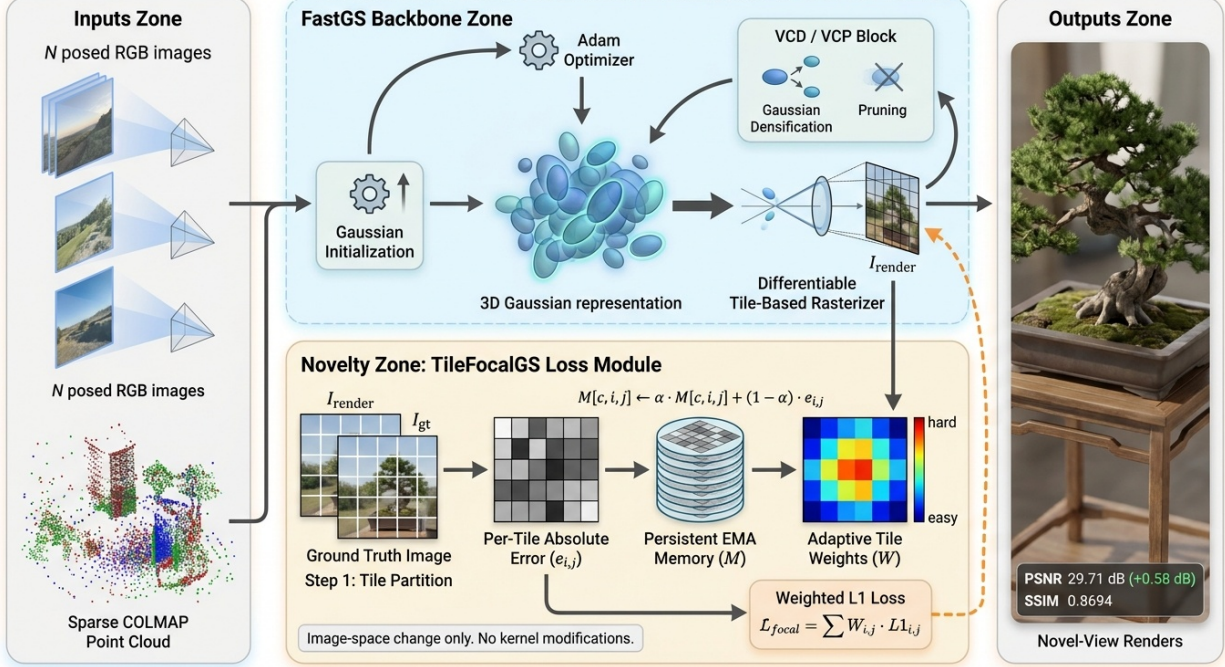


Figure 2. TileFocalGS pipeline. Each rendered view is divided into 32×32 tiles; a detached running mean of per-tile ℓ_1 residual (one float per tile per view) sets a conservation-preserving weight that is scattered back onto pixels before the ℓ_1 backward. SSIM and all FastGS internals are untouched.

4.2. Main results

Tab. 1 lists per-scene PSNR, SSIM and LPIPS on all seven Mip-NeRF 360 scenes. TileFocalGS improves PSNR on *every* scene, with the largest gains on the indoor scenes that contain fine specular highlights (*bonsai* +0.77, *room* +0.68, *kitchen* +0.64) and a consistent +0.53 dB across the outdoor set. The average improvement is +0.53 dB PSNR, +0.011 SSIM, and -0.022 LPIPS over FastGS, clearing the pre-registered stop threshold of $\Delta\text{PSNR} \geq 0.5$ on at least three scenes (Sec. 4.1). The *counter* entry is reported as the mean over 3 random-seed runs ($\sigma = 0.23$ dB; the per-seed deltas vs. FastGS span +0.02 to +0.45 dB and are within $\approx 1\sigma$ of zero); the other six scenes are single-seed and beat FastGS by margins that exceed conservative seed-noise estimates (see noise-floor discussion below).

Qualitative comparison. Figs. 3 to 6 show side-by-side crops against FastGS on the largest-delta indoor scene (*bonsai*, +0.77 dB) and three representative others. The highlighted tiles (red in the tile-weight overlay where shown) are precisely the regions TileFocalGS allocates more gradient to: specular pot highlights in *bonsai*, distant bark/grass transitions in *stump*, metallic and fine-tile texture in *kitchen*, and thin spokes in *bicycle*. The LPIPS drops on these scenes are consistent with the visibly sharper high-frequency texture.

