ConeGS: Error-Guided Densification Using Pixel Cones for Improved Reconstruction with Fewer Primitives

Bartłomiej Baranowski Stefano Esposito Patricia Gschoßmann Anpei Chen[†] Andreas Geiger
University of Tübingen
baranowskibrt.github.io/conegs

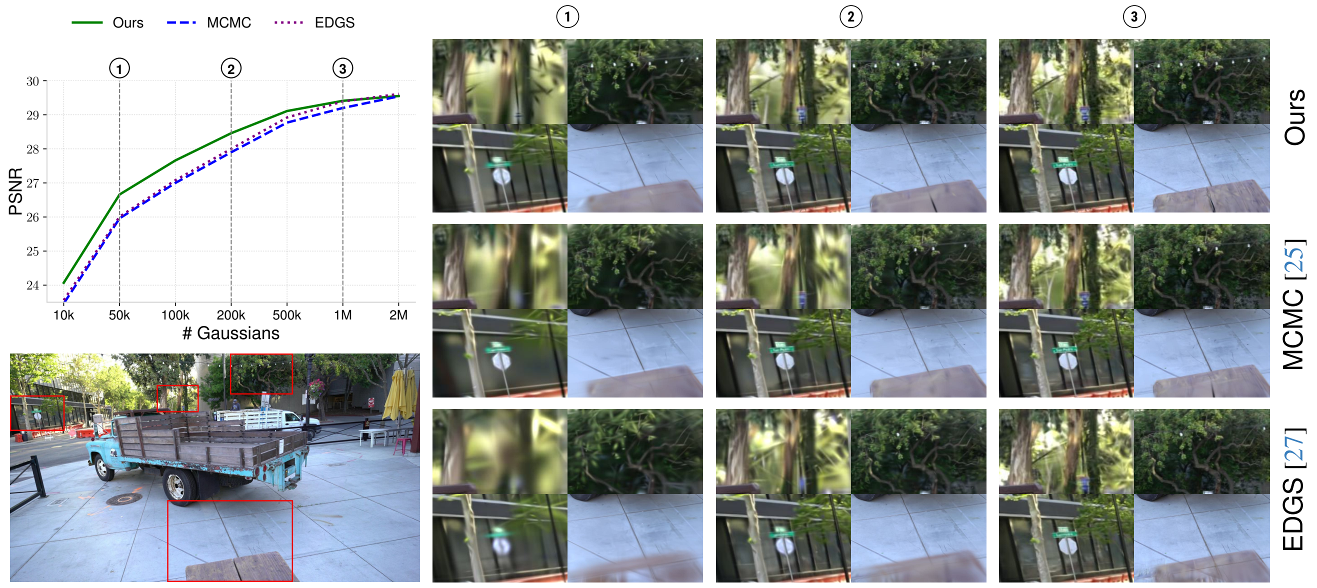


Figure 1. **ConeGS** replaces cloning-based densification with a novel method that generates pixel-cone-sized primitives in regions of high image-space error. By improving placement and removing reliance on existing scene structure thanks to a flexible iNGP-based exploration, it achieves higher reconstruction quality than baselines using the same number of primitives. Results are averaged over Mip-NeRF 360 [3] and OMMO [33] datasets, with a visual comparison on the `truck` scene from Tanks & Temples [26].

Abstract

3D Gaussian Splatting (3DGS) achieves state-of-the-art image quality and real-time performance in novel view synthesis but often suffers from a suboptimal spatial distribution of primitives. This issue stems from cloning-based densification, which propagates Gaussians along existing geometry, limiting exploration and requiring many primitives to adequately cover the scene. We present ConeGS, an image-space-informed densification framework that is independent of existing scene geometry state. ConeGS first creates a fast Instant Neural Graphics Primitives (iNGP) reconstruction as a geometric proxy to estimate per-pixel depth. During the subsequent 3DGS optimization, it identifies high-error pixels and inserts new Gaussians along the corresponding viewing cones at the predicted depth values, initializing their size according to the cone diameter. A pre-activation opacity penalty rapidly removes redundant Gaussians, while a primitive budgeting strategy controls the total number of primitives, either by a fixed budget or by adapting to scene complexity, ensuring high reconstruction quality. Experiments show that ConeGS consistently enhances reconstruction quality and rendering performance across Gaussian budgets, with especially strong gains under tight primitive constraints where efficient placement is crucial.

1. Introduction

Neural Radiance Fields (NeRF) [37] have significantly advanced novel view synthesis, achieving remarkable fidelity

[†]Corresponding author.

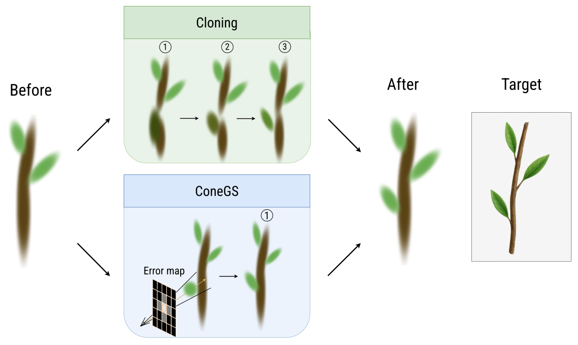


Figure 2. **Densification comparison.** Cloning-based methods are difficult to tune, and the resulting primitives may require many iterations to fit correctly into the scene. **ConeGS**, by contrast, places primitives precisely using the pixel viewing cone size, enabling faster scene integration without reliance on the existing geometry.

in scene reconstruction. However, representing scenes with neural networks makes NeRF slow to train and render, though it provides smooth parameterization and flexibility to handle changes in scene structure. Recently, 3D Gaussian Splatting (3DGS) [24] has gained attention as a faster, more practical alternative to NeRF, explicitly modeling scenes with sets of 3D Gaussians to achieve interactive rendering speeds while maintaining competitive visual fidelity. However, 3DGS increases expressiveness through cloning and splitting, which offer limited exploration, rely on hard-to-define densification rules, and generate many unnecessary primitives. As a result, primitives often accumulate in suboptimal regions, leaving large parts of the scene underrepresented or mispredicted.

To address these issues, we propose **ConeGS**, which replaces cloning-based densification with a novel strategy that targets pixels exhibiting high photometric error. By sampling these pixels and using depth estimates from a fast Instant Neural Graphics Primitives (iNGP) [39] reconstruction, new Gaussians are placed precisely in regions where the current representation is insufficient. This targeted placement increases expressiveness in areas requiring higher primitive density, improving reconstruction quality while avoiding redundant primitives. To determine the size of new Gaussians, we draw inspiration from Mip-NeRF [2]. During densification, each Gaussian is initialized according to the size of the viewing cone of the pixel from which it is generated at the specified depth. Their initial size is thus defined directly by their image-space coverage, eliminating the need for local size analysis or adjustments to reconstructed regions. Figure 2 illustrates the effectiveness of the proposed approach. Combined with a pre-activation opacity penalty that quickly removes redundant Gaussians, this enables scene representation with fewer primitives while preserving high reconstruction quality. We further incorporate two primitive budgeting strategies to regulate the total number of primitives, either through a fixed budget or by adapting to scene complexity. ConeGS outperforms baseline methods across diverse datasets and a wide range of primitive budgets. The advantage is most pronounced under tight primitive budgets. At higher budgets, it matches the quality of cloning-based methods, where efficient primitive placement is less critical, while still rendering faster than the baselines. In summary, our contributions are:

- A densification strategy that places new Gaussians in regions of high photometric error in image space, guided by depth estimates from an iNGP-based geometric proxy.
- An approach that determines the size of new Gaussians from the viewing cones of the pixels from which they are generated.
- An improved opacity penalty that promptly removes low-opacity Gaussians, combined with a budgeting strategy that balances scene complexity and primitive count.

Finally, our method is also compatible with other 3DGS improvements, making it straightforward to integrate with existing approaches for greater efficiency, or with methods where cloning strategies are ambiguous or hard to formalize [20, 34, 49].

2. Related work

Neural Radiance Fields: NeRFs [37] represent scenes as continuous volumetric radiance fields, enabling high-quality novel view synthesis. This is achieved by parameterizing the scene with a neural network (typically an MLP), whose weights encode the scene globally. Despite producing photorealistic results, these methods rely on costly volumetric rendering and remain computationally inefficient. Extensions such as Mip-NeRF [2] and Mip-NeRF360 [3] reduce aliasing via conical frustum integration, while Zip-NeRF [4] improves view consistency with hierarchical sampling and multi-scale supervision. Hybrid approaches [7, 45, 46, 56] mitigate this by combining explicit data structures with compact neural representations, enabling faster optimization and real-time rendering. Instant Neural Graphics Primitives (iNGP) [39] further accelerate training through multi-resolution hash-grid encoding and shallow MLPs.

Primitive-based Differentiable Rendering: 3D Gaussian Splatting (3DGS) [24] has emerged as an efficient alternative to Neural Radiance Fields (NeRF) [37]. Rather than modeling the scene as a global volume, 3DGS represents it with local explicit 3D Gaussians and uses differentiable rasterization, resulting in significantly faster rendering. Its balance of fidelity and efficiency has attracted significant attention and spurred a wide range of follow-up research. Prior works have focused on tackling anti-aliasing [53, 57], reconstructing dynamic scenes [52, 54], enabling generative content creation [48, 66], reducing rendering artifacts [43], substituting alpha composition with volumetric rendering [35, 47], extracting geometry [16, 21, 58], level-of-detail reconstruction [44], frequency-based regularization [59, 60], and introducing new primitives or kernel functions [18, 20, 31, 49]. Recent efforts have also targeted reducing computational and memory costs, often through feature quantization or code-book encoding [10, 15, 34, 41], or scene simplification [63]. [11] reduces computation by lowering the number of primitives through an aggressive densification and pruning strategy, while [11, 12, 64] insert new Gaussians at the currently estimated depths using re-initialization. Unlike our method, this approach overwrites existing structures instead of adding new points, and further depends on the scene already being well reconstructed. [25] improves primitive distribution and exploration by incorporating positional errors and applying penalties to opacity and scaling. Closely related to our approach, several works focus on improving densification to reduce redundancy and better capture fine details. Strategies include refining cloning heuristics [5, 22, 25], per-Gaussian property- or saliency-based cues [36], geometry- and volume-aware criteria [1, 23, 65], addressing gradient collision [55], perceptual sensitivity [64], learnable schemes [32, 40], and based on Gaussian Processes [17]. Some works target densification in challenging settings [38], filling holes in the representation [9, 29], though typically adding only a few primitives. PixelSplat [6] models dense probability distributions for more robust Gaussian placement, influencing later approaches [8]. Recent work [27] suggests that densification may be unnecessary for high-quality reconstruction given strong initialization. Like our method, they start by estimating scene geometry, but rely only on correspondences from a pretrained dense matching network, without enhancing densification, and at higher GPU memory cost than our approach. Concurrent work [62], employs Gaussians with spatially varying texture colors, improving fine-detail reconstruction and reducing the number of primitives needed. Other methods use neural radiance fields for depth supervision [14, 28] or point cloud extraction [14, 42, 50] to initialize a scene, but not to improve densification directly. Concurrent work [13] applies NeRF for initialization and limited densification, constrained by existing Gaussian locations, and does not explore varying Gaussian sizes, which we find beneficial for reconstruction quality.

