# Taming 3DGS: High-Quality Radiance Fields with Limited Resources

SASWAT SUBHAJYOTI MALLICK*, Carnegie Mellon University, USA
RAHUL GOEL*, Graz University of Technology, Austria and IIIT Hyderabad, India
BERNHARD KERBL, Carnegie Mellon University, USA
FRANCISCO VICENTE CARRASCO, Carnegie Mellon University, USA
MARKUS STEINBERGER, Graz University of Technology, Austria
FERNANDO DE LA TORRE, Carnegie Mellon University, USA

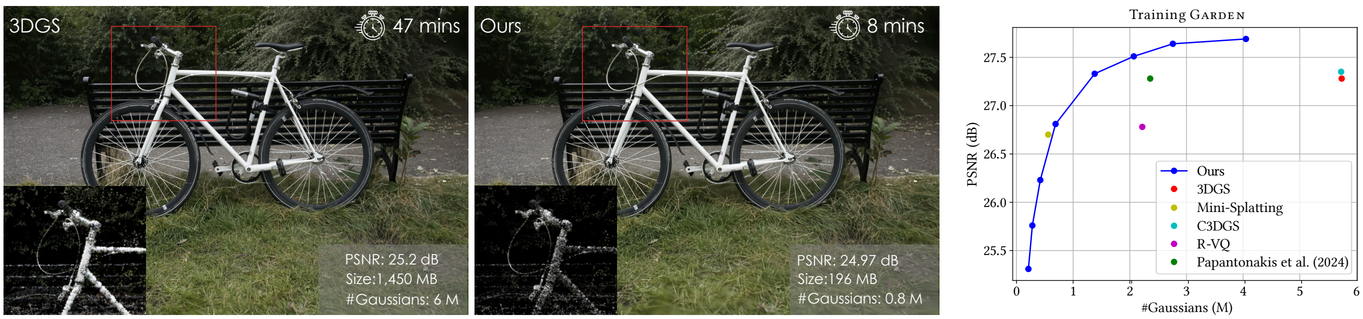



Fig. 1. Our method makes 3DGS optimization fast and flexible, achieving high rendering quality on a budget. Left: model size and training time are reduced by more than 5×. Right: Our method produces models with an exact, user-specified target size, surpassing 3DGS quality as the target increases.

3D Gaussian Splatting (3DGS) has transformed novel-view synthesis with its fast, interpretable, and high-fidelity rendering. However, its resource requirements limit its usability. Especially on constrained devices, training performance degrades quickly and often cannot complete due to excessive memory consumption of the model. The method converges with an indefinite number of Gaussians—many of them redundant—making rendering unnecessarily slow and preventing its usage in downstream tasks that expect fixed-size inputs. To address these issues, we tackle the challenges of training and rendering 3DGS models on a budget. We use a guided, purely constructive densification process that steers densification toward Gaussians that raise the reconstruction quality. Model size continuously increases in a controlled manner towards an exact budget, using score-based densification of Gaussians with training-time priors that measure their contribution. We further address training speed obstacles: following a careful analysis of 3DGS' original pipeline, we derive faster, numerically equivalent solutions for gradient computation and attribute updates, including an alternative parallelization for efficient backpropagation. We also propose quality-preserving approximations where suitable to reduce training time even further.

Taken together, these enhancements yield a robust, scalable solution with reduced training times, lower compute and memory requirements, and high quality. Our evaluation shows that in a budgeted setting, we obtain competitive quality metrics with 3DGS while achieving a 4–5× reduction in both model size and training time. With more generous budgets, our measured quality surpasses theirs. These advances open the door for novel-view synthesis in constrained environments, e.g., mobile devices.

*Both authors contributed equally to this research.

Authors' Contact Information: Saswat Subhajyoti Mallick, smallick@andrew.cmu.edu, Carnegie Mellon University, USA; Rahul Goel, goel@tugraz.at, Graz University of Technology, Austria and IIIT Hyderabad, India; Bernhard Kerbl, bkerbl@andrew.cmu.edu, Carnegie Mellon University, USA; Francisco Vicente Carrasco, fvicente@andrew.cmu.edu, Carnegie Mellon University, USA; Markus Steinberger, steinberger@tugraz.at, Graz University of Technology, Austria; Fernando de la Torre, ftorre@andrew.cmu.edu, Carnegie Mellon University, USA.

CCS Concepts: • **Computing methodologies → Rasterization**; **Reconstruction**; *Image-based rendering*; *Parallel algorithms.*

Additional Key Words and Phrases: Radiance Fields, Gaussian Splatting

## 1 INTRODUCTION

Novel View Synthesis (NVS) predicts unseen views from multi-view datasets, enabling users to freely explore 3D content from as little as a handful of easy-to-obtain photographs. State-of-the-art NVS solutions can yield photo-realistic results that produce high-quality user experiences for e-commerce, entertainment, and immersive telecommunication. Recently, NVS methods have also emerged as a powerful conditioning tool for high-quality 3D surface reconstruction. The extensive research body on NVS covers various methodologies, ranging from image- and mesh-based to purely neural representations. Within this domain, 3D Gaussian Splatting (3DGS) has been gaining popularity, since it combines high-quality image synthesis, fast real-time rendering, and amenable training times [Kerbl et al. 2023]. 3DGS leverages an explicit, point-based scene representation, a differentiable rendering pipeline, and GPU-optimized rasterization to achieve photo-realistic image synthesis at high frame rates. However, its optimization procedure is difficult to control; this process—although it includes several heuristics—is often wasteful and can lead to excessive memory overheads.

Starting from a sparse set of input points, many of the eventual optimized primitives are redundant: Gaussians often make only minor contributions in areas where fewer would suffice, while other regions remain under-reconstructed and blurry. This inefficient distribution of Gaussian primitives impacts not only training time but also the practical aspects of the representation. A typical 3DGS model can yield several millions of Gaussians for a single unbounded scene and require more than one gigabyte of disk space. Such substantial memory usage and geometry workload complicate real-time rendering on low-end devices, preventing application in constrained settings like network streaming or AR/VR on embedded systems.