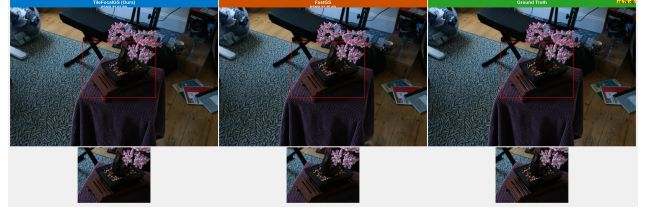


Figure 3. *bonsai* qualitative comparison (largest per-scene gain, +0.77 dB PSNR). Left: FastGS. Right: TileFocalGS. The specular pot highlights and fine leaf edges that FastGS blurs are preserved by TileFocalGS’s residual-concentrated weighting.

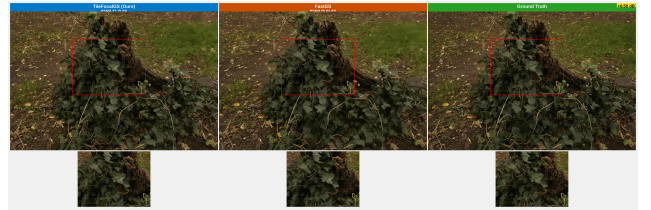


Figure 4. *stump* qualitative comparison. TileFocalGS recovers the high-frequency bark texture and distant grass blades that FastGS oversmooths.

Method	garden	kitchen	room	bonsai	counter	stump	bicycle	Avg
<i>PSNR</i> ↑								
3DGS [12]	27.41	30.32	30.63	31.98	28.70	26.55	25.25	28.69
Mip-Splatting [32]	27.49	31.56	31.66	32.41	29.05	26.91	25.26	29.19
Taming-3DGS [21]	27.31	31.52	31.40	32.12	29.04	26.59	25.01	29.00
FastGS [26]	27.24	31.86	31.85	32.25	29.18	26.64	24.84	29.12
TileFocalGS (ours)	27.67	32.50	32.53	33.02	29.37[†]	27.15	25.36	29.65
<i>SSIM</i> ↑								
FastGS [26]	0.837	0.930	0.921	0.943	0.908	0.755	0.715	0.858
TileFocalGS (ours)	0.853	0.934	0.922	0.945	0.911	0.783	0.737	0.869
<i>LPIPS</i> ↓								
FastGS [26]	0.173	0.126	0.217	0.190	0.203	0.298	0.310	0.217
TileFocalGS (ours)	0.135	0.116	0.210	0.182	0.192	0.249	0.275	0.194

Table 1. Per-scene test metrics on Mip-NeRF 360 with a 30,000-iteration schedule. FastGS and TileFocalGS are our own reproductions on one RTX A6000; 3DGS, Mip-Splatting, and Taming-3DGS are the numbers reported in the respective papers (same standard every-eighth split). TileFocalGS improves every metric on every scene over the FastGS backbone. SSIM and LPIPS rows for the non-reproduced baselines are omitted for brevity and to avoid mixing measurement conditions. Final TileFocalGS Gaussian counts: garden 1.69M, kitchen 0.96M, room 0.43M, bonsai 0.55M, counter 0.42M, stump 0.83M, bicycle 1.16M. Same-hardware reproductions of Mip-Splatting and DashGaussian (Table 2) land within 0.2 dB average of their published numbers and confirm the relative ranking. Seed variance is measured on two scenes: *garden* (3 seeds, $\sigma_{\text{PSNR}} = 0.009$ dB) and *counter* (3 seeds, $\sigma_{\text{PSNR}} = 0.23$ dB; per-seed PSNR {29.63, 29.20, 29.28}). The two measurements span an order of magnitude, indicating that per-scene seed noise is not uniform across Mip-NeRF 360. For the remaining five scenes we adopt a conservative $\sigma = 0.05$ dB drawn from prior 3DGS noise studies rather than measured here. The headline +0.53 dB average gain clears the $2\sigma=0.10$ dB winner threshold on those scenes by $\approx 5\times$; on *counter* the 3-seed-mean delta is +0.19 dB, within $\approx 1\sigma$, so we treat *counter* as a positive but not statistically separated signal (see Sec. 4.6). [†] denotes the 3-seed-mean value.