3. Preliminaries

3D Gaussian Splatting: 3DGS [24] represents a scene as an unordered set of 3D Gaussian primitives $\{\mathcal{G}_i | i = 1, \dots, M\}$. Each primitive $\mathcal{G}_i = (\mathbf{p}_i, \mathbf{s}_i, \mathbf{R}_i, o_i, \mathbf{c}_i)$ is defined by its position $\mathbf{p}_i \in \mathbb{R}^3$, scaling vector $\mathbf{s}_i \in \mathbb{R}^3$, rotation matrix $\mathbf{R}_i \in \mathbb{R}^{3\times3}$, opacity $o_i \in \mathbb{R}$, and view-dependent color $\mathbf{c}_i \in \mathbb{R}^3$. The color $\mathbf{c}_i$ is represented by spherical harmonics (SH) coefficients $\mathbf{k}_i \in \mathbb{R}^{3L}$, where L is the number of coefficients determined by the chosen SH order. The 3D covariance matrix is given by $\mathbf{\Sigma}_i = \mathbf{R}_i\mathbf{S}_i\mathbf{S}_i^T\mathbf{R}_i^T$, where $\mathbf{S}_i = \mathrm{diag}(\mathbf{s}_i)$ is the scaling matrix. The color $\hat{C}$ of a pixel is computed by α-blending over a set of N Gaussians, sorted by depth, whose projections overlap the pixel:

$$\hat{C} = \sum\nolimits_{i\in N} \mathbf{c}_i\alpha_i \prod\nolimits_{j=1}^{i-1}(1-\alpha_j), \tag{1}$$

$$\alpha_i = o_i K(\mathbf{p}_c, \boldsymbol{\mu}_i^{2D}, \mathbf{\Sigma}_i^{2D}), \tag{2}$$

where α_i is the blending weight of the i-th Gaussian, $\mathbf{p}_c$ is the pixel center in image coordinates, $\boldsymbol{\mu}_i^{2D}$ and $\mathbf{\Sigma}_i^{2D}$ are the 2D projected mean and covariance of $\mathcal{G}_i$, and $K(\cdot)$ is a Gaussian filter response in screen space. The exact form of K depends on the chosen filter [24, 57, 67]. Gaussians are traditionally initialized from an SfM point cloud, with each component of $\mathbf{s}_i$ set equal to the mean Euclidean distance to the three nearest neighbors $\mathcal{N}_3(i)$ of Gaussian i:

$$\mathbf{s}_i = (s_i, s_i, s_i), \quad s_i = \frac{1}{3}\sum_{k\in\mathcal{N}_3(i)} \|\mathbf{p}_k - \mathbf{p}_i\| \quad . \tag{3}$$

During training, the Gaussian parameters are optimized with the loss:

$$\mathcal{L}_{\text{GS}} = (1-\lambda)\,\text{MAE}(I, I^*) + \lambda\,\mathcal{L}_{D\text{-}SSIM}, \tag{4}$$

where $\lambda = 0.2$, MAE is the mean absolute error between the rendered image I and the ground-truth image I^*, and $\mathcal{L}_{D\text{-}SSIM} = 1 - \text{SSIM}(I, I^*)$ [51].

Neural Radiance Fields: NeRFs [37] model a scene as a continuous 3D field that maps a 3D location along a camera ray and the viewing direction of the corresponding pixel to a density $\sigma \in \mathbb{R}$ and color $\mathbf{c} \in \mathbb{R}^3$. A camera ray is parameterized as $\mathbf{r}(t) = \mathbf{p}_{\text{cam}} + t\mathbf{d}$, where $\mathbf{p}_{\text{cam}}$ is the camera position and $\mathbf{d}$ is a unit direction vector pointing toward the center of a pixel. Each ray is discretized into N intervals defined by distances $\{t_i, t_{i+1}\}_{i=1}^N$. For each sample position $\mathbf{r}(t_i)$ along the ray, the NeRF is queried to predict the sample's color $\mathbf{c}_i$ and density σ_i. Using volumetric rendering [37], the corresponding pixel color is approximated as:

$$\hat{C} = \sum\nolimits_{i=1}^{N} \mathbf{c}_i\alpha_i \prod\nolimits_{j=1}^{i-1}(1-\alpha_j), \tag{5}$$

$$\alpha_i = 1 - \exp(-\sigma_i\delta_i) \quad \text{with} \ \ \delta_i = t_{i+1} - t_i. \tag{6}$$

Here, α_i is the opacity of the i-th sample, δ_i is the length of its ray segment, and the product term represents the transmittance $\tau_i = \prod_{j=1}^{i-1}(1-\alpha_j)$.

Sampling only a single ray per pixel can lead to blur and aliasing. Mip-NeRF [2] addresses this by replacing the ray with a cone that models the pixel footprint, i.e. the 3D volume a pixel covers in world space. The cone is divided into frustums, and integration is performed over these volumes rather than along a 1D line. The cone's radius $r_{\text{cone}}(t)$ defines the cross-section of the pixel cone at distance t and is computed from the directions of rays passing through the pixel and its neighbors:

$$r_{\text{cone}}(t) = t\,\frac{\|\mathbf{d}_x - \mathbf{d}\| + \|\mathbf{d}_y - \mathbf{d}\|}{2}, \tag{7}$$

where $\mathbf{d}$ is the direction of the ray through the center of the pixel, and $\mathbf{d}_x, \mathbf{d}_y$ are the directions of rays through the neighboring pixels in the x and y directions, respectively.

4. Method

This section outlines our ray-based densification approach for 3DGS. First, we explain how we use an iNGP model as a geometric proxy to initialize the 3D Gaussian scene (Section 4.1). Next, we detail our ray-based densification strategy, which uses the iNGP to place pixel-cone-sized Gaussians in high-error regions, along with associated optimization changes (Section 4.2). Finally, we provide additional implementation details in Section 4.3. An overview of the complete pipeline is shown in Figure 3.

4.1. Initialization

We use a trained iNGP model [39] as a geometric proxy to initialize the 3DGS scene and guide densification. Trained briefly on input images, it provides accurate depth estimates, that position both the initial Gaussian primitives and those added later during densification, with minimal impact on training time. Additionally, the depths can be evaluated on the fly during optimization, reducing both memory usage and computation compared to precomputing all depth maps. We initialize the scene with $\mathcal{P}_{\text{init}}$ Gaussians, set to one million as in [42], or fewer if a smaller budget is specified (see Section 4.2). To construct this set, we uniformly sample $\mathcal{P}_{\text{init}}$ image-pixel pairs (I, u, v) from the training set pixel domain $\mathcal{I}_{\text{train}}$. Each sampled image-pixel pair defines exactly one Gaussian in the initialized scene. For each sample, we define its associated camera ray

$$\mathbf{r}_I(u, v, t) = \mathbf{p}_I + t\,\mathbf{d}_I(u, v), \tag{8}$$

where I is an image index, (u, v) are pixel coordinates, $\mathbf{p}_I \in \mathbb{R}^3$ is the camera center, and $\mathbf{d}_I(u, v) \in \mathbb{R}^3$ is the normalized ray direction. We query the iNGP along this ray to obtain discrete transmittance values $\{\tau_k\}$, from which the median depth t_{med} is computed as:

$$t_{\text{med}} = t_k \quad \text{where} \quad \tau_{k-1} > 0.5 \geq \tau_k. \tag{9}$$

The center of the j-th Gaussian primitive is then set to:

$$\mathbf{p}_j = \mathbf{p}_I + t_{\text{med}}\,\mathbf{d}_I(u, v), \tag{10}$$

yielding the set of initial centers $\{\mathbf{p}_j\}_{j \in \mathcal{I}_{\text{sample}}}$ with $\mathcal{I}_{\text{sample}} \subset \mathcal{I}_{\text{train}}$. The scale $\mathbf{s}_j$ is initialized isotropically using the average distance to the three nearest neighbors $\mathcal{N}_3(j)$, following Eq. (3). The rotation is set to identity $\mathbf{R}_j = \mathbf{I}$, the opacity to $o_j = 0.1$, and the SH coefficients to:

$$\mathbf{k}_j = (\mathbf{k}_{1:3,j}, \mathbf{k}_{4:L,j}), \quad \mathbf{k}_{1:3,j} = \mathbf{c}_{j,0}, \quad \mathbf{k}_{4:L,j} = \mathbf{0}, \tag{11}$$

where $\mathbf{c}_{j,0}$ is the RGB color for the sampled pixel (I, u, v) rendered with iNGP using a zeroed-out view direction. Although our densification strategy can achieve high-quality reconstructions without scene initialization, we retain this step to ensure consistently strong performance across all metrics (see Section 5.2).

4.2. Optimization

We fully replace the standard 3DGS cloning-based densification with an error-guided strategy that adapts the iNGP-based ray-depth rendering procedure from Section 4.1 to position new Gaussian primitives. Below, we outline the sampling, scaling, budgeting, and pruning stages of our densification pipeline.

Error-Weighted Gaussian Densification: To limit the number of primitives while targeting poorly reconstructed regions, we add new Gaussians at pixels with high photometric error. At iteration j, we render an image I_j and compute the per-pixel absolute error (L_1 loss) $E(\mathbf{p}) = |I_j(\mathbf{p}) - I^*(\mathbf{p})|$ with respect to the ground-truth image I^*. We then sample N_{sample} pixels without replacement according to a multinomial distribution $\mathcal{M}$ with probabilities proportional to the normalized error map:

$$\{\mathbf{p}_s\}_{s=1}^{N_{\text{sample}}} \sim \mathcal{M}\left(N_{\text{sample}}, \frac{E(\mathbf{p})}{\sum_{\mathbf{p}' \in I_j} E(\mathbf{p}')}\right), \tag{12}$$

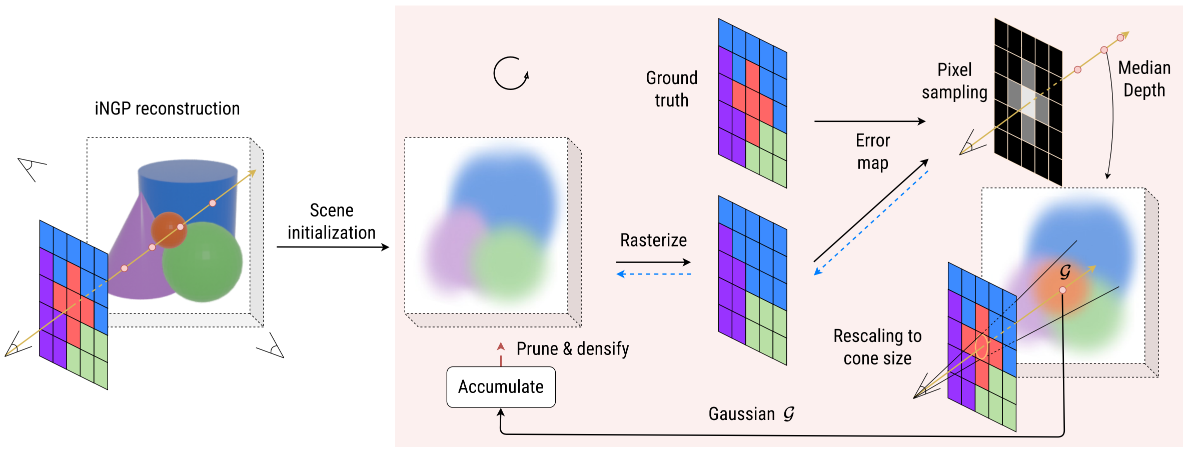
While L_1 loss does not always indicate possible improvements and can also arise from noise or difficult-to-optimize reflections, we found it to be a reliable indication of lacking expressiveness, especially at low primitive budgets. For each sampled pixel $\mathbf{p}_s$, a new Gaussian $\mathcal{G}_s$ is created, with its center placed along the corresponding ray at the median depth $t_{\text{med},s}$ given by the iNGP. Newly spawned Gaussians are appended to an accumulation set:

$$\mathcal{G}_{\text{accum}} \leftarrow \mathcal{G}_{\text{accum}} \cup \{\mathcal{G}_s\}_{s=1}^{N_{\text{sample}}}. \tag{13}$$

Every 100 iterations, low-opacity Gaussians in the scene $\mathcal{G}_{\text{scene}}$ are pruned, and Gaussians in $\mathcal{G}_{\text{accum}}$ are merged into the scene:

$$\mathcal{G}_{\text{scene}} \leftarrow \mathcal{G}_{\text{scene}} \cup \mathcal{G}_{\text{accum}}, \quad \mathcal{G}_{\text{accum}} \leftarrow \emptyset. \tag{14}$$

The newly inserted Gaussians are then jointly optimized with existing primitives. Unlike cloning-based methods,



(a) Initialization (Section 4.1) (b) Optimization (Section 4.2)

Figure 3. **Overview of the ConeGS pipeline.** (*a*) First, an iNGP reconstruction is obtained to serve as a geometric proxy for object surfaces, guiding the placement of Gaussians both during scene initialization and throughout the 3DGS optimization process. (*b*) During 3DGS optimization, ConeGS performs error-guided densification by sampling a subset of pixels with high L_1 error. For each sampled pixel, a new Gaussian $\mathcal{G}$ is created along the pixel's viewing cone at the depth estimated by iNGP and scaled to match the cone's size. New Gaussians are accumulated and, every 100 iterations, inserted into the scene after pruning those with low opacity. Blue arrows indicate gradient updates to Gaussian parameters, and the red arrow marks scene updates.

which constrain new primitives to the vicinity of existing ones and thus hinder exploration of unseen regions, our approach places Gaussians directly in high-error areas, enabling effective scene coverage even far from existing geometry. Moreover, we preserve the integrity of well-reconstructed areas, since new Gaussians are added on top of the existing structure rather than created through splitting or cloning. Although occasional iNGP depth inaccuracies may introduce misplaced Gaussians, diverse viewpoint coverage ensures that inconsistent ones are quickly corrected or pruned, while multiview-consistent ones are retained. The full densification pipeline is shown in Figure 4.