In addition to being excessive, the memory consumption of 3DGS is also hard to predict: even when starting from the same number of input points, the difference between two reconstructed scenes w.r.t. the number of Gaussians (and thus required storage) can be as much as one order of magnitude. This hinders its usability for downstream applications with fixed input size (e.g., classifier networks), preventing them from using an otherwise efficient, explicit representation. Similarly, training time—although acceptable—fluctuates strongly and overall fails to reflect the much higher rendering speed of 3DGS.

In order to tame 3DGS, we propose a strict moderation in the Gaussian densification process to provide close control over its resource consumption (see Fig. 1). Given a user-defined model size, we ensure a deterministic training schedule that can yield the exact number of desired Gaussians. To achieve high quality with fewer primitives (4–5× on average), we tackle the suboptimal distribution and high redundancy of the original method. We propose an alternative densification algorithm, guided by a flexible, score-based sampling of Gaussian primitives. Our suggested scoring scheme for high quality at a budget combines loss-relevant components that we collect per Gaussian, and across multiple sampled training views. Densification occurs according to the pre-defined budget in the vicinity of the top-scoring Gaussians. In contrast to previous work, our densification uses a *purely constructive* schedule: we do not require substantial pruning or culling of Gaussians during training. Therefore, we avoid unnecessary peaks in the optimization that could violate the user's hardware or budget constraints. We acknowledge insightful concurrent work to ours on revising the densification in 3D Gaussian Splatting [Bulò et al. 2024].

Redundancy in 3DGS is not limited to its eventual primitive distribution. With this work, we seek to address the issues that make 3DGS hard to control. These challenges––unpredictable size and training duration, inability to guide reconstruction detail––require varied solutions, which eventually synergize to achieve our goal of easily controllable behavior. Therefore, we analyze the time cost and quality tradeoff for individual steps in the training pipeline and propose alternative, more efficient substitutes. This includes revisiting the parallelization opportunities of backpropagation, which we change from a per-pixel to a per-splat approach. Our contributions to taming 3DGS can thus be summarized as follows:

(1) *A purely constructive, budget-constrained optimization* for 3DGS, enabling full control over model size and resources.
(2) *A flexible framework for score-based densification*, allowing for use case-specific behavior and prioritization, e.g., by indicating important regions of interest.
(3) *Analysis and significant speedup of relevant training steps*, using both equivalent and approximate substitute methods.

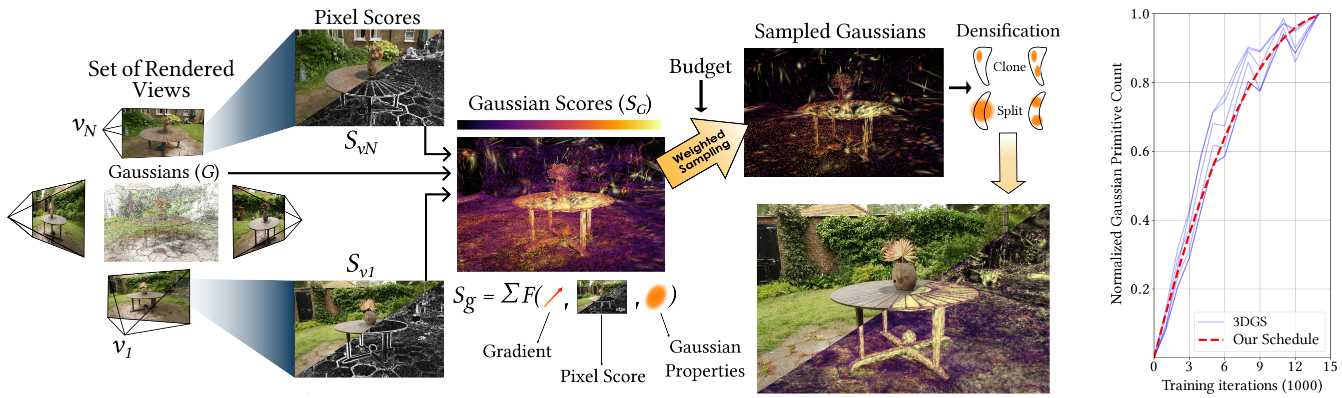
## 2 RELATED WORK

An extensive body of previous work focuses on novel-view synthesis: we first provide a brief overview of the most common approaches to this problem, before delving into solutions that focus specifically on raising the efficiency and portability of 3D Gaussian Splatting. Finally, we discuss point cloud downsampling approaches, from which we draw inspiration in our score-based densification.

*Novel-View Synthesis.* Previous work has explored a wide range of solutions for reconstructing or predicting the appearance of scenes, ranging from small-scale models [Buehler et al. 2023; Chaurasia et al. 2013; Jain et al. 2023] to unbounded environments [Bódis-Szomorú et al. 2016; Hedman et al. 2018; Riegler and Koltun 2021]. In contrast, Neural Radiance Fields (NeRFs) [Mildenhall et al. 2021] use an implicit representation, which is trained using gradient descent to recover a volumetric, continuous radiance field. While the initially proposed method was limited to single objects—taking over a day to process them—several follow-up works raised the scope and speed of NeRF scene reconstruction [Barron et al. 2021, 2022; Chen et al. 2022; Zhang et al. 2020]. To address the high rendering times, voxel-based representations [Karnewar et al. 2022; Sun et al. 2022] have been proposed to complement or replace selected components of the NeRF architecture. Significant breakthroughs for both training and rendering performance were marked by the use of hash grids [Müller et al. 2022] and space warping [Wang et al. 2023], at the cost of introducing quality caps. State-of-the-art NeRF-based techniques [Barron et al. 2023; Duckworth et al. 2023; Niemeyer et al. 2024; Wu et al. 2022; Zhang et al. 2022] are capable of reconstructing unbounded scenes with high quality and render at interactive frame rates, however, training them requires significant time and compute effort.

The recently introduced 3D Gaussian Splatting (3DGS) uses an initial point cloud—a common side product of calibration—and converts it to optimizable 3D Gaussian primitives [Kerbl et al. 2023]. 3DGS achieves high quality and extremely fast rendering; however, it suffers from exorbitant, unpredictable storage demands and fluctuating training times, making it a poor choice for performing novel-view synthesis *at a budget*.