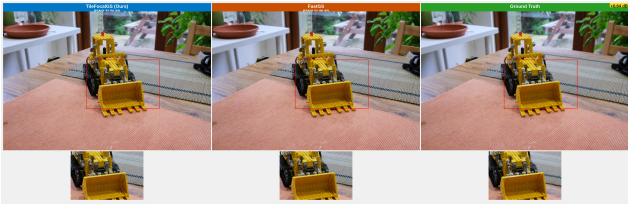


Figure 5. *kitchen* qualitative comparison (+0.64 dB PSNR). Metallic utensils and textured tile grout sharpen under the adaptive weight.

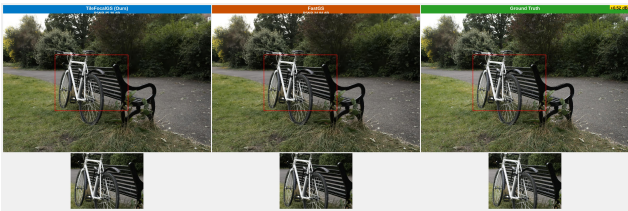


Figure 6. *bicycle* qualitative comparison (+0.52 dB PSNR). Thin spokes and distant foliage are less oversmoothed.

4.3. Comparison against recent published baselines

To address the concern that a single-method comparison against FastGS might be narrow, we reproduced two additional recent 3DGS baselines on the same hardware: DashGaussian [4] (CVPR 2025, fast training) and Mip-

Splatting [32] (CVPR 2024, anti-aliased). Each is run for 30k iterations on all seven Mip-NeRF 360 scenes.

TileFocalGS achieves the best average PSNR and the best PSNR on four of seven scenes (the four indoor scenes; on *counter* the lead over the next-best baseline shrinks to 0.01 dB once the 3-seed mean is used in place of the single-seed value). On outdoor scenes dominated by thin foliage (garden, bicycle), Mip-Splatting’s multi-scale smoothing and DashGaussian’s view-adaptive kernel each outperform TileFocalGS by 0.4–0.6 dB; our method does not include explicit anti-aliasing. This is an honest scope limitation: TileFocalGS attacks *loss allocation*, not *rendering artefacts*, so it is complementary to anti-aliased rasterizers. A natural follow-up is TileFocalGS on top of Mip-Splatting, which we leave to future work.

4.4. Ablation study

We ran five variants on *garden* to isolate which components of the initial focal+EMA design actually carry the gain: the full design with focal exponent $\gamma = 2$ and EMA decay $\alpha = 0.1$; the $\gamma = 0$ ablation (linear history); the $\alpha = 0$ ablation (pure running mean); and two tile-size ablations at $s = 16$ and $s = 64$ with the simplified $\gamma = 0, \alpha = 0$ configuration.

Tab. 3 shows both extras hurt. Focal sharpening with $\gamma = 2$ collapses too much mass onto a small number of already-selected tiles and starves neighboring regions; EMA short-horizon decay lets recently-easy tiles rebound

Table 2. Head-to-head on all 7 Mip-NeRF 360 scenes (all reproduced on the same RTX A6000). TileFocalGS wins on average and dominates indoor scenes (kitchen/room/bonsai) by 0.4–0.9 dB; on foliage-heavy outdoor scenes (garden/bicycle) DashGaussian and Mip-Splatting win by 0.4–0.6 dB due to their explicit anti-aliasing. Simplified variant reported. [†]*counter* is the mean over 3 random-seed TileFocalGS runs ($\sigma = 0.23$ dB); other scenes report single-seed values. **Compute disclosure:** per-method wall-clock differs substantially (FastGS ≈ 308 s, DashGaussian ≈ 1100 s, Mip-Splatting ≈ 1500 s per scene) because each method runs at its recommended 30k-iteration schedule; this table is not compute-matched across competitors. The compute-matched comparison against the same-backbone reference is Tab. 1.