Pixel-Footprint-Aligned Scaling: Selecting an appropriate scale for newly added Gaussians during densification is crucial. If primitives are too large, they may obscure fine details and be pruned prematurely, whereas overly small ones contribute little to the rendered image, yielding weak gradients and slowing convergence. Although k-NN–based scaling is effective for initialization, recomputing nearest-neighbor distances at every densification step is computationally expensive and sensitive to outliers. Large distances can produce inflated scales, causing new Gaussians to overlap well-reconstructed regions and hinder further optimization (see Section 5.2). To avoid these issues, we set the initial scale of each newly added Gaussian directly from the pixel footprint at the median depth $t_{\text{med},i}$ along its corresponding camera ray, as defined in Eq. 7:

$$\mathbf{s}_i = \lambda_{\text{scale}}\, r_{\text{cone}}(t_{\text{med},i})\,(1,1,1), \tag{15}$$

where $\lambda_{\text{scale}} = 2$ converts the cone radius $r_{\text{cone}}(t_{\text{med},i})$ to the diameter of its cross-section. This ensures that, from the spawning viewpoint, the Gaussian's projection onto the image plane approximately matches the pixel width, independent of scene depth. The assigned scale is only an initial value, with subsequent optimization steps jointly updating all primitives to allow newly added Gaussians to adjust to the existing scene. Our pixel-aligned, depth-aware scaling provides three key benefits: (1) it is independent of the current primitive distribution, avoiding the structural biases of cloning-based methods that replicate and reinforce local geometry, (2) pixel size allows Gaussians to contribute to optimization immediately and efficiently fit fine details while minimizing overlap with existing structure, and (3) its isotropic shape promotes stable multi-view integration, without shapes that the cloned Gaussians inherit.

Primitive Budgeting: We consider two budgeting strategies for controlling the number of Gaussians in the scene. The first enforces a hard upper bound, as in [25], ensuring that densification never exceeds the prescribed budget. This constraint regulates memory and computation while preventing uncontrolled growth of the primitive set. At each densification step, we set the number of sampled pixels N_{sample} so that newly added Gaussians replace those pruned, avoiding excess primitives that would otherwise be discarded under the budget. This is computed as:

$$N_{\text{sample}} = \frac{\max(0.2N_{\text{GS}},\ 1.2N_{\text{last}})}{100}, \tag{16}$$

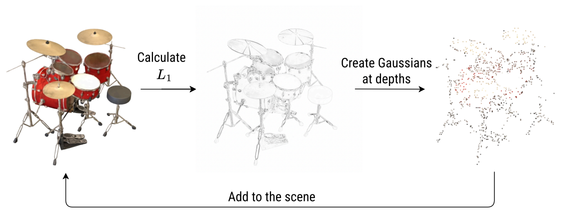


Figure 4. **Densification overview.** Illustration of the proposed error-guided strategy. We render an image with 3DGS, compute the per-pixel L_1 error, sample pixels proportionally to their error magnitude, and place new Gaussians at the iNGP-predicted depth along the corresponding viewing rays.

where N_{GS} is the current total number of Gaussians, and N_{last} is the number of primitives inserted in the previous densification step, and the division by 100 reflects the densification interval. This formulation keeps N_{GS} close to the budget limit even under aggressive pruning, maintaining consistent scene coverage throughout optimization.

The second strategy adapts the number of primitives to the scene's complexity, enabling controlled growth without imposing a fixed upper bound:

$$N_{\text{sample}} = \frac{\beta N_{\text{GS}}}{100}. \tag{17}$$

Here, β controls the growth rate of the primitive set. Smaller values balance the number of Gaussians added with those pruned, maintaining a relatively stable primitive count, whereas larger values yield higher primitive counts, increasing geometric detail at the cost of memory and computing power. With scene initialization at 1M primitives and the application of the opacity penalty, we balance Gaussians added and pruned, unlike [5], which requires a predefined upper limit on primitives.

Opacity-Regularized Pruning: Following [24, 25], Gaussians with opacity below 0.005 are pruned every 100 iterations to remove primitives with negligible contribution to the rendered image. Earlier work has promoted sparsity through different strategies: periodically resetting opacities [24], which can destabilize training [5], reducing opacities by a constant amount after each densification [5], or introducing a post-activation opacity penalty $\mathcal{L}_{\mathbf{o}}^{\text{post}} = \|\sigma(\mathbf{o}_{\text{pre}})\|_1$ [25], where $\mathbf{o}_{\text{pre}}$ denotes the opacity logits before the sigmoid and σ is the sigmoid function. This applies the strongest constraint around 0.5 and only a weak penalty near the pruning threshold. In contrast, we employ a pre-activation opacity penalty $\mathcal{L}_{\mathbf{o}}^{\text{pre}} = \|\mathbf{o}_{\text{pre}}\|_1$. It provides a steady constraint across the full opacity range, including very low values, gradually reducing under-contributing primitives. The penalty acts throughout training, and our densification strategy can freely add new primitives, allowing any structure lost through pruning to be recovered more easily than with cloning-based approaches. In all experiments, this loss is scaled by $\lambda_{\mathbf{o}} = 0.0002$.

4.3. Implementation Details

For the iNGP model, we use the proposal-based implementation from NerfAcc [30], trained for 20k iterations with the original setup and architecture. Gaussian optimization runs for 30k iterations, with our densification active for the first 25k. Unlike 3DGS, all SH components are optimized from the start, enabled by the stable initialization that removes the need for gradual SH introduction.

5. Evaluation

Dataset and Metrics: We evaluate our method on publicly available scenes from Mip-NeRF360 [3] and OMMO dataset [33], with `01` scene from OMMO resized to have 1600 pixels width. Following [24, 25], we also include the `train` and `truck` scene from Tanks & Temples [26], as well as `Dr Johnson` and `playroom` from the Deep-Blending [19] dataset. We report PSNR, SSIM [51], and LPIPS [61], with rendering speeds averaged across the full test set. All FPS measurements were recorded on an NVIDIA RTX 2080 Ti, whereas training speeds are reported on an NVIDIA A100, since EDGS requires more memory. These training speeds do not include later-added speed improvements [36].

Baselines: We primarily compare our method against 3DGS [24], it's extension with iNGP point cloud initialization [14], MCMC [25] using different initialization types (random, SfM, iNGP point clouds), as well as, GaussianPro [9], Perceptual-GS [64], EDGS (with densification) [27], with the densification stopped for all of them if the primitive budget is reached. If the number of primitives at initialization would be higher than the specified budget, the number of primitives is sampled uniformly to fit below it. In the random initialization settings we follow the process described in MCMC [25]. We additionally test on Mini-Splatting2 [11] by matching their final number of primitives instead of a specific budget, due to their method relying on generating a high number of initial Gaussians.

5.1. Results

We observe improvements over the baselines across a wide range of specified budgets in Table 1, with plots comparing the most important methods on the budget and no-budget scenario in Figure 6. For a lower limit on Gaussians, we outperform the benchmarks across all datasets and metrics, while providing a competitive reconstruction quality compared to the best performing baselines on the high budget scenarios. In Table 2 we additionally show that even on a high number of primitives of 1M and including the iNGP training, our method provides competitive speed to other

	Mip-NeRF360 [3]			OMMO [33]			Tanks & Temples [26]			DeepBlending [19]		
	PSNR ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Number of Gaussians limited to 100k												
3DGS [24] (SfM init.)	23.61	0.693	0.413	26.45	0.820	0.296	22.38	0.774	0.333	24.65	0.827	0.412
Foroutan et al. [14]†	26.64	0.781	0.318	26.89	0.829	0.276	22.48	0.766	0.341	25.31	0.822	0.418
MCMC [25] (rand. init.)	25.72	0.730	0.369	25.92	0.808	0.313	21.45	0.750	0.365	27.94	0.859	0.369
MCMC [25] (SfM init.)	27.06	0.800	0.303	27.01	0.841	0.266	22.50	0.780	0.332	28.94	0.876	0.333
MCMC [25] (iNGP init.)	27.35	0.797	0.299	26.95	0.837	0.265	22.69	0.775	0.326	29.02	0.872	0.337
GaussianPro [9]	25.57	0.766	0.338	26.14	0.822	0.289	20.59	0.757	0.348	28.15	0.870	0.342
Perceptual-GS [64]	25.66	0.774	0.320	26.13	0.819	0.292	20.76	0.759	0.347	28.32	0.874	0.338
EDGS [27]	27.09	0.798	0.296	26.99	0.838	0.261	22.32	0.777	0.324	28.43	0.872	0.337
Ours	27.74	0.809	0.285	27.59	0.852	0.243	23.12	0.791	0.310	29.44	0.880	0.328
Number of Gaussians limited to 500k												
3DGS [24] (SfM init.)	28.22	0.821	0.260	28.96	0.883	0.196	23.54	0.816	0.265	29.25	0.882	0.302
Foroutan et al. [14]†	28.88	0.862	0.204	28.80	0.884	0.185	23.52	0.816	0.257	29.61	0.886	0.297
MCMC [25] (rand. init.)	28.38	0.844	0.237	28.23	0.874	0.212	22.96	0.808	0.279	28.84	0.875	0.315
MCMC [25] (SfM init.)	28.82	0.861	0.214	28.72	0.885	0.194	23.68	0.825	0.259	29.44	0.887	0.308
MCMC [25] (iNGP init.)	29.02	0.867	0.198	28.75	0.885	0.186	23.57	0.823	0.243	29.61	0.885	0.288
GaussianPro [9]	28.02	0.827	0.253	28.32	0.876	0.206	22.31	0.802	0.286	29.33	0.886	0.301
Perceptual-GS [64]	28.66	0.856	0.211	28.42	0.876	0.203	22.77	0.813	0.267	29.44	0.888	0.296
EDGS [27]	28.82	0.865	0.193	29.01	0.893	0.168	23.47	0.831	0.229	29.33	0.889	0.286
Ours	29.08	0.870	0.190	29.14	0.892	0.170	23.69	0.829	0.229	29.86	0.891	0.285
Number of Gaussians bounded by Mini-Splatting2 [11]												
Mini-Splatting2 [11]	28.89	0.875	0.183	28.06	0.875	0.198	22.79	0.823	0.239	29.99	0.898	0.279
Ours	29.26	0.875	0.179	28.90	0.887	0.179	23.66	0.829	0.231	29.82	0.891	0.280

† Original code was not publicly available. Our implementation uses iNGP initialization and does not include the additional depth-based loss.

Table 1. **Quantitative results** with a Gaussian number limit of 100k and 500k. Mini-Splatting2 [11] does not support constraining the number of Gaussians during reconstruction, so we match its Gaussian count. We highlight the best, second best and third best results among methods with the same Gaussian counts. Per-scene metrics for selected methods are provided in the supplementary material.