*3D Gaussian Splatting and Compression.* Several recent works have managed to considerably reduce the on-disk storage requirements of 3DGS. Compressing a model's feature space is a widely adopted technique [Lee et al. 2023; Navaneet et al. 2023]; the parameters of the Gaussians (geometry, color, opacity) can be clustered and indexed using codebooks. This reduces the compute and storage footprint per primitive, alleviating total memory consumption without significant quality degradation. Niedermayr *et al.* [2023] follow a similar recipe, but use thorough, sensitivity-aware clustering on Gaussian parameters, followed by a quantization-aware fine-tuning and entropy encoding. Fan *et al.* [2023] weight Gaussians on their volume and opacity to prune the less significant ones, followed by distillation from synthetic (pseudo-)views and quantization of parameters. Papantonakis *et al.* [2024] cull Gaussian primitives based on their spatial density and adaptively prune view-dependent color coefficients on demand. While these methods are effective in reducing the storage requirements of 3DGS, they do little to make the process more *controllable.* Furthermore, although several approaches consider the decimation of Gaussian primitives, they usually cause modest reductions of ≈2×. Other aspects of previously proposed on-disk compression techniques, such as code-booking or entropy minimization, are directly compatible with our method, which would




(a) Score-based Sampling for Densification

(b) Predictable Growth

Fig. 2. Overview of our method. (a) We propose a systematic redesign of 3DGS densification. To select Gaussians to densify, we sample training views and compute per-pixel saliency. A scoring function $F$ combines gradient, saliency, and primitive properties into a per-Gaussian score $\mathbf{S}_g$. (b) The addition of new Gaussians follows a predictable schedule. We follow a growth curve that mimics 3DGS' behavior and can be fitted to yield any desired model size after training.

lead to even smaller file sizes due to our higher primitive reduction. Our solution approaches compact 3DGS from a different direction: while previous work identifies superfluous Gaussians for removal, our scoring instead guides *densification* directly. Furthermore, it is easy to compute from only image and per-Gaussian data, and flexibly supports various use cases, which we demonstrate.

*Point Cloud Downsampling.* By interpreting Gaussian means as singular points in space, we find that optimizing for high quality at low primitive counts is closely related to *point cloud downsampling*. Point clouds are 3D points distributed in space, often representing surfaces or the density of measured objects. Especially when resulting from real-world scanning, the considerable size of point cloud data can become a computation burden. This causes setbacks for downstream applications running on compute-constrained hardware settings. Previous work addresses this problem by quantizing the space and approximating samples using nearest neighbors [Goldberger et al. 2004; Plötz and Roth 2018], resampling points based on their density and distribution. Learning-based methods introduce task-specific sampling [Dovrat et al. 2019] and yield results competitive with heuristic methods, such as farthest point sampling. Nezhadarya *et al.* [2020] uses a critical points layer, which qualifies the most significant points to the next network layer.

Yang *et al.* [2019] implement Gumbel subset sampling to improve the classification accuracy of a network trained on point cloud data. Lang *et al.* [2020] introduce a differentiable projection during nearest-neighbor search that "softens" the discrete points. Inspired by these sampling-based methods to produce compact, salient models, we revise 3DGS densification as a sampling-guided procedure.

## 3 METHOD

Our approach is outlined in Fig. 2a. SfM point clouds are used as an initialization to train a 3DGS-based model from calibrated multi-view images with a pre-determined densification schedule. The original 3DGS densification algorithm continuously adds primitives (details) to regions with high positional gradients, splitting large Gaussians, cloning smaller ones, and removing transparent ones. We replace this module with a less frequently executed procedure built upon steerable sampling. The maximum number of new Gaussians added at every stage is pre-determined: although our method mimics the original 3DGS growth curve, the peak (and final) number of Gaussians is fully controllable by the user who provides the limits for model size (Fig. 2b). Crucially, this constructive approach avoids temporary spikes in model size which are usually observed in previous work [Fang and Wang 2024]. Intuitively, closely following a constructive schedule avoids oscillation around a target budget and thus unpredictable behavior, which is our key goal.

To maximize the quality per Gaussian, our densification is guided using a score-based ranking and employs *high-opacity Gaussians* to increase the primitives' expressiveness. In addition, training duration is significantly reduced through several proposed modifications that target the primary bottlenecks of the original pipeline, including a faster, numerically equivalent solution for backpropagation. Taken together, these measures yield an optimization with high controllability, flexibility, and performance.

### 3.1 3D Gaussian Splatting Background

3DGS [Kerbl et al. 2023] is a point-based approach that models scenes using a set of 3D Gaussians, parameterized by position ($\mu$), covariance ($\Sigma$), and opacity $o$. Ignoring inter-primitive overlap, the theoretical contribution of a 3D Gaussian at a point $x$ is defined by:

$$G(x) = oe^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}, \quad \Sigma = RSS^TR^T, \tag{1}$$

where $R$ is a rotation and $S$ a scaling matrix. View-dependent appearance is modeled by Spherical Harmonics (SH) of order 3 and a direct color component for base appearance. For a particular viewpoint, the visible set of 3D Gaussians is rendered in a tile-based, differentiable rasterizer to obtain a 2D image by $\alpha$-blending their projections (splats). 3DGS training minimizes a combined $L_1$ and SSIM loss w.r.t. the rendered and ground truth image by optimizing the parameters—position, rotation, scaling, opacity, and SH—of each Gaussian.

### 3.2 Predictable Model Growth

Throughout optimization, 3DGS continuously *densifies* its representation by adding Gaussian primitives to resolve under-reconstructed regions. However, the number of added primitives at each stage is decided based on a simple thresholding operation, with no control over the progressive or final count. This evolutionary automaton—although effective—leads to hard-to-predict, often exorbitant model sizes and fluctuating training times.