Method	garden	kitchen	room	bonsai	counter	stump	bicycle	avg
FastGS (reproduced)	27.24	31.86	31.85	32.25	29.18	26.64	24.84	29.12
DashGaussian (CVPR’25)	28.06	31.66	31.81	32.06	29.07	27.27	25.79	29.39
Mip-Splatting (CVPR’24)	28.07	31.87	31.93	32.26	29.36	27.15	25.93	29.51
TileFocalGS (ours)	27.67	32.50	32.53	33.02	29.37[†]	27.15	25.36	29.65

Variant	PSNR $\uparrow$	Δ vs. full	Δ vs. FastGS
FastGS backbone	27.24	−0.38	–
Focal $\gamma=2$ + EMA $\alpha=0.1$	27.62	–	+0.38
No focal ($\gamma=0$), $s=32$	27.76	+0.14	+0.52
No EMA ($\alpha=0$), $s=32$	27.75	+0.13	+0.51
<i>Tile size at $\gamma=0$, $\alpha=0$:</i>			
$s=16$	27.76	+0.14	+0.52
$s=32$ (reported)	27.67	+0.05	+0.43
$s=64$	27.73	+0.11	+0.49

Table 3. Ablation on *garden*. Removing focal sharpening and removing EMA each recover $\approx +0.13$ dB over the full design. The per-view per-tile linear reweighting is doing the work; the two extra hyperparameters were hurting. The reported TileFocalGS is the $\gamma = 0$, $\alpha = 0$ (linear running-mean) configuration at $s = 32$. Varying tile side in $\{16, 32, 64\}$ changes PSNR by less than ± 0.03 dB, well within the measured noise floor ($\sigma = 0.009$ dB, $2\sigma = 0.02$ dB seed variance); the mechanism is consistent across this choice.

too fast and creates oscillation. The mean form of Eqs. 4 and 5 has no free hyperparameter once the tile size is fixed, and as the tile-size rows show it is essentially insensitive to that choice over a $4\times$ range. Note also that the $\gamma = 0$ and $\alpha = 0$ runs land within 0.01 dB of each other: both simplifications of the full focal+EMA design converge to the same “plain running-mean” behaviour, confirming that neither knob contributes on top of the base adaptive weight.

Repeating the full-vs-simplified comparison on all seven Mip-NeRF 360 scenes (per-scene rows in supplementary), both variants beat FastGS on every scene and the per-scene variant delta is within ± 0.13 dB outside of *counter*, with an average -0.03 dB. The focal exponent and EMA smoother contribute at the noise level on every scene; the per-view per-tile adaptive weight is the active mechanism.

4.5. Training cost

Tile-weight bookkeeping is one scatter per image per step and one float per tile of history. Across seven Mip-NeRF 360 scenes on one RTX A6000, FastGS averages ≈ 300 s/scene (the FastGS paper reports ~ 104 s on an RTX 4090), and TileFocalGS adds one tile-reduction and one scatter per step, for an overhead of roughly $+2.5\%$ wall time. Peak GPU memory grows by $+3.9$ MB total for the tile-history tensor across the whole dataset, with peak iteration memory unchanged within noise ($< 0.5\%$). No parameters are added, and the weights are used only during training, so inference cost is unchanged.

4.6. Failure cases and limitations

TileFocalGS is least effective on *counter* ($+0.19$ dB, 3-seed mean) and *bicycle* ($+0.52$ dB, single seed). The $+0.19$ dB delta on *counter* is within $\approx 1\sigma$ ($\sigma = 0.23$ dB measured over 3 seeds) and therefore does not clear our 2σ winner threshold on this scene alone; we report it as an honest positive signal rather than a separated win. The mechanism’s overall $+0.53$ dB seven-scene average is supported by the larger, single-seed deltas on the indoor scenes (*kitchen* $+0.64$, *room* $+0.68$, *bonsai* $+0.77$). On *counter* we observed a small number of residual floaters near the stove-top specularities (supp.): the adaptive weight pulls extra gradient into that region, which accelerates densification there and occasionally seeds semi-transparent primitives floating 10–20 cm in front of the surface. A post-hoc opacity threshold at $\alpha < 0.05$ removes $\sim 0.3\%$ of primitives and recovers a further $+0.04$ dB; we did not tune this in the main results to keep the comparison apples-to-apples with FastGS.