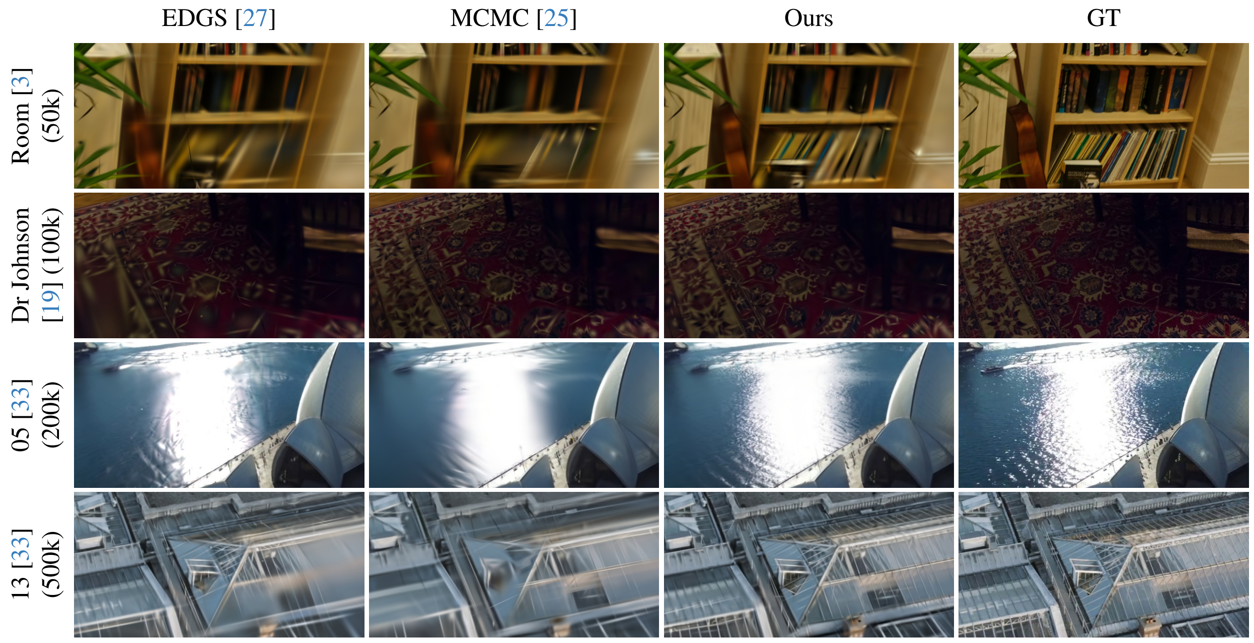


Figure 5. **Qualitative results** comparing our method with MCMC [25] (with SfM point cloud initialization) and EDGS [27] on the Mip-NeRF 360 [3], OMMO [33], and DeepBlending [19] datasets, with varying Gaussian budgets (given in parentheses).

methods. The qualitative results in Figure 5 show significant improvement using a wide range of primitive budgets, demonstrating that for the same limit of primitives our method is able to produce a much better reconstruction, especially in areas that are challenging to properly capture on a low budget, such as isolated or high frequency structures. We provide additional qualitative and quantitative results, along with further scene analysis, in the appendix.

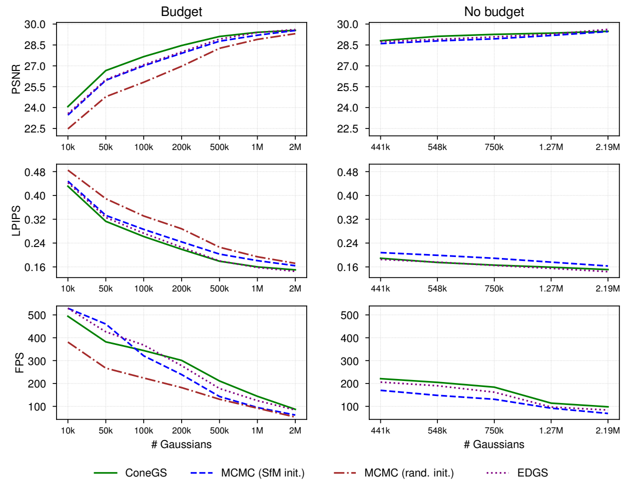


Figure 6. **PSNR, LPIPS and FPS plots** for a specified primitive budget (left) and without a budget (right), where the number of primitives corresponds to the chosen β values. Numerical results are provided in the appendix. Values are averaged across Mip-NeRF360 [3] and OMMO [33].

	Ours 10k iters	Ours 20k iters	Ours 40k iters	EDGS [27]	3DGS [24] (SfM init.)	MCMC [25] (rand. init.)	MCMC [25] (SfM init.)
PSNR ↑	29.33	29.37	29.37	29.18	28.71	28.98	29.23
SSIM ↑	0.877	0.88	0.881	0.877	0.846	0.865	0.875
LPIPS ↓	0.171	0.168	0.166	0.168	0.222	0.204	0.190
FPS ↑	134	137	137	131	112	92	95
3DGS time ↓	20.7	20.5	20.5	19.4	22.6	26.3	25.1
Init. time ↓	1.6	3.1	6.1	2.3	-	-	-
Overall time ↓	22.3	23.6	26.6	21.7	22.6	26.3	25.1

Table 2. **Quantitative results** showing overall scene reconstruction timings for 3DGS [24], MCMC [25] and EDGS [27] methods, as well as our method with differing iNGP reconstruction durations, with the number of Gaussians capped at 1M. Results are averaged over the Mip-NeRF360 [3] dataset. 3DGS time reports the target scene optimization, including densification, while init. time shows the iNGP reconstruction needed by our method and the initial matching for EDGS [27].

5.2. Ablations

We analyze the impact of individual components in our method by conducting a series of ablation studies, presented in Table 3. **(a), (b)** Longer iNGP reconstruction only slightly improves reconstruction quality. **(c)** continuing training the iNGP model also during optimization, **(d)** initializing Gaussians with the ground truth pixel color, or **(e)** predicting spherical harmonics with iNGP, leads to marginally worse results. Demonstrating the strength of our densification method, using **(f)** pixel-cone-sized primitives during initialization, or even not using any initialization **(g)**, results in worse PSNR but maintains low LPIPS and considerably improves rendering speed, thanks to less overlap between primitives, reducing blending. **(h)** Sampling pixels uniformly instead of guiding Gaussian creation using the L_1 loss from the training set produces lower reconstruction quality, although due to uniform sampling in image space still focusing more on parts of the scene seen most across views, the drop is not drastic. Densifying with 3DGS depth **(i)**, also without using iNGP even for initialization **(j)**, strongly affects the results. Similarly, k-NN sizing of newly added primitives based on their closest neighbors **(k)**, or changing the opacity penalty **(l), (m), (n)**, has a large effect on reconstruction quality, reinforcing the benefits of our densification approach. **(o), (p)** Altering the size of Gaussians created during densification from their default pixel-width cone size slightly reduces reconstruction quality, although the small difference suggests that Gaussians are quickly resized to fit the scene.

Ablation	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑
Ours	27.74	0.810	0.285	328
(a) 10k iNGP iter.	27.74	0.808	0.287	313
(b) 40k iNGP iter.	27.72	0.811	0.284	333
(c) Train iNGP during 3DGS	27.73	0.809	0.285	310
(d) Color from GT image	27.73	0.807	0.287	320
(e) Prediction of SH with iNGP	27.74	0.805	0.290	320
(f) Cone-sized initialization	27.38	0.806	0.287	437
(g) Without initialization	27.46	0.811	0.285	415
(h) Uniform image-space sampling	27.51	0.806	0.286	307
(i) Densify with 3DGS depth	27.43	0.797	0.296	332
(j) SfM initialization + 3DGS depth dens.	27.15	0.790	0.302	329
(k) Densify with k-NN scaling	27.54	0.802	0.295	299
(l) No opacity penalty	27.31	0.794	0.301	294
(m) Post-densification opacity decrease [5]	27.46	0.798	0.297	256
(n) MCMC-style opacity penalty [25]	27.49	0.803	0.293	239
(o) $\lambda_{\text{scale}} = 1$	27.70	0.810	0.285	329
(p) $\lambda_{\text{scale}} = 4$	27.63	0.808	0.285	329

Table 3. **Ablation study** of our method with 100k Gaussians, averaged over the Mip-NeRF360 [3] dataset. We highlight the best, second best, and third best results.

6. Conclusion

We introduce **ConeGS**, a reconstruction pipeline that replaces cloning-based densification with a method guided by photometric error and a coarse iNGP proxy, where new primitives are sized by pixel cones. Together with an improved opacity penalty, this allows creating primitives independently of existing structures through more flexible exploration. ConeGS consistently improves reconstruction quality and rendering performance across Gaussian budgets, with strong gains under tight primitive constraints. It achieves up to a 0.6 PSNR increase and a 20% speedup over cloning-based baselines.

Limitations: ConeGS works well on standard scenes but can struggle with very large scenes, inaccurate camera poses, or sparse viewpoints, sometimes producing extra floating Gaussians (see appendix). Its gains are also limited at very high Gaussian budgets, where dense coverage reduces the benefit of error-guided placement, offering mainly faster rendering.

References

[1] Mohamed Abdul Gafoor, Marius Preda, and Titus Zaharia. Refining gaussian splatting: A volumetric densification approach. In *Computer Science Research Notes*. University of West Bohemia, Czech Republic, 2025. 3

[2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. 2, 3, 15

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 1, 2, 6, 7, 8, 12, 13, 14, 15, 17, 18

[4] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023. 2, 15

[5] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting. *ArXiv*, abs/2404.06109, 2024. 3, 6, 8

[6] David Charatan, Sizhe Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *CVPR*, 2024. 3

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *ECCV*, 2022. 2

[8] Yuedong Chen, Haofei Xu, Chuanxia Zheng, Bohan Zhuang, Marc Pollefeys, Andreas Geiger, Tat-Jen Cham, and Jianfei Cai. Mvsplat: Efficient 3d gaussian splatting from sparse multi-view images. *arXiv preprint arXiv:2403.14627*, 2024. 3

[9] Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. GaussianPro: 3d gaussian splatting with progressive propagation. In *International Conference on Machine Learning (ICML)*, 2024. 3, 6, 7, 13

[10] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 3

[11] Guangchi Fang and Bing Wang. Mini-splatting2: Building 360 scenes within minutes via aggressive gaussian densification. *ArXiv*, abs/2411.12788, 2024. 3, 6, 7, 12, 13

[12] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians, 2024. 3

[13] Shuangkang Fang, I-Chao Shen, Takeo Igarashi, Yufeng Wang, ZeSheng Wang, Yi Yang, Wenrui Ding, and Shuchang Zhou. Nerf is a valuable assistant for 3d gaussian splatting, 2025. 3

[14] Yalda Foroutan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. Evaluating alternatives to sfm point cloud initialization for gaussian splatting. 2024. 3, 6, 7, 13

[15] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings, 2024. 3

[16] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *CVPR*, 2024. 3

[17] Zhihao Guo, Jingxuan Su, Shenglin Wang, Jinlong Fan, Jing Zhang, Li Hong Han, and Peng Wang. Gp-gs: Gaussian processes for enhanced gaussian splatting. *ArXiv*, abs/2502.02283, 2025. 3

[18] Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. Ges : Generalized exponential splatting for efficient radiance field rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19812–19822, 2024. 3

[19] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 2018. 6, 7, 13, 14, 17, 18

[20] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliege, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2, 3

[21] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. 2024. 3

[22] Binxiao Huang, Zhengwu Liu, and Ngai Wong. Decomposing densification in gaussian splatting for faster 3d scene reconstruction, 2025. 3

[23] Hanqing Jiang, Xiaojun Xiang, Han Sun, Hongjie Li, Liyang Zhou, Xiaoyu Zhang, and Guofeng Zhang. Geotexdensifier: Geometry-texture-aware densification for high-quality photorealistic 3d gaussian splatting, 2024. 3

[24] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 2, 3, 6, 7, 8, 13

[25] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 1, 3, 5, 6, 7, 8, 12, 13, 14

[26] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 1, 6, 7, 12, 13, 14, 17, 18

[27] Dmytro Kotovenko, Olga Grebenkova, and Björn Ommer. EDGS: eliminating densification for efficient convergence of 3dgs. *arXiv*, 2504.13204, 2025. 1, 3, 6, 7, 8, 12, 13, 15

[28] 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. *arXiv preprint arXiv:2403.06912*, 2024. 3

[29] Mingrui Li, Shuhong Liu, Tianchen Deng, and Hongyu Wang. Densesplat: Densifying gaussian splatting slam with neural radiance prior, 2025. 3

[30] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: Efficient sampling accelerates nerfs. *arXiv preprint arXiv:2305.04966*, 2023. 6

[31] Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. Deformable beta splatting, 2025. 3

[32] Yueh-Cheng Liu, Lukas Höllein, Matthias Nießner, and Angela Dai. Quicksplat: Fast 3d surface reconstruction via learned gaussian initialization. *ArXiv*, abs/2505.05591, 2025. 3