To define a simpler, yet similarly effective and fully predictable growth pattern, we investigate the densification behavior of 3DGS across the outdoor scenes in the MipNeRF360 dataset. Fig. 2b plots the development in the number of total Gaussians for each scene as training progresses with the original method; note that curves have been renormalized on the range between their initial and final 3DGS primitive count. We find that the number of Gaussians added in each step follows a trend of quadratic decrease. We exploit this pattern to determine a schedule of added primitives at each step, using a parabolic curve that starts from the SfM initialization and peaks precisely at the *user-defined budget*:

$$A(x) = \frac{B - S - 2N}{N^2} x^2 + 2x + B, \tag{2}$$

where $N$ is the number of densification steps, $B$ is the final count (budget), and $S$ is the number of SfM points at initialization. As in 3DGS, we prune low-opacity Gaussians over time, thus following an additive schedule directly may produce fewer primitives than the given target. To avoid this, we instead compute the difference between our current and *accumulated* target count and densify the corresponding number of primitives. Sec. 5 demonstrates the effectiveness of this scheme and the graceful quality degradation resulting from lower budget limits.

### 3.3 Steerable Densification with Sampling

The original 3DGS approach suggests that high positional gradients on a Gaussian indicate insufficient samples in its vicinity. Hence, such Gaussians are regularly densified, either by *cloning* or *splitting* (depending on their size). Bleeding-edge research reformulated the 3DGS optimization process as a sequence of Stochastic Langevin Gradient Descent (SLGD) updates [Kheradmand et al. 2024]. At any point, the optimized set of Gaussians can be interpreted as samples from a likelihood distribution tied to 3DGS' overall loss. Obtaining a complete, high-fidelity reconstruction demands a solution that delicately balances optimization and exploration. Letting image loss also steer the densification procedure seems intuitive: a high loss can indicate the need for denser sampling or additional exploration.

In the spirit of maintaining a steerable, yet interpretable densification procedure, we propose a flexible solution that incorporates salient indicators like image loss directly into the process. This is enabled via two key features: a score-based, customizable sampling of densification candidates and a significantly reduced densification frequency. The former combines salient per-Gaussian and per-pixel metrics, such as loss, to decide each primitive's probability of densification. The reduction in densification frequency is motivated by the interplay of loss, sample placement, and optimization. A Gaussian will cause high image loss for two reasons: either its neighborhood is insufficiently sampled, or it has been erroneously placed. When using loss for guidance, frequent densification can thus cause repeated duplication of misplaced Gaussians. However, when given sufficient time and observations, 3DGS will eliminate out-of-place Gaussians by lowering their opacity before densification occurs.

We invoke densification at a frequency of only one-fifth of 3DGS (i.e., every 500 iterations). Given a set of $N$ camera views, $V = \{v_i\}_{i=1}^N$, the set of $M$ fitted Gaussians, $G = \{g_j\}_{j=1}^M$, and the set of $N$ rendered views, $R = \{r_i\}_{i=1}^N$, we evaluate a scoring function $F$ that is parameterized by per-Gaussian primitive attributes and projected per-pixel metrics. This involves the following:

(1) **Determine per-view saliency matrix $\mathbf{S}_v$**: For each view $v$, this matrix indicates pixels that may be undersampled (high loss) or contain high-frequency information. Additionally, this function enables prioritizing regions of interest:

$$\mathbf{S}_v = \mathbb{1}_{ROI} \odot (\lambda_1 \mathcal{L}_1(v, r_v) + \lambda_2 E(v)), \quad v \in V \tag{3}$$

where $\mathcal{L}_1$ is the L1 loss, $E$ is a Laplacian filter, $\mathbb{1}_{ROI}$ is a binary matrix (each element $\in \{0, 1\}$) indicating a masked region of interest, $\odot$ is the element-wise product, and $\lambda_1, \lambda_2$ are hyperparameters, set to 0.5 in our experiments.

(2) **Compute Gaussian scores $\mathbf{S}_G$**: We compute a global score vector $\mathbf{S}_G$ that holds a score $S_g$ for each Gaussian $g$ in $G$. We do this by evaluating $F(\cdot)$ and summing over all $N$ views:

$$S_g = \sum_i^N F(\nabla_g, c_g^i, \mathbb{1}_g^i, \mathbf{D}_g^i, \mathbf{S}_v^i, \mathbf{B}_g^i, z_g^i, o_g, s_g) \tag{4}$$

$$\mathbf{S}_G = [S_{g_1}, ..., S_{g_M}]^T, \quad g_j \in G \tag{5}$$

Here, $\nabla_g$ is the Gaussian's positional gradient. $c_g^i$ denotes the number of pixels covered by $g$ in view $i$. $\mathbb{1}_g^i$ is a binary matrix that indicates these pixels. $\mathbf{D}_g^i$ is a matrix that holds the distance of each pixel to the center of $g$. $\mathbf{B}_g^i$ contains each pixel's blending weight for $g$. Attributes $z_g^i$, $o_g$, and $s_g$ constitute the depth in $i$, opacity, and scale of $g$, respectively.

$\mathbf{S}_G$ is representative of the need to resample each Gaussian to converge to the final scene and serves as the foundation of our score-based densification. Alg. 1 provides more details on this process. For the choice of $F$, we restrict each parameter's range using median scaling to remove outliers, followed by multiplication with $i$'s photometric loss. The so-rescaled parameters are then accumulated into a weighted sum, whose coefficients can be tuned for specific use cases. In the following, we explain the role of each parameter (and our proposed weighting) to achieve high quality with few Gaussians.

$\nabla_g$ (50): We adopt the magnitude of the positional gradient as a criterion from [Kerbl et al. 2023] *et al.* According to the authors, high $\|\nabla_g\|_2$ can be interpreted as a 3D discontinuity detector. While provably effective, it alone usually leads to wasteful behavior and superfluous Gaussians.

$c_g^i$ (0.1): The pixel count of $g$ acts as an indicator for primitives that tend to have large projections, which lead to a blurry appearance in rendered images. Recent work uses similar indicators to guide Gaussian growth [Zhang et al. 2024].

$\mathbf{D}_g^i$ (50): Splats that cover only a few pixels may still appear as thin elongated "slivers" on screen. We encourage their densification by scoring the sum of distances of covered pixels to the center of $g$.

$\mathbf{S}_v^i$ (10): We weight the accumulated per-pixel saliency scores of pixels covered by $g$ (i.e., sum of element-wise products with $\mathbb{1}_g^i$). This enables the previously computed saliencey to guide densification.