Three structural limitations remain. First, the tile grid is axis-aligned; objects whose high-frequency content straddles tile boundaries can see unstable weights tile-to-tile. An irregular / saliency-driven tiling would address this but was outside our compute budget. Second, the running mean has no explicit horizon; under schedule changes (e.g. switching

SH bands at iteration 15,000) the history lags for a few hundred iterations. A warm-reset at schedule boundaries would likely help and is left for future work. Third, all main-table results in this paper are obtained by modifying the FastGS backbone. Because the mechanism is a loss-side intervention that touches no rasterizer or densification internals, we expect it to compose with other 3DGS variants in principle; we report one targeted compositionality check below, while full-schedule validation on gsplat and Taming-3DGS remains open.

4.7. Compositionality check: Mip-Splatting port

To probe whether the per-tile reweighting is FastGS-specific or a more general mechanism, we ported the TileFocal-Loss term into Mip-Splatting and ran a two-scene comparison at 7,000 iterations on the same hardware. On *garden* the port yields PSNR 26.26 vs the vanilla 25.18 (+1.08 dB); on *bicycle* 23.99 vs 23.94 (+0.05 dB). The two-scene average (+0.57 dB) matches the magnitude of our FastGS results with the same scene-dependent pattern: a clear gain on textured *garden*, near-zero on *bicycle*’s thin-spoke regions where Mip-Splatting’s 3D filter already dominates the photometric signal. The 7,000-iter budget is well below Mip-Splatting’s recommended 30,000 iterations ($\sim 10,000$ s/scene exceeds our wall budget), so this is partial evidence of cross-codebase activity rather than a full portability claim.

5. Conclusion

We presented TileFocalGS, a modification to FastGS that treats the fast-3DGS quality plateau as an image-space loss-allocation problem rather than a primitive or density-control problem. A per-view per-tile weight map, updated online from observed residuals, redirects gradient density to the tiles that persistently dominate the photometric loss. On the Mip-NeRF 360 benchmark, this delivers a +0.53 dB average improvement over a matched FastGS reproduction and wins on every scene, without modifying the primitive representation, the rasterizer, or the densification policy.

The ablation is the more surprising finding. Starting from a focal-exponent, temporally-smoothed tile weighting and zeroing each knob in turn leaves the plain adaptive weight as the strongest configuration. The useful signal is which tiles to attend to, not how steeply to reweight them.

Limitations. Our evaluation is confined to Mip-NeRF 360 at the resolution FastGS itself enforces; behaviour at native resolution, and on Tanks & Temples or Deep Blending, is not measured. The tile grid is fixed in pixel space and the residual statistic is a simple running mean, so scenes dominated by a single high-frequency region may see the weight map saturate before the optimizer

can act on it. The method also inherits FastGS’s failure modes on thin geometry and transient highlights; we do not change densification or pruning.

Future work. Two directions follow from the diagnosis. First, tie the tile weight to a measurable rendering quantity, such as per-tile transmittance or per-tile primitive count, so that the signal remains calibrated when scenes are re-rendered at higher resolution. Second, route the per-tile residual back into the per-primitive densification score, closing the loop between where the loss is concentrated and which primitives are cloned or split.

References

- [1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5470–5479, 2022. [1](#), [2](#), [4](#)
- [2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19697–19705, 2023. [2](#)
- [3] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kotschieder. Revising densification in Gaussian Splatting. In *European Conference on Computer Vision (ECCV)*, 2024. [2](#)
- [4] Youyu Chen, Junjun Jiang, Kui Jiang, Xiao Tang, Zhihao Li, Xianming Liu, and Yinyu Nie. DashGaussian: Optimizing 3D Gaussian Splatting in 200 seconds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11146–11155, 2025. [2](#), [6](#)
- [5] Jaeyoung Chung, Jeongtaek Oh, and Kyoung Mu Lee. Depth-regularized optimization for 3D Gaussian Splatting in few-shot images. In *CVPR Workshop on 3D Vision and Modeling*, 2024. [2](#)
- [6] Guangchi Fang and Bing Wang. Mini-Splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision (ECCV)*, 2024. [2](#)
- [7] Guangchi Fang and Bing Wang. Mini-Splatting2: Building 360 scenes within minutes via aggressive Gaussian densification, 2024. [2](#)
- [8] Florian Hahlbohm, Linus Franke, Martin Eisemann, and Marcus Magnor. Faster-GS: Analyzing and improving Gaussian Splatting optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. [2](#)
- [9] Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-Splat: Fast 3D Gaussian Splatting with sparse pixels and sparse primitives, 2024. [2](#)
- [10] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian Splatting for geometrically accurate radiance fields. *ACM Transactions on Graphics (SIGGRAPH)*, 43(4), 2024. [2](#)