[33] Chongshan Lu, Fukun Yin, Xin Chen, Tao Chen, Gang YU, and Jiayuan Fan. A large-scale outdoor multi-modal dataset and benchmark for novel view synthesis and implicit scene reconstruction, 2023. 1, 6, 7, 8, 12, 13, 14, 15, 17, 18

[34] 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 *CVPR*, 2024. 2, 3

[35] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jon Barron, and Yinda Zhang. Ever: Exact volumetric ellipsoid rendering for real-time view synthesis, 2024. 3

[36] 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*, New York, NY, USA, 2024. Association for Computing Machinery. 3, 6

[37] 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 *ECCV*, 2020. 1, 2, 3

[38] Mahmud A. Mohamad, Gamal Elghazaly, Arthur Hubert, and Raphael Frank. Denser: 3d gaussians splatting for scene reconstruction of dynamic urban environments, 2024. 3

[39] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 2022. 2, 4

[40] Seungtae Nam, Xiangyu Sun, Gyeongjin Kang, Younggeun Lee, Seungjun Oh, and Eunbyung Park. Generative densification: Learning to densify gaussians for high-fidelity generalizable 3d reconstruction. *arXiv preprint arXiv:2412.06234*, 2024. 3

[41] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *CVPR*, 2024. 3

[42] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv.org*, 2024. 3, 4

[43] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics*, 4 (43), 2024. 3

[44] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2024. 3

[45] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 2

[46] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 2

[47] Chinmay Talegaonkar, Yash Belhe, Ravi Ramamoorthi, and Nicholas Antipa. Volumetrically consistent 3d gaussian rasterization, 2025. 3

[48] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023. 3

[49] Nicolas von Lützow and Matthias Nießner. Linprim: Linear primitives for differentiable volumetric rendering. *ArXiv*, abs/2501.16312, 2025. 2, 3

[50] Zipeng Wang and Dan Xu. Pygs: Large-scale scene representation with pyramidal 3d gaussian splatting, 2024. 3

[51] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004. 3, 6

[52] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *CVPR*, 2024. 2

[53] Zhiwen Yan, Weng Fei Low, Yu Chen, and Gim Hee Lee. Multi-scale 3d gaussian splatting for anti-aliased rendering. In *CVPR*, 2024. 2

[54] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023. 2

[55] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering fine details in 3d gaussian splatting. In *ACM MM*, 2024. 3

[56] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 2

[57] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *CVPR*, 2024. 2, 3

[58] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient high-quality compact surface reconstruction in unbounded scenes. *arXiv:2404.10772*, 2024. 3

[59] Zhaojie Zeng, Yuesong Wang, Lili Ju, and Tao Guan. Frequency-aware density control via reparameterization for high-quality rendering of 3d gaussian splatting. *ArXiv*, abs/2503.07000, 2025. 3

[60] Jiahui Zhang, Fangneng Zhan, Muyu Xu, Shijian Lu, and Eric P. Xing. Fregs: 3d gaussian splatting with progressive frequency regularization. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21424–21433, 2024. 3

[61] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, 2018. 6

[62] Xin Zhang, Anpei Chen, Jincheng Xiong, Pinxuan Dai, Yujun Shen, and Weiwei Xu. Neural shell texture splatting: More details and fewer primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 3

[63] Yangming Zhang, Wenqi Jia, Wei Niu, and Miao Yin. Gaussianspa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3d gaussian splatting. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26673–26682, 2024. 3

[64] Hongbi Zhou and Zhangkai Ni. Perceptual-gs: Scene-adaptive perceptual densification for gaussian splatting. *ArXiv*, abs/2506.12400, 2025. 3, 6, 7, 13

[65] Zheng Zhou, Yu-Jie Xiong, Chun-Ming Xia, Jia-Chen Zhang, and Hong-Jian Zhan. Gradient-direction-aware density control for 3d gaussian splatting. 2025. 3

[66] Yuanchen Guo Yangguang Li Ding Liang Yanpei Cao Songhai Zhang Zixin Zou, Zhipeng Yu. Triplane meets gaussian splatting: Fast and generalizable single-view 3d reconstruction with transformers. *arXiv preprint arXiv:2312.09147*, 2023. 3

[67] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, 2001. 3

Appendix

This appendix introduces additional results (Section A1) and ablations (Section A2). We then discuss our reconstructed scene structure (Section A3), possible failure cases (Section A4), and provide visualizations for the different types of initializations mentioned in the main paper (Section A5). Finally, we discuss the parallels between 3DGS and NeRF rendering, which allows training a radiance field using the rendering from 3DGS (Section A6).

A1. Detailed results

Table A1 and Table A2 present results for the no-budget scenario, which were used for plots in Figure 6. We use various values of the β parameter without specifying a budget for our method, except for the last cell, which uses the budget set by the number of Gaussians generated by 3D Gaussian Splatting. For each cell, we also compare MCMC [25] and EDGS [27] where they are set to match the number of Gaussians produced in each of the cells. The comparisons show that our method is able to produce high-quality results even without specifying a budget, instead adjusting the number of primitives based on the scene complexity, while still remaining sparse in the number of primitives. Notably, even when setting $\beta = 0$, which effectively disables densification, our method still performs well due to a dense initialization and effective filtering of unnecessary primitives enforced by the opacity penalty, consistent with findings in [27]. However, this configuration does not allow explicit control over the number of primitives and may limit the achievable reconstruction quality.

If an even lower number of Gaussians is desired while still using a no-budget scenario, the number of initialized Gaussians can be reduced, effectively lowering the number of primitives generated at the end.

We show in Table A3 the results on the selected benchmarks for the budget of 1M primitives. Additionally, we present per-scene results for 100k Gaussians in Table A8,

Ablation	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑	# Gaussians
Ours ($\beta = 0$)	29.12	0.873	0.183	185	542k
MCMC (SfM)	28.86	0.865	0.209	136	
EDGS	28.87	0.868	0.187	194	
Ours ($\beta = 0.01$)	29.25	0.877	0.174	169	674k
MCMC (SfM)	29.06	0.870	0.199	119	
EDGS	28.97	0.873	0.178	167	
Ours ($\beta = 0.02$)	29.34	0.880	0.166	146	942k
MCMC (SfM)	29.15	0.876	0.189	101	
EDGS	29.10	0.878	0.167	134	
Ours ($\beta = 0.04$)	29.38	0.881	0.161	105	1.66M
MCMC (SfM)	29.32	0.882	0.174	77	
EDGS	29.19	0.881	0.159	102	
Ours	29.35	0.879	0.159	80	2.57M
MCMC (SfM)	29.56	0.887	0.164	55	
EDGS	29.37	0.884	0.153	67	
3DGS	29.03	0.870	0.184	69	

Table A1. **No-budget Mip-NeRF360.** Comparison of reconstruction quality without a fixed limit on the number of primitives. For each cell, our method is run with a different β parameter, which determines the number of Gaussians generated, and the other methods are limited to this number. In the last cell, the number of Gaussians is set to match the amount produced by 3DGS, with all other methods constrained accordingly. Results are averaged over the Mip-NeRF360 [3] dataset.

Ablation	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑	# Gaussians
Ours ($\beta = 0$)	28.54	0.879	0.195	253	352k
MCMC (SfM)	28.37	0.877	0.207	199	
EDGS	28.62	0.885	0.183	219	
Ours ($\beta = 0.01$)	29.02	0.890	0.176	237	438k
MCMC (SfM)	28.55	0.882	0.199	175	
EDGS	28.84	0.890	0.173	211	
Ours ($\beta = 0.02$)	29.19	0.894	0.166	217	581k
MCMC (SfM)	28.79	0.888	0.190	157	
EDGS	29.07	0.896	0.163	188	
Ours ($\beta = 0.04$)	29.32	0.898	0.157	116	927k
MCMC (SfM)	29.05	0.895	0.177	105	
EDGS	29.35	0.902	0.151	134	
Ours	29.60	0.904	0.144	114	1.75M
MCMC	29.46	0.904	0.157	89	
EDGS	29.85	0.909	0.137	99	
3DGS	29.30	0.896	0.171	88	

Table A2. **No-budget OMMO.** Comparison of reconstruction quality without a fixed limit on the number of primitives. For each cell, our method is run with a different β parameter, which determines the number of Gaussians generated, and the other methods are limited to this number. In the last cell, the number of Gaussians is set to match the amount produced by 3DGS, with all other methods constrained accordingly. Results are averaged over the OMMO [33] dataset.

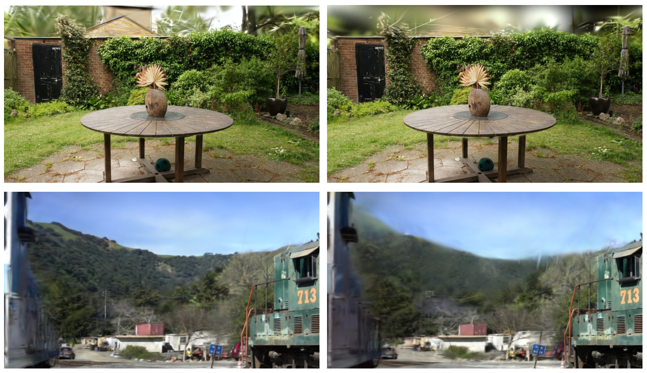


Figure A1. **Qualitative comparison** between our approach and Mini-Splatting2 [11] on the `garden` scene from Mip-NeRF360 [3] and on the `train` scene from Tanks & Temples [26].

	Mip-NeRF360 [3]			OMMO [33]			Tanks & Temples [26]			DeepBlending [19]		
	PSNR ↑	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Number of Gaussians limited to 1M												
3DGS [24] (SfM init.)	28.71	0.846	0.222	29.40	0.894	0.178	23.60	0.826	0.245	28.93	0.881	0.292
Foroutan et al. [14]†	29.22	0.876	0.177	29.32	0.896	0.163	23.75	0.829	0.230	29.60	0.886	0.282
MCMC [25] (rand. init.)	28.98	0.865	0.204	28.83	0.888	0.185	23.41	0.824	0.249	28.94	0.877	0.303
MCMC [25] (SfM init.)	29.23	0.875	0.190	29.18	0.895	0.174	23.93	0.836	0.233	29.67	0.887	0.288
MCMC [25] (iNGP init.)	29.32	0.879	0.173	28.76	0.887	0.181	23.37	0.804	0.268	29.87	0.892	0.276
GaussianPro [9]	28.56	0.845	0.226	28.79	0.885	0.189	23.01	0.818	0.256	29.27	0.887	0.291
Perceptual-GS [64]	29.27	0.876	0.176	29.00	0.888	0.179	23.26	0.828	0.240	29.41	0.889	0.286
EDGS [27]	29.18	0.877	0.168	29.58	0.903	0.149	23.66	0.842	0.200	29.43	0.889	0.271
Ours	29.37	0.880	0.168	29.44	0.899	0.154	23.70	0.835	0.207	29.69	0.889	0.273

† Original code was not publicly available. Our implementation uses iNGP initialization and does not include the additional depth-based loss.

Table A3. **Quantitative results** with a Gaussian number limit of 1M. We highlight the best, second best and third best results among methods with comparable numbers of Gaussians.

	Ours	3DGS (SfM)	MCMC (SfM)	EDGS	Perceptual-GS
# Memory (MiB)	9545	9049	8671	14517	12403

Table A4. **Peak GPU memory usage** on the `15` scene from the OMMO dataset [33] on the maximum budget of 500k primitives. We highlight the best, second best and third best results among all.

	PSNR ↑	**SSIM ↑**	**LPIPS ↓**	**FPS ↑**	**Train (min) ↓**
ConeGS	29.14	0.892	0.170	217	25
ConeGS (iNGP retrain)	29.27	0.893	0.168	212	29

Table A5. **Ablation on additional iNGP training** on the OMMO [33] dataset with 500k primitives, evaluating the effect of continuing training iNGP in parallel with the full 3DGS optimization. We highlight the best, second best and third best results among all.