$\mathbf{B}_g^i$ (50): The sum of per-pixel blending weights used in rendering indicates high-contributing Gaussians. Densifying them has the highest chance of causing changes in scene appearance and quality.

$z_g^i$ (5): The depth of each Gaussian allows us to distinguish between foreground and background. Note that this value is 0 for all Gaussians outside the view frustum: thus, it serves as a combined measurement of $g$'s visibility in the capture and its average distance to the camera. This prioritizes densifying commonly seen primitives without neglecting rarely seen background Gaussians.

$o_g$ (100): We use a high weight on opacity to steer densification away from low-opacity Gaussians. Low opacity is characteristic of floaters, or Gaussians that the optimization is currently phasing out.

$s_g$ (25): Overly large Gaussians—even if not observed up close during training—hurt generalizability to unseen views. Scoring the product of Gaussians' scales yields more uniformly sized primitives.

Given the final score vector $\mathbf{S}_G$ and a budgeted target number $B$ of Gaussians to add, we perform densification by randomly resampling $B$ primitives from all Gaussians using $\mathbf{S}_G$ as sampling weights. In practice and for all experiments, we use $N = 10$ uniformly sampled training views for computing per-pixel saliency scores. Regarding runtime complexity, our scoring adds an extra $O(N \times \texttt{width} \times \texttt{height})$ step for computing all $N$ $\mathbf{S}_v$, each of which is propagated to the Gaussians with an auxiliary rendering pass to obtain $\mathbf{S}_G$.

### 3.4 High-Opacity Gaussians

While the basic Gaussian primitives of 3DGS can yield high quality, their expressiveness is limited by their rigid Gaussian falloff [Hamdi et al. 2024]. To remedy this, Kerbl et al. [Kerbl et al. 2024] used simple clamped Gaussians with opacities >1 to approximate the appearance of Gaussian clusters in a hierarchical level-of-detail structure. We find that these high-opacity Gaussians can also boost the ability for modeling opaque surfaces with a low number of primitives. Therefore, we convert the regular, capped Gaussian primitives to high-opacity Gaussians after reaching the midpoint of our training (15K iterations). This involves replacing the opacity activation with `abs` and clamping blending weights to 1 from above during rendering. As shown by our ablation, this change positively impacts quality metrics, particularly PSNR.

## 4 3DGS RUNTIME ANALYSIS AND OPTIMIZATION

To better understand the performance challenges of 3DGS, we benchmark the original training pipeline, written in PyTorch, with explicit CUDA extensions for differentiable rasterization. We provide a breakdown of the time taken by the high-level steps in each iteration for multiple scenes, at different stages of training, in Fig. 3. We note that, throughout the training routine, backpropagation of gradients is the dominating bottleneck, closely followed by Adam optimizer updates as the number of Gaussians increases. With these insights, we propose targeted solutions for accelerating 3DGS training.

**Algorithm 1** Proposed steerable densification method

```
T ← Target Gaussian count at current iteration
𝒢 ← All Gaussians {g_1, g_2, ..., g_|G|}
𝒢_t ← Gradient threshold
ℛ_t ← Radius threshold
for image i ∈ sampled views(N) do
    P_i ← Photometric loss
    Initialise: c_g^i = 0; D_g^i = 0; s_g^i = 0; B_g^i = 0
    for pixel p ∈ i do
        for g ∈ Gaussians contributing to p do
            c_g^i += 1
            D_g^i += Distance from center of g to p
            s_g^i += S_v^i(p)
            B_g^i += Blending weight of g on p
        end for
    end for
    S_g = S_g + P_i · F(∇_g, c_g^i, D_g^i, s_g^i, B_g^i, z_g^i, o_g, s_g)
end for
S_G = [S_g1, ..., S_gM]^T,    g_i ∈ 𝒢
B = T − |𝒢|                                    ▷ #Gaussians to add
Top gaussian indices: G′ ~ (𝒢, S_G, B)
for g ∈ G′ do
    if (∇_g > 𝒢_t) & (radius_g > ℛ_t) then
        SPLIT
    else if (∇_g > 𝒢_t) & (radius_g ≤ ℛ_t) then
        CLONE
    end if
end for
```

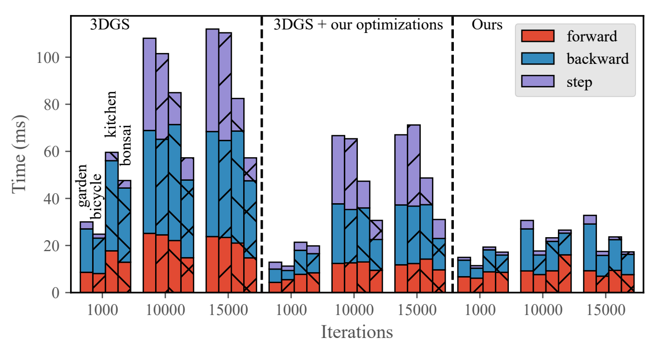



Fig. 3. Time spent in different parts (forward pass, backward pass, optimizer step) of one 3DGS iteration in four scenes (GARDEN, BICYCLE, KITCHEN, BONSAI). Left: analysis of original 3DGS at different stages of training. Center: original 3DGS densification with our proposed performance optimizations. Right: using our compact densification and performance optimizations.

### 4.1 Backpropagation with Per-Splat Parallelization

In the original 3DGS backward pass, gradients are propagated from the pixels onto the Gaussians. The total gradient calculation involves computing many per-pixel, per-splat values, which are then accumulated globally via reduction. Kerbl et al. [2023] take the natural approach of mapping threads to pixels and iterating over the depth-sorted splats back-to-front. Within a tile, each thread considers splats in reverse blending order, evaluates a per-pixel gradient portion, and atomically adds it to the corresponding splat's accumulated gradient. While correct, this leads to multiple threads contending for access to the same locations and thus serialized atomic operations, as shown in Fig. 4. The fact that each Gaussian splat maintains *a multitude* of gradients for its attributes further aggravates the overhead of this reduction [Durvasula et al. 2023].