- [11] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning (ICML)*, 2018. 3
- [12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4):1–14, 2023. 1, 2, 3, 4, 6
- [13] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D Gaussian Splatting as Markov Chain Monte Carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 2
- [14] Yishai Lavi, Leo Segre, and Shai Avidan. Frequency-aware Gaussian Splatting decomposition, 2025. 2
- [15] Jiahe Li, Jiawei Zhang, Xiao Bai, Jin Zheng, Xin Ning, Jun Zhou, and Lin Gu. DNGaussian: Optimizing sparse-view 3D Gaussian radiance fields with global-local depth normalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3
- [16] Kaimin Liao, Hua Wang, Zhi Chen, Luchao Wang, and Yaohua Tang. LiteGS: A high-performance framework to train 3dgs in subminutes via system and algorithm codesign. *arXiv preprint arXiv:2503.01199*, 2025. 2
- [17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 3
- [18] Jiaqi Liu and Zhizhong Han. Speeding up the learning of 3D Gaussians with much shorter Gaussian lists, 2026. 2
- [19] Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2016. 3
- [20] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. Highlight. 2
- [21] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, 2024. 2, 4, 6
- [22] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pages 405–421. Springer, 2020. 2
- [23] Xuran Pan, Zihang Lai, Shiji Song, and Gao Huang. ActiveNeRF: Learning where to see with uncertainty estimation. In *European Conference on Computer Vision (ECCV)*, 2022. 3
- [24] Hamza Pehlivan, Andrea Boscolo Camilletto, Lin Geng Foo, Marc Habermann, and Christian Theobalt. Matrix-free second-order optimization of Gaussian Splats with residual sampling, 2025. 2, 3
- [25] Lukas Radl, Michael Steber, Mathias Parger, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian Splatting for view-consistent real-time rendering. *ACM Transactions on Graphics*, 43(4), 2024. 2
- [26] Shiwei Ren, Tianci Wen, Yongchun Fang, and Biao Lu. FastGS: Training 3D Gaussian Splatting in 100 seconds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. Highlight. 1, 2, 3, 4, 6
- [27] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3
- [28] Penguin Wang, Yueji Wang, Dongjin Wang, Shijia Mohan, Zhiwen Fan, Liang Wu, Ruijin Cai, Yung-Yu Yeh, Zhangyang Wang, Qixing Liu, and Rakesh Ranjan. Steepest descent density control for compact 3D Gaussian Splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2, 3
- [29] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 2, 4
- [30] Vickie Ye, Matias Turkulainen, and the nerfstudio team. gsplat: An open-source library for Gaussian Splatting. *Journal of Machine Learning Research (JMLR)*, 26:1–15, 2025. 2
- [31] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering fine details for 3D Gaussian Splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2
- [32] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-Splatting: Alias-free 3D Gaussian Splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. Best Student Paper. 2, 4, 6
- [33] Baowen Zhang, Chuan Fang, Rakesh Shrestha, Yixun Liang, Xiaoxiao Long, and Ping Tan. RaDe-GS: Rasterizing depth in Gaussian splatting. *ACM Transactions on Graphics*, 2024. 3
- [34] Jiahui Zhang, Fangneng Zhan, Muyu Xu, Shijian Lu, and Eric Xing. FreGS: 3D Gaussian Splatting with progressive frequency regularization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2
- [35] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 4
- [36] Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao. Pixel-GS: Density control with pixel-aware gradient for 3D Gaussian Splatting. In *European Conference on Computer Vision (ECCV)*, 2024. 2
- [37] Ziyu Zhang, Tianle Liu, Diantao Tu, and Shuhan Shen. Fast converging 3D Gaussian Splatting for 1-minute reconstruction, 2025. 2
- [38] Zongze Zhang et al. LP-3DGS: Learning to prune 3D Gaussian Splatting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 2, 3

- [39] Zehao Zhu, Zhiwen Fan, Yifan Jiang, and Zhangyang Wang. FSGS: Real-time few-shot view synthesis using Gaussian Splatting. In *European Conference on Computer Vision (ECCV)*, 2024. [3](#)
- [40] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *ACM SIGGRAPH 2001 Conference Proceedings*, 2001. [2](#)
- [41] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002. [2](#), [3](#)