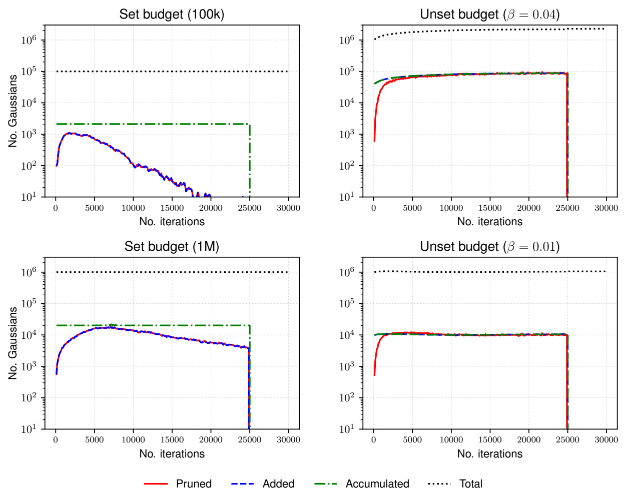


Figure A2. **Number of primitives** pruned, added, accumulated, as well as the total number, during each densification, on different budget scenarios. Results obtained from the `garden` scene from Mip-NeRF360 [3].

500k Gaussians in Table A9, 1M Gaussians in Table A10, and 2M Gaussians in Table A11. We also show additional qualitative results with Mini-Splatting2 [11] in Figure A1.

A2. Additional ablations and comparisons

We evaluate the peak GPU memory usage during optimization for several methods in Table A4. The results show that our method is very close to 3DGS [24] and MCMC [25] in terms of memory consumption, while remaining considerably lower than EDGS [27] and Perceptual-GS [64]. This efficiency allows our method to run on a wider range of GPUs, making it more accessible to devices with limited memory.

We expand on ablation **(c)** from the main paper, where iNGP continues training in parallel with the full 3DGS optimization. Since iNGP training, like densification, is guided by the L1 error from 3DGS, the iNGP model can better focus on regions where 3DGS reconstruction may fall short, potentially improving densification. The original ablation was tested on a budget of 100k primitives, which may not fully reveal this effect. To explore further, we run experiments with larger budgets. On the 500k budget for challenging OMMO [33] scenes (Table A5), we observe additional performance gains, though at the cost of longer training time. We also test this setup with a budget of 1M primitives across all datasets (Table A6), finding minimal improvements on difficult scenes and slight decreases on Mip-NeRF360, which may be caused by overfitting to certain areas.

Figure A2 illustrates the number of primitives added and removed during each densification and pruning step. It also tracks the accumulation buffer of primitives. When no budget is specified, all accumulated primitives are added to the scene (see Eq. 17). Under a fixed budget, however, not all of them are used. This is because accumulation happens every iteration and must remain available for pruning and

	Mip-NeRF360 [3]			OMMO [33]			Tanks & Temples [26]			DeepBlending [19]		
	PSNR ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Number of Gaussians limited to 1M												
Ours	29.37	0.880	0.168	29.44	0.899	0.154	23.70	0.835	0.207	29.69	0.889	0.273
Ours, iNGP training during 3DGS	29.25	0.879	0.168	29.54	0.900	0.152	23.72	0.836	0.207	29.73	0.889	0.273

Table A6. **Ablation on additional iNGP training** for the budget of 1M primitives. We highlight the best, second best and third best results among methods with comparable numbers of Gaussians.

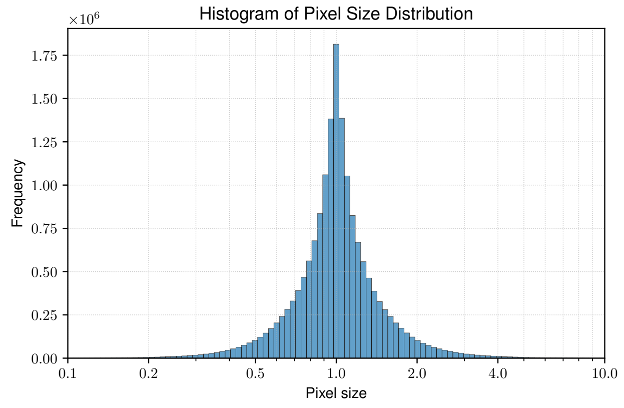


Figure A3. **The histogram of perceived image-space size** of pixel-sized Gaussians rendered from different viewpoints. Size expressed in pixel widths. Analysis doesn't include the low-pass filter from 3DGS rasterization.

	Ours	MCMC (SfM)	MCMC	3DGS (SfM)
# Gaussians per pixel	30.72	49.55	48.45	34.74

Table A7. **Blending analysis.** Mean number of Gaussians contributing to alpha blending per pixel, averaged across the Mip-NeRF360 [3] dataset using a budget of 1M Gaussians. We highlight the best, second best and third best results among all.

densification, while the exact number that can be added is unknown beforehand. As a result, the system accumulates more primitives than are usually required to keep the total count close to the budget after pruning (see Eq. 16).

The primitive size is defined to be approximately one pixel from a single viewpoint. From other viewpoints, this size is not exactly one pixel, but should remain close. To confirm this, we measure the size of newly added pixel-sized primitives from multiple viewpoints and report the results in Figure A3. The distribution shows that the apparent size remains near one pixel, with very few cases above four pixels or below 0.2 pixels.

A3. Scene structure

To confirm that our method produces reconstructions with desirable characteristics, such as a balanced distribution of scaling values and placement close to surfaces, which are often important for downstream tasks, we analyze the scenes created with our method in comparison to MCMC [25].

Although Gaussian Splatting with the MCMC densifica-

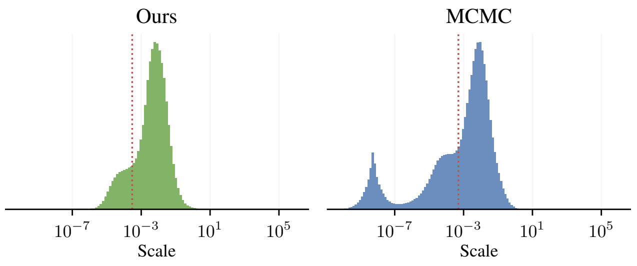


Figure A4. **Histograms of the Gaussian scaling values** for our method and MCMC with SfM initialization on 2M Gaussians. The red lines indicate the minimum scaling required for a Gaussian to cover at least one pixel, disregarding the low-pass filter.

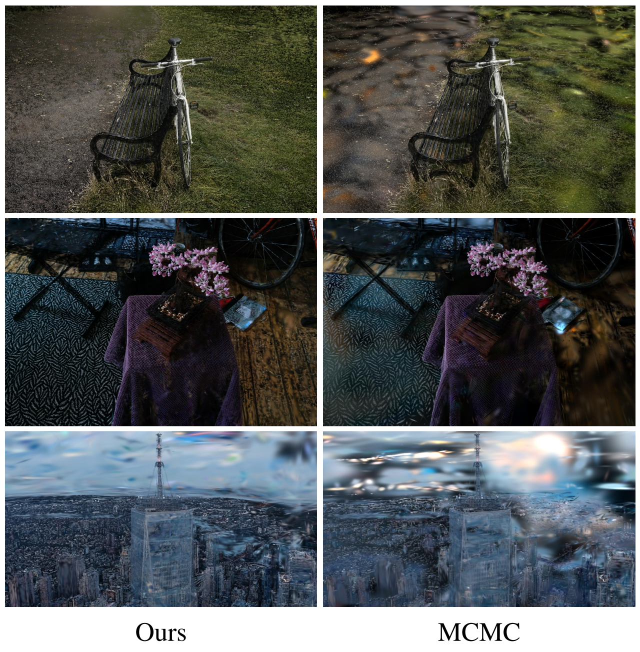


Figure A5. **Shrunk Gaussians.** Visual comparison between the proposed method and MCMC, rendered using Gaussians scaled to half their original size. Both methods were trained with a limit of 1 million primitives on the `bicycle` and `bonsai` scenes from Mip-NeRF360 [3], and the `10` scene from OMMO [33].

tion strategy explores the scene effectively, it also has clear shortcomings. Its cloning strategy and scaling penalty often cause Gaussians to shrink strongly along certain dimensions. The low-pass filter used by 3DGS renderer can hide this effect visually, but it prevents Gaussians from expanding in those directions and interferes with tasks that depend on accurate scales, such as MCMC position error. As shown in Figure A4, many of its Gaussians shrink below the pixel radius, in contrast to the more balanced distribution produced by our method.

Figure A5 shows that our approach produces Gaussians that are more uniform in size and placed closer to object surfaces. This avoids an overreliance on oversized background primitives located far from the surface. This primitive distribution not only improves geometric alignment but also increases rendering speed by reducing blending and sorting overhead during rasterization. To support this hypothesis, Table A7 reports the mean number of Gaussians blended per pixel. Our method consistently requires less blending compared to the selected benchmarks. The difference is especially large compared to MCMC, which blends over 60% more Gaussians per pixel on average.

A4. Failure cases

While our method generally produces strong reconstructions on almost all tested scenes, its reliance on iNGP can make it susceptible to floaters in challenging scenarios such as noisy poses, very large or sparse-view scenes, or other cases where reliable iNGP reconstruction is difficult. One such example is shown in Figure A6, where the scene contains distortions and lacks sufficient viewpoint coverage near the cameras. This leads to spurious high-density regions in iNGP close to the cameras, which in turn degrades densification quality by placing Gaussians in incorrect regions. Although this can reduce performance, the method still produces high-quality reconstructions overall (see per-scene results).

A5. Initialization visualizations

The choice of initialization has a strong impact on the performance of 3DGS reconstruction. A sufficiently good initialization can even remove the need for additional densification ([27], Table A1, Table A2). We visualize different initialization types in Figure A7. Initialization from a sparse SfM point cloud often leaves large gaps that are difficult to fill correctly, while our initialization produces a more uniform coverage. Another important strategy is initialization based on pixel width. This approach does not provide a useful inductive bias of larger primitives and can lead to more gaps in unobserved regions, although it may offer faster rendering due to reduced blending (Section 5.2). Finally, our formulation allows scaling the initial primitives in image space, producing a balance between these two types of initialization.

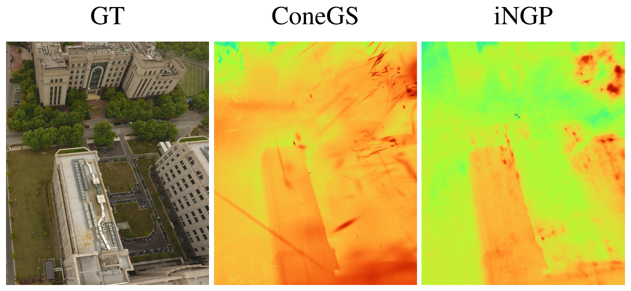


Figure A6. **Depth maps.** Depth maps for ConeGS and the iNGP reconstruction model on the `01` scene from the OMMO [33] dataset. Since the Gaussians are generated from an iNGP reconstruction, the presence of floaters in it leads to floaters also appearing in the Gaussian Splatting results.

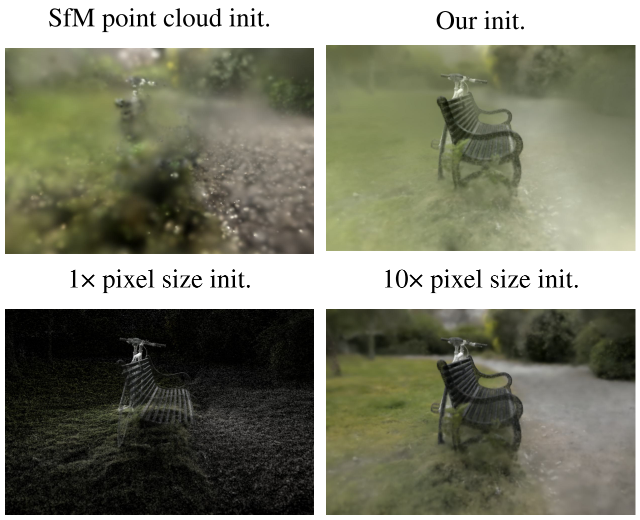


Figure A7. **Initialization Methods Comparison.** Visual comparison of four initialization methods for 3DGS reconstruction on the `bicycle` scene from Mip-NeRF 360 dataset [3]: (1) SfM point cloud initialization, (2) iNGP initialization with kNN-based sizing, (3) iNGP initialization with 1× sizing, and (4) iNGP initialization using the smaller of 10× pixel size or kNN-based sizing.