We propose a solution where each tile uses a parallelization scheme over the 2D *splats* instead of pixels. This new approach lets threads maintain a per-splat state and continually exchange per-pixel states consisting of transmittance $T$ and accumulated color $RGB$ (as opposed to storing per-pixel information and exchanging the larger per-splat data). Ignoring corner cases, let us assume a simplified setting where #threads = #pixels = #splats = N. At each point in time, thread $i$ computes a gradient portion for splat $i$; to do this, it requires the state of each pixel $j$ after blending the front-most $i$ primitives. During the forward pass, each thread stores one per-pixel state every N splats in the autodiff context for backward, resulting in available starting states $(0, j), (N, j), \ldots \forall j$. From these, each thread in a tile generates pixel state $(i, j)$ at the beginning of the backward pass. Threads then exchange pixel states via fast collaborative sharing. In each step, thread $i+1$ applies the default alpha blending logic to go from its received $(i, j)$ to $(i+1, j)$ and incorporates this information into the gradient. For more details please refer to Fig. 4 and accompanying video.

We also observe that iterating the tail of each tile's depth-sorted list of splats often becomes redundant due to occlusion. This is avoided in the forward pass, which terminates upon saturation. To exploit this in backpropagation as well, we keep track of the last contributor across the tile and use it to skip entire groups of splat $\iff$ tile pairings. Finally, we reduce the overall rasterization workload via tighter culling as proposed by Radl et al. [2024], minimizing redundant splats in the forward and backward pass.

Fig. 5 compares the time taken for the backward methods of 3DGS, concurrent work DISTWAR [Durvasula et al. 2023] and Ours, with the original 3DGS and our compact optimization schedule.

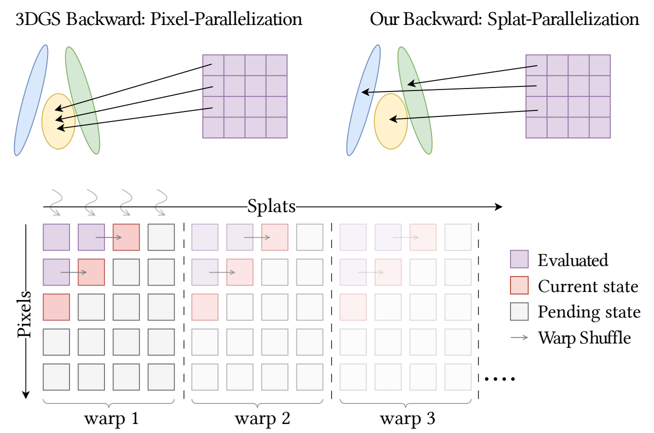



Fig. 4. Gradient backpropagation. (Top) 3DGS utilizes per-pixel parallelization for backpropagation. Atomic gradient additions create frequent collisions, slowing down the backward. Instead, we parallelize on the projected 2D splats, such that each thread (and pixel) contributes to one Gaussian at a time. (Bottom) The gradient calculation requires processing a set of per-pixel, per-splat values resulting in an implicit traversal of a splat $\iff$ pixel state table. During the forward pass, we store the pixel states for every 32$^{nd}$ splat in the per-tile sorted lists. For the backward, we divide the splats into buckets of size 32, each of which gets scheduled to a CUDA warp. Warps use intra-warp shuffling to produce their share of the state table cheaply.

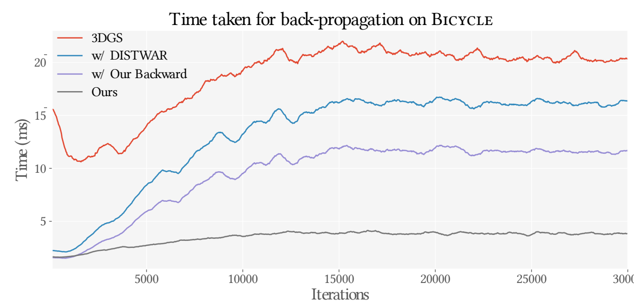



Fig. 5. Backward pass duration in training of Bicycle using 3DGS, DISTWAR [Durvasula et al. 2023] and our variants. For our approach, we plot the times when we used 3DGS densification and our proposed budget schedule.

## 4.2 Accelerated SH and Differentiable Loss Computation

Fig. 3 reveals the significant time spent on Adam updates as the number of Gaussians increases. Of these updates, SHs—48 out of 59 optimized per-Gaussian attributes—are responsible for the vast majority. To amend this, we switch all bands beyond the first to a batched update schedule, performing only one step of Adam optimization every 16 iterations.

The original 3DGS implementation combines the 0th SH band (i.e., base color) and higher bands into a single tensor before rasterization. This consumes a surprising portion of the forward pass. We avoid this by extending the differential rasterizer to load Gaussian SH coefficients from separate tensors.

3DGS loss computation involves evaluating the SSIM metric. It is configured to use 11×11 Gaussian kernel convolution: we propose using optimized CUDA kernels to perform differentiable 2D convolution via two consecutive 1D convolutions, since Gaussian kernels are separable in nature. In addition, we use a fused kernel for the evaluation of the SSIM metric from the convolved results. This speeds up the loss calculation and is particularly impactful when the number of optimized Gaussians is low compared to image resolution, which is the case when training on a budget.

## 5 EVALUATION AND DISCUSSION

This section evaluates our proposed approach both quantitatively and qualitatively. Our implementation is based on top of the original 3DGS codebase [Kerbl et al. 2023]. Most original hyperparameters are retained; however, we added a separate Adam optimizer for batched SH updates, increasing the SH learning rate four times (0.001) and reducing the opacity learning rate by half (0.025). The evaluation was conducted using an NVIDIA RTX A4500 GPU. Results for other techniques, including training times, were obtained on the same hardware or adjusted to ensure comparability.

## 5.1 Datasets and Metrics

We run benchmarks on three established datasets—Tanks&Temples [Knapitsch et al. 2017], Deep Blending [Hedman et al. 2018], and MipNeRF360 [Barron et al. 2022], which contain 2, 2, and 9 scenes, respectively. These datasets cover bounded indoor and unbounded outdoor scenarios with detailed backgrounds. We use the same train/test split as the original 3DGS publication and follow-up work.

In addition to common quality metrics (peak signal-to-noise ratio (PSNR), structural similarity (SSIM), and perceptual similarity (LPIPS) [Zhang et al. 2018], an important focus of our work is resource efficiency: Our method aims to achieve high quality with low resource usage. We assess these qualities by timing the optimization (Train time), counting the final number of Gaussians (#G), as well as recording the *peak* number (Peak #G) during training.

## 5.2 Results

We evaluate our method in two separate, budgeted scenarios. Results for the first scenario are shown at the top of Table 1, and those for the second scenario at the bottom. For qualitative results, see Fig. 6.

In the first, we select a reasonable budget for individual scenes, based on their spatial extent and SfM point count. For the small-scale indoor scenes in MipNeRF-360, we set the budget to 2× the SfM points. For the larger, full-room indoor captures of Deep Blending, we use 5×, and for unbounded outdoor scenes, we use 15×. For the outdoor Tanks&Temples, the initial SfM point count is significantly higher, thus we set the budget to 2× here as well. Note that this parameterization could be automatized by providing scenes in real-world coordinates or a corresponding multiplier. To evaluate the resources/quality tradeoff, we compare with recent works that aim at reducing the memory footprint of 3DGS: (Compressed 3DGS) [Niedermayr et al. 2023], Compact-3DGS (R-VQ) [Lee et al. 2023], and [Papantonakis et al. 2024]. Due to its exceptionally fast training, we also compare with the high-quality version of Instant-NGP (INGP-Big) [Müller et al. 2022]. To perform a thorough evaluation and provide comprehensive context, we also evaluate the concurrent work for Mini-Splatting [Fang and Wang 2024]. Assessing the results in the top half of Table 1, we find that among splatting-based methods, Ours achieves outstanding reduction (slightly outperformed only by Mini-Splatting in one dataset). Notably, our compact method is competitive with (and sometimes surpasses) 3DGS in terms of quality, especially PSNR. However, the most striking benefit of our approach is efficiency: Mini-Splatting—similar to 3DGS—relies on heavily oversampling the scene before pruning, creating a vast gap of up to 10× between their peak and final model size. In contrast, our method uses a purely constructive optimization that only adds Gaussians towards an exact target budget. In addition, we achieve this using between half and one-third of the time of the next-fastest 3DGS-based methods and occasionally outperform even Instant-NGP in terms of speed. The reduction of primitives naturally leads to an accelerated rendering performance. Average achieved frames per second are significantly higher using Ours (compared with 3DGS): Tanks&Temples 246 FPS (vs. 127), MipNeRF-360 142 FPS (86), and Deep Blending 258 FPS (92).

In the second budgeted scenario, we configure our optimization to reach the exact same model size as the original 3DGS. Since the expressiveness of our method rises with the available budget, in this scenario, we compare our results with representative, high-quality approaches from different domains: Plenoxels [Fridovich-Keil et al. 2022], and two sophisticated NeRF methods, MipNeRF360 [Barron et al. 2022] and ZipNeRF [Barron et al. 2023]. Finally, we consider the original 3DGS technique [Kerbl et al. 2023]. We provide the corresponding results in the bottom half of Table 1. Although our optimization differs significantly from 3DGS, we demonstrate that our budgeting mechanism allows Ours (Big) to match their model size exactly. The achieved quality easily surpasses 3DGS and MipNeRF-360, second only to the recent, much slower Zip-NeRF approach.

## 5.3 Ablations

Table 2 examines the effect of individually removing several of our contributions. This analysis is performed in the first budgeted scenario. Note that all configurations yield the same number of Gaussians. However, omitting the consideration of image loss (or our score-based sampling altogether) from densification significantly harms quality. We observe a similar impact when omitting the use of high-opacity Gaussians. Reverting to the original SH update frequency can lead to minuscule quality improvements, but causes a performance drop of up to 25%. Early results have shown that an even better speed/quality tradeoff may be achieved with an alternative "sparse" Adam optimizer that only applies gradient updates to attributes with non-zero gradients. Replacing our per-splat backward pass with the original has an even higher performance cost, indicating the effectiveness of our optimizations. Table 3 assesses the impact of alternative growth curves, showing the relative optimality of choosing the quadratic curve. To quantify the impact and importance of the factors included in our score function, we include an ablation of their average effect on scene quality. Starting from our full scoring function, removing individual components, in order of severity, incurs PSNR penalties: blending weights (-1.31 dB), Laplacian filter (-1.29 dB), pixel coverage (-1.28 dB), pixel distance (-1.26 dB), depth (-1.26 dB), scale (-1.26 dB), positional gradient (-1.25 dB), opacity (-1.25 dB), saliency (-1.24 dB), and L1 (-0.22 dB).

As an additional case study, Fig. 1 ablates the quantitative effect on Garden when varying the available budget. We see a consistent improvement as budget increases, showing a clear correlation between provided budget and achieved image quality. While our approach does not target the peculiarities of PyTorch, we note that our first budgeted scenario allows training with consistently less than 10 GB VRAM—compact enough for a mid-range NVIDIA RTX 3080. Table 4 lists both required training time and quality as the number of Gaussians (budget) is modified. We observe a linear relation of budget and training time, but a flattening curve in quality as the expressiveness of the model saturates.

To evaluate the impact and robustness of terms in the scoring function, we tested for a variance-based sensitivity (Sobol method). We found that within 10% variance of the sensitivity parameters, our first and total order sensitivities are within the range 0.01–0.05,

Table 1. Quantitative comparison of other methods with our technique in two budgeted scenarios (top half: compact models, bottom half: match 3DGS size). For quality, we compare PSNR, SSIM, and LPIPS metrics. For resource efficiency, we report training time, and, where applicable, the final number (#G), and peak number (Peak #G) of Gaussians used. Best and Second Best results are highlighted for each dataset and category.