A6. Gaussian-Based Radiance Field training

To establish a more direct connection between the volumetric ray marching procedure in Neural Radiance Fields (NeRF) and the rendering in 3D Gaussian Splatting (3DGS), the set of conical frustums sampled along a cone in Mip-NeRF and its derivatives [2–4] can be reinterpreted as a set of 3D Gaussian primitives, covering approximately the same area. The purpose of this reformulation is to enable training a neural radiance field model using only the 3DGS renderer, without using the traditional volumetric ray integration.

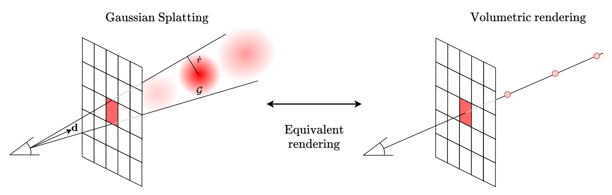


Figure A8. **Training equivalence.** Illustration of the equivalence between training an implicit radiance field model using NeRF-style ray marching and 3D Gaussian Splatting rasterization.

To align the 3DGS and NeRF rendering formulations, it is necessary to demonstrate that an entire ray can be equivalently represented as a sequence of Gaussians such that, when rendered, the result is equal to the volumetric integration process. This is achieved by casting cones along pixel directions and subdividing each into a set of conical frustums. Each frustum is then mapped to a Gaussian primitive, where the midpoint

$$t_{\mu,i} = \frac{t_i + t_{i+1}}{2} \tag{A1}$$

is used to define the Gaussian center $\mathbf{p}_i$. The view-dependent RGB color $\mathbf{c}_i$ and scalar density σ_i, which is converted to opacity o_i using Eq. 6, are predicted by a neural network. As detailed in Section 4.1, the predicted RGB color $\mathbf{c}_i$ is encoded as spherical harmonics coefficients, which are compatible with the 3DGS rasterizer.

Since the Gaussians lie along the cone and their centers are co-linear with the pixel center and the camera origin, their projected 2D means fall directly on the target pixel. Consequently, the maximum opacity contribution aligns with the pixel center, and the kernel evaluation satisfies

$$K(\mathbf{p}_c, \boldsymbol{\mu}_i^{\text{2D}}, \boldsymbol{\Sigma}_i^{\text{2D}}) = 1. \tag{A2}$$

This condition ensures that the Gaussian opacity o_i corresponds directly to the opacity α_i used in blending operations, as defined in Eq. 6 and Eq. 2. The opacity is therefore determined by the predicted density and the length of the corresponding frustum.

To maintain consistent 2D projection footprints across different depths, each Gaussian's 3D covariance must be adjusted based on its location along the cone. Because elongating Gaussians along the ray direction does not affect the resulting 2D projection, the scaling can be derived using the pixel footprint size and set isotropically using the formulation in Eq. 15, while the quaternion can be set to the identity quaternion.

Given these assignments for position, color, opacity, scale, and orientation, each Gaussian can be rendered using the 3DGS renderer to produce the same pixel color as that produced by volumetric rendering. Assuming no low-pass filtering is applied and that only a single image is rendered at a time, it becomes possible to train an Instant-NGP-style model using only the 3DGS rendering pipeline. Although this approach is marginally slower than traditional ray marching, it yields equivalent radiance field representations. A visual comparison of both approaches is shown in Figure A8.

This reinterpretation also provides a more principled foundation for the proposed densification strategy. Instead of using depth, the strategy can be understood as selecting a Gaussian that corresponds to a surface-level conical frustum of pixels with high photometric error. This establishes a clearer theoretical link between cone-based densification and neural implicit models such as NeRF.

	Scene	Ours	3DGS (SfM)	MCMC (SfM)	EDGS
Mip-NeRF360 [3]	Bicycle	24.05 / 0.652 / 0.379 / 368	20.99 / 0.470 / 0.543 / 466	23.34 / 0.630 / 0.400 / 339	23.68 / 0.635 / 0.393 / 444
	Garden	24.69 / 0.709 / 0.336 / 431	22.87 / 0.589 / 0.445 / 430	24.51 / 0.710 / 0.350 / 569	24.47 / 0.706 / 0.341 / 493
	Stump	25.71 / 0.707 / 0.351 / 456	22.79 / 0.547 / 0.498 / 506	25.16 / 0.680 / 0.370 / 402	25.08 / 0.673 / 0.377 / 588
	Room	31.09 / 0.903 / 0.254 / 275	27.02 / 0.859 / 0.328 / 267	30.37 / 0.900 / 0.270 / 224	30.08 / 0.896 / 0.265 / 317
	Counter	28.22 / 0.879 / 0.251 / 229	25.64 / 0.845 / 0.307 / 252	27.83 / 0.880 / 0.260 / 198	27.77 / 0.877 / 0.251 / 245
	Kitchen	29.73 / 0.896 / 0.183 / 239	20.53 / 0.671 / 0.440 / 297	28.37 / 0.890 / 0.210 / 246	28.84 / 0.891 / 0.186 / 269
	Bonsai	30.67 / 0.920 / 0.241 / 276	25.41 / 0.866 / 0.331 / 326	29.84 / 0.910 / 0.260 / 281	29.73 / 0.906 / 0.256 / 298
	Average	27.74 / 0.809 / 0.285 / 325	23.61 / 0.692 / 0.413 / 363	27.06 / 0.800 / 0.303 / 323	27.09 / 0.798 / 0.296 / 379
OMMO [33]	01	22.58 / 0.625 / 0.458 / 212	20.53 / 0.543 / 0.575 / 159	22.66 / 0.620 / 0.470 / 180	22.50 / 0.608 / 0.475 / 222
	03	25.39 / 0.842 / 0.244 / 356	24.09 / 0.803 / 0.295 / 112	24.30 / 0.810 / 0.290 / 217	24.85 / 0.827 / 0.259 / 313
	05	27.95 / 0.863 / 0.246 / 415	26.87 / 0.841 / 0.296 / 354	27.70 / 0.860 / 0.270 / 318	27.69 / 0.856 / 0.259 / 398
	06	27.30 / 0.913 / 0.202 / 321	26.53 / 0.899 / 0.229 / 107	26.25 / 0.910 / 0.210 / 339	26.60 / 0.910 / 0.198 / 339
	10	29.31 / 0.853 / 0.252 / 338	28.86 / 0.833 / 0.291 / 125	28.70 / 0.840 / 0.280 / 337	28.78 / 0.839 / 0.275 / 341
	13	30.36 / 0.899 / 0.211 / 449	29.79 / 0.889 / 0.247 / 171	29.45 / 0.880 / 0.250 / 378	28.98 / 0.878 / 0.234 / 468
	14	29.73 / 0.924 / 0.145 / 394	27.09 / 0.874 / 0.220 / 237	29.26 / 0.920 / 0.160 / 360	28.92 / 0.909 / 0.169 / 374
	15	28.10 / 0.893 / 0.182 / 405	27.80 / 0.877 / 0.213 / 153	27.72 / 0.890 / 0.200 / 427	27.58 / 0.877 / 0.215 / 420
	Average	27.59 / 0.852 / 0.242 / 361	26.45 / 0.820 / 0.296 / 177	27.00 / 0.841 / 0.266 / 320	26.99 / 0.838 / 0.261 / 359
Tanks & Temples [26]	Truck	24.54 / 0.822 / 0.291 / 184	23.59 / 0.803 / 0.318 / 128	23.86 / 0.810 / 0.316 / 155	23.33 / 0.799 / 0.314 / 183
	Train	21.70 / 0.759 / 0.328 / 204	21.17 / 0.744 / 0.349 / 118	21.13 / 0.749 / 0.347 / 160	21.31 / 0.756 / 0.334 / 180
	Average	23.12 / 0.790 / 0.309 / 194	22.38 / 0.774 / 0.334 / 123	22.49 / 0.780 / 0.332 / 158	22.32 / 0.778 / 0.324 / 182
Deep Blending [19]	Dr Johnson	28.78 / 0.875 / 0.338 / 447	23.58 / 0.808 / 0.440 / 521	28.16 / 0.869 / 0.343 / 550	27.66 / 0.865 / 0.347 / 444
	Playroom	30.09 / 0.885 / 0.318 / 489	25.72 / 0.845 / 0.384 / 489	29.72 / 0.883 / 0.323 / 552	29.19 / 0.879 / 0.327 / 486
	Average	29.44 / 0.880 / 0.328 / 468	24.65 / 0.827 / 0.412 / 505	28.94 / 0.876 / 0.333 / 551	28.43 / 0.872 / 0.337 / 465

Table A8. **Detailed results** on a selection of datasets and methods with the number of Gaussians limited to 100k. Each field contains PSNR, SSIM, LPIPS and FPS respectively. We highlight the best, second best and third best results among all. Slight discrepancies from the main table are due to rounding.

	Scene	Ours	3DGS (SfM)	MCMC (SfM)	EDGS
Mip-NeRF360 [3]	Bicycle	25.25 / 0.758 / 0.242 / 234	24.06 / 0.651 / 0.370 / 174	24.90 / 0.740 / 0.282 / 162	24.95 / 0.751 / 0.251 / 248
	Garden	26.87 / 0.837 / 0.157 / 246	25.16 / 0.743 / 0.296 / 297	26.36 / 0.823 / 0.189 / 221	26.49 / 0.833 / 0.161 / 252
	Stump	27.03 / 0.793 / 0.215 / 255	25.27 / 0.688 / 0.348 / 226	26.81 / 0.779 / 0.250 / 175	26.50 / 0.768 / 0.237 / 296
	Room	32.02 / 0.924 / 0.205 / 192	31.47 / 0.913 / 0.234 / 112	31.72 / 0.921 / 0.221 / 114	31.36 / 0.923 / 0.203 / 168
	Counter	28.87 / 0.906 / 0.194 / 148	28.86 / 0.901 / 0.213 / 113	28.95 / 0.907 / 0.205 / 93	29.05 / 0.912 / 0.184 / 137
	Kitchen	31.38 / 0.929 / 0.124 / 172	30.76 / 0.918 / 0.143 / 111	31.04 / 0.921 / 0.141 / 111	31.35 / 0.928 / 0.122 / 154
	Bonsai	32.16 / 0.944 / 0.191 / 190	31.95 / 0.936 / 0.218 / 135	31.97 / 0.939 / 0.210 / 128	32.02 / 0.942 / 0.191 / 170
	Average	29.08 / 0.870 / 0.190 / 205	28.22 / 0.821 / 0.260 / 167	28.82 / 0.861 / 0.214 / 143	28.82 / 0.865 / 0.193 / 204
OMMO [33]	01	23.46 / 0.690 / 0.356 / 140	23.81 / 0.682 / 0.385 / 102	23.73 / 0.685 / 0.378 / 90	23.63 / 0.686 / 0.363 / 84
	03	27.01 / 0.887 / 0.181 / 197	26.18 / 0.868 / 0.220 / 111	26.50 / 0.875 / 0.212 / 105	27.16 / 0.892 / 0.176 / 93
	05	28.61 / 0.877 / 0.199 / 232	28.53 / 0.874 / 0.235 / 162	28.71 / 0.877 / 0.233 / 154	28.79 / 0.881 / 0.196 / 122
	06	27.79 / 0.932 / 0.159 / 195	27.59 / 0.931 / 0.161 / 120	26.99 / 0.934 / 0.159 / 152	27.38 / 0.941 / 0.135 / 105
	10	31.16 / 0.905 / 0.165 / 235	31.04 / 0.894 / 0.194 / 143	30.56 / 0.892 / 0.192 / 168	31.11 / 0.906 / 0.165 / 129
	13	33.02 / 0.949 / 0.115 / 280	32.40 / 0.939 / 0.149 / 165	32.18 / 0.934 / 0.155 / 175	31.95 / 0.944 / 0.123 / 151
	14	31.57 / 0.950 / 0.096 / 225	31.57 / 0.946 / 0.107 / 135	31.14 / 0.946 / 0.108 / 130	31.51 / 0.950 / 0.095 / 112
	15	30.50 / 0.944 / 0.090 / 231	30.59 / 0.934 / 0.114 / 149	29.92 / 0.933 / 0.113 / 162	30.52 / 0.942 / 0.095 / 125
	Average	29.14 / 0.892 / 0.170 / 217	28.96 / 0.884 / 0.196 / 136	28.72 / 0.885 / 0.194 / 142	29.01 / 0.893 / 0.169 / 115
Tanks & Temples [26]	Truck	25.41 / 0.858 / 0.204 / 122	24.80 / 0.842 / 0.253 / 93	25.24 / 0.851 / 0.242 / 68	24.66 / 0.850 / 0.212 / 114
	Train	21.97 / 0.799 / 0.253 / 140	22.28 / 0.790 / 0.277 / 72	22.12 / 0.798 / 0.275 / 86	22.27 / 0.812 / 0.246 / 104
	Average	23.69 / 0.829 / 0.229 / 131	23.54 / 0.816 / 0.265 / 82	23.68 / 0.825 / 0.259 / 77	23.47 / 0.831 / 0.229 / 109
Deep Blending [19]	Dr Johnson	29.20 / 0.890 / 0.291 / 270	28.85 / 0.881 / 0.307 / 151	28.71 / 0.884 / 0.313 / 180	28.60 / 0.888 / 0.292 / 269
	Playroom	30.51 / 0.892 / 0.279 / 287	29.66 / 0.883 / 0.298 / 161	30.16 / 0.889 / 0.303 / 209	30.06 / 0.890 / 0.281 / 277
	Average	29.86 / 0.891 / 0.285 / 278	29.26 / 0.882 / 0.303 / 156	29.44 / 0.887 / 0.308 / 194	29.33 / 0.889 / 0.287 / 273

Table A9. **Detailed results** on a selection of datasets and methods with the number of Gaussians limited to 500k. Each field contains PSNR, SSIM, LPIPS and FPS respectively. We highlight the best, second best and third best results among all. Slight discrepancies from the main table are due to rounding.

	Scene	Ours	3DGS (SfM)	MCMC (SfM)	EDGS
Mip-NeRF360 [3]	Bicycle	25.45 / 0.778 / 0.198 / 156	24.40 / 0.688 / 0.325 / 129	25.34 / 0.768 / 0.238 / 106	25.26 / 0.778 / 0.199 / 161
	Garden	27.42 / 0.861 / 0.115 / 162	26.70 / 0.827 / 0.172 / 142	26.83 / 0.847 / 0.147 / 135	27.05 / 0.857 / 0.118 / 158
	Stump	27.23 / 0.802 / 0.184 / 164	25.64 / 0.719 / 0.301 / 162	27.21 / 0.798 / 0.213 / 110	26.71 / 0.783 / 0.203 / 179
	Room	32.30 / 0.928 / 0.196 / 139	31.62 / 0.918 / 0.221 / 82	32.09 / 0.926 / 0.208 / 79	31.71 / 0.928 / 0.191 / 111
	Counter	29.21 / 0.911 / 0.181 / 107	29.11 / 0.907 / 0.201 / 75	29.24 / 0.913 / 0.191 / 61	29.27 / 0.917 / 0.171 / 94
	Kitchen	31.66 / 0.932 / 0.117 / 118	31.18 / 0.924 / 0.131 / 78	31.44 / 0.927 / 0.130 / 72	31.83 / 0.933 / 0.114 / 101
	Bonsai	32.33 / 0.945 / 0.183 / 132	32.34 / 0.941 / 0.205 / 95	32.46 / 0.944 / 0.199 / 81	32.44 / 0.947 / 0.181 / 115
	Average	29.37 / 0.880 / 0.168 / 140	28.71 / 0.846 / 0.222 / 109	29.23 / 0.875 / 0.189 / 92	29.18 / 0.878 / 0.168 / 131
OMMO [33]	01	23.73 / 0.711 / 0.315 / 104	24.18 / 0.703 / 0.350 / 73	24.18 / 0.709 / 0.341 / 63	24.02 / 0.709 / 0.321 / 84
	03	27.39 / 0.896 / 0.167 / 127	26.96 / 0.886 / 0.197 / 64	27.20 / 0.890 / 0.189 / 67	27.72 / 0.905 / 0.157 / 85
	05	28.56 / 0.877 / 0.185 / 157	28.72 / 0.877 / 0.227 / 115	28.98 / 0.883 / 0.217 / 98	29.15 / 0.885 / 0.181 / 101
	06	27.87 / 0.936 / 0.150 / 136	27.68 / 0.933 / 0.158 / 110	27.46 / 0.939 / 0.146 / 103	27.75 / 0.945 / 0.127 / 102
	10	31.60 / 0.914 / 0.147 / 170	31.47 / 0.906 / 0.171 / 104	30.89 / 0.904 / 0.169 / 120	31.82 / 0.920 / 0.138 / 116
	13	33.55 / 0.957 / 0.098 / 190	33.13 / 0.949 / 0.129 / 116	32.83 / 0.945 / 0.131 / 124	32.83 / 0.955 / 0.100 / 125
	14	31.80 / 0.953 / 0.090 / 146	31.93 / 0.951 / 0.097 / 89	31.44 / 0.950 / 0.099 / 96	32.08 / 0.955 / 0.086 / 92
	15	30.99 / 0.950 / 0.081 / 151	31.13 / 0.943 / 0.096 / 100	30.42 / 0.942 / 0.097 / 104	31.25 / 0.950 / 0.079 / 104
	Average	29.44 / 0.899 / 0.154 / 148	29.40 / 0.894 / 0.178 / 96	29.18 / 0.895 / 0.174 / 97	29.58 / 0.903 / 0.149 / 101
Tanks & Temples [26]	Truck	25.34 / 0.864 / 0.181 / 97	25.06 / 0.851 / 0.234 / 70	25.57 / 0.862 / 0.217 / 45	25.02 / 0.862 / 0.181 / 82
	Train	22.05 / 0.805 / 0.232 / 109	22.13 / 0.801 / 0.255 / 51	22.29 / 0.811 / 0.249 / 47	22.29 / 0.823 / 0.220 / 78
	Average	23.70 / 0.835 / 0.207 / 103	23.60 / 0.826 / 0.245 / 60	23.93 / 0.837 / 0.233 / 46	23.66 / 0.843 / 0.201 / 80
Deep Blending [19]	Dr Johnson	28.98 / 0.886 / 0.282 / 194	28.85 / 0.884 / 0.293 / 106	29.06 / 0.884 / 0.291 / 164	28.75 / 0.890 / 0.277 / 180
	Playroom	30.39 / 0.891 / 0.264 / 200	29.02 / 0.877 / 0.290 / 112	30.28 / 0.890 / 0.284 / 173	30.11 / 0.889 / 0.265 / 182
	Average	29.69 / 0.889 / 0.273 / 197	28.94 / 0.881 / 0.292 / 109	29.67 / 0.887 / 0.288 / 168	29.43 / 0.890 / 0.271 / 181

Table A10. **Detailed results** on a selection of datasets and methods with the number of Gaussians limited to 1M. Each field contains PSNR, SSIM, LPIPS and FPS respectively. We highlight the best, second best and third best results among all. Slight discrepancies from the main table are due to rounding.

	Scene	Ours	3DGS (SfM)	MCMC (SfM)	EDGS
Mip-NeRF360 [3]	Bicycle	25.43 / 0.781 / 0.175 / 99	24.85 / 0.729 / 0.267 / 79	25.56 / 0.786 / 0.205 / 71	25.48 / 0.792 / 0.168 / 93
	Garden	27.59 / 0.870 / 0.097 / 97	27.22 / 0.852 / 0.129 / 84	27.33 / 0.862 / 0.122 / 82	27.46 / 0.870 / 0.097 / 89
	Stump	27.03 / 0.796 / 0.177 / 100	26.18 / 0.749 / 0.255 / 100	27.39 / 0.808 / 0.189 / 69	26.79 / 0.788 / 0.186 / 100
	Room	32.32 / 0.929 / 0.189 / 92	31.80 / 0.919 / 0.218 / 63	32.14 / 0.929 / 0.199 / 53	31.89 / 0.930 / 0.184 / 81
	Counter	29.23 / 0.912 / 0.175 / 70	29.07 / 0.907 / 0.201 / 68	29.41 / 0.917 / 0.181 / 42	29.37 / 0.918 / 0.164 / 65
	Kitchen	31.80 / 0.934 / 0.114 / 72	31.60 / 0.927 / 0.126 / 54	32.00 / 0.931 / 0.122 / 47	32.06 / 0.935 / 0.111 / 65
	Bonsai	32.63 / 0.948 / 0.177 / 84	32.34 / 0.941 / 0.204 / 84	32.80 / 0.948 / 0.189 / 53	32.62 / 0.947 / 0.175 / 85
	Average	29.43 / 0.881 / 0.158 / 88	29.01 / 0.861 / 0.200 / 76	29.52 / 0.883 / 0.172 / 60	29.38 / 0.883 / 0.155 / 83
OMMO [33]	01	23.92 / 0.725 / 0.283 / 73	24.51 / 0.721 / 0.321 / 49	24.51 / 0.731 / 0.303 / 45	24.24 / 0.725 / 0.285 / 64
	03	27.73 / 0.902 / 0.155 / 70	27.05 / 0.888 / 0.193 / 47	27.73 / 0.900 / 0.171 / 45	27.77 / 0.909 / 0.149 / 82
	05	28.65 / 0.877 / 0.176 / 91	28.69 / 0.877 / 0.227 / 112	29.20 / 0.887 / 0.201 / 67	29.31 / 0.887 / 0.169 / 90
	06	27.93 / 0.937 / 0.143 / 79	27.67 / 0.933 / 0.157 / 109	27.58 / 0.942 / 0.138 / 63	27.93 / 0.947 / 0.123 / 97
	10	31.81 / 0.920 / 0.135 / 104	31.77 / 0.915 / 0.153 / 70	31.59 / 0.913 / 0.151 / 80	32.26 / 0.928 / 0.123 / 89
	13	33.88 / 0.961 / 0.088 / 112	33.79 / 0.957 / 0.112 / 73	33.30 / 0.953 / 0.111 / 80	33.55 / 0.961 / 0.087 / 90
	14	32.02 / 0.955 / 0.086 / 83	31.95 / 0.951 / 0.096 / 82	31.75 / 0.953 / 0.093 / 62	31.76 / 0.956 / 0.084 / 86
	15	31.25 / 0.953 / 0.075 / 86	31.35 / 0.947 / 0.090 / 83	30.80 / 0.947 / 0.086 / 67	31.63 / 0.954 / 0.072 / 87
	Average	29.65 / 0.904 / 0.143 / 87	29.60 / 0.899 / 0.169 / 78	29.56 / 0.903 / 0.157 / 64	29.81 / 0.908 / 0.137 / 86
Tanks & Temples [26]	Truck	25.43 / 0.865 / 0.167 / 69	25.39 / 0.859 / 0.217 / 53	25.86 / 0.869 / 0.193 / 38	25.09 / 0.865 / 0.162 / 55
	Train	21.95 / 0.809 / 0.217 / 77	22.29 / 0.802 / 0.253 / 55	22.35 / 0.822 / 0.228 / 40	21.84 / 0.827 / 0.204 / 58
	Average	23.69 / 0.837 / 0.192 / 73	23.84 / 0.831 / 0.235 / 54	24.11 / 0.846 / 0.211 / 39	23.47 / 0.846 / 0.183 / 56
Deep Blending [19]	Dr Johnson	29.01 / 0.884 / 0.274 / 127	28.75 / 0.886 / 0.282 / 77	28.83 / 0.882 / 0.285 / 104	28.65 / 0.887 / 0.268 / 113
	Playroom	30.50 / 0.887 / 0.247 / 127	29.32 / 0.880 / 0.279 / 83	30.22 / 0.890 / 0.272 / 105	30.11 / 0.886 / 0.248 / 112
	Average	29.76 / 0.886 / 0.261 / 127	29.04 / 0.883 / 0.281 / 80	29.53 / 0.886 / 0.279 / 104	29.38 / 0.887 / 0.258 / 112

Table A11. **Detailed results** on a selection of datasets and methods with the number of Gaussians limited to 2M. Each field contains PSNR, SSIM, LPIPS and FPS respectively. We highlight the best, second best and third best results among all. Slight discrepancies from the main table are due to rounding.