| | Tanks&Temples | | | | | | MipNeRF-360 | | | | | | Deep Blending | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SSIM | PSNR | LPIPS | Train time | #G ($10^6$) | Peak #G | SSIM | PSNR | LPIPS | Train time | #G ($10^6$) | Peak #G | SSIM | PSNR | LPIPS | Train time | #G ($10^6$) | Peak #G |
| INGP-Big | 0.745 | 21.92 | 0.305 | 7 m | - | - | 0.699 | 25.59 | 0.331 | 8 m | - | - | 0.817 | 24.96 | 0.390 | 8 m | - | - |
| C3DGS | 0.843 | 23.57 | 0.182 | 28 m | 1.53 | 1.84 | 0.811 | 27.34 | 0.221 | 43 m | 2.44 | 2.94 | 0.900 | 29.54 | 0.252 | 39 m | 2.43 | 2.81 |
| RVQ | 0.831 | 23.30 | 0.202 | 27 m | 0.83 | 1.46 | 0.797 | 26.99 | 0.245 | 48 m | 1.41 | 2.57 | 0.901 | 29.75 | 0.260 | 38 m | 1.04 | 2.25 |
| [Papantonakis et al. 2024] | 0.844 | 23.66 | 0.178 | 18 m | 0.71 | 0.71 | 0.814 | 27.43 | 0.220 | 25 m | 0.83 | 0.83 | 0.902 | 29.57 | 0.247 | 22 m | 0.97 | 0.97 |
| Mini-Splatting | 0.847 | 23.42 | 0.181 | 20 m | 0.31 | 4.32 | 0.822 | 27.26 | 0.217 | 30 m | 0.49 | 4.32 | 0.909 | 30.04 | 0.244 | 24 m | 0.56 | 4.51 |
| Ours | 0.835 | 23.89 | 0.207 | 5 m | 0.29 | 0.29 | 0.799 | 27.29 | 0.253 | 8 m | 0.63 | 0.63 | 0.902 | 29.89 | 0.263 | 5 m | 0.27 | 0.27 |
| Plenoxels | 0.719 | 21.08 | 0.379 | 25 m | - | - | 0.626 | 23.08 | 0.463 | 26 m | - | - | 0.795 | 23.06 | 0.51 | 28 m | - | - |
| MipNeRF360 | 0.759 | 22.22 | 0.257 | 48 h | - | - | 0.792 | 27.69 | 0.237 | 48 h | - | - | 0.901 | 29.4 | 0.245 | 48 h | - | - |
| Zip-NeRF | - | - | - | - | - | - | 0.828 | 28.54 | 0.189 | 1.5 h | - | - | - | - | - | - | - | - |
| 3DGS | 0.847 | 23.65 | 0.176 | 22 m | 1.84 | 1.84 | 0.815 | 27.46 | 0.215 | 33 m | 3.31 | 3.31 | 0.904 | 29.64 | 0.243 | 34 m | 2.81 | 2.81 |
| Ours (Big) | 0.851 | 24.04 | 0.170 | 10 m | 1.84 | 1.84 | 0.822 | 27.79 | 0.205 | 16 m | 3.31 | 3.31 | 0.907 | 30.14 | 0.235 | 13 m | 2.81 | 2.81 |

Table 2. Ablations of our method's components on all datasets in the first budgeted scenario.

| | Tanks&Temples | | | | MipNeRF-360 | | | | Deep Blending | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SSIM | PSNR | LPIPS | Time | SSIM | PSNR | LPIPS | Time | SSIM | PSNR | LPIPS | Time |
| Ours | 0.835 | 23.89 | 0.207 | 5 m | 0.799 | 27.29 | 0.253 | 8 m | 0.902 | 29.89 | 0.263 | 5 m |
| No Score-Based Sampling | 0.829 | 23.61 | 0.222 | 4 m | 0.762 | 26.69 | 0.292 | 8 m | 0.899 | 29.78 | 0.276 | 4 m |
| No Image Loss | 0.828 | 23.47 | 0.224 | 5 m | 0.774 | 26.65 | 0.274 | 8 m | 0.884 | 29.14 | 0.281 | 5 m |
| No High-Opacity Gaussians | 0.813 | 23.65 | 0.221 | 5 m | 0.779 | 26.84 | 0.277 | 8 m | 0.876 | 28.92 | 0.286 | 5 m |
| No Reduction in SH Updates | 0.837 | 23.94 | 0.201 | 6 m | 0.803 | 27.37 | 0.249 | 10 m | 0.905 | 29.91 | 0.258 | 6 m |
| No Per-Splat Backward | 0.835 | 23.89 | 0.207 | 9 m | 0.799 | 27.29 | 0.253 | 18 m | 0.902 | 29.89 | 0.263 | 11 m |

Table 3. Achieved metrics with different training schedule curves.

| | Tanks&Temples | | | MipNeRF-360 | | | Deep Blending | | |
|---|---|---|---|---|---|---|---|---|---|
| | SSIM | PSNR | LPIPS | SSIM | PSNR | LPIPS | SSIM | PSNR | LPIPS |
| Quadratic | 0.835 | 23.89 | 0.207 | 0.799 | 27.29 | 0.253 | 0.902 | 29.89 | 0.263 |
| Linear | 0.832 | 23.77 | 0.214 | 0.794 | 27.21 | 0.261 | 0.898 | 29.65 | 0.275 |
| Exponential | 0.831 | 23.75 | 0.216 | 0.788 | 27.09 | 0.270 | 0.897 | 29.63 | 0.277 |

Table 4. Measurement of quality (PSNR) development with the number of Gaussians (M) and training time taken (in minutes) for GARDEN.

| Budget (M) | 0.7 | 1.4 | 2.1 | 2.8 | 3.5 | 4.2 | 4.9 |
|---|---|---|---|---|---|---|---|
| PSNR (dB) | 26.74 | 27.24 | 27.39 | 27.44 | 27.53 | 27.59 | 27.64 |
| Time (m) | 6 | 9 | 11 | 13 | 15 | 18 | 20 |

which proves that the method is robust across minor hyperparameter variances. When parameters were changed by 20%, first-order Sobol indices measured up to 0.16, showing the limit of robustness.

Finally, we show the flexibility of our cost function on two use cases: one prioritizing regions of interest (Fig. 7), the other producing superior results in single-object foreground reconstruction, by encouraging densification of low-depth Gaussians (Fig. 8).

## 6 CONCLUSION

We have presented an efficient and controllable splatting-based optimization technique for generating high-quality radiance fields. Our approach restrains the unpredictable behavior of the recent 3DGS technique, allowing for exact primitive budgeting, flexible sample guiding, and highly improved resource efficiency, avoiding excessive peaks in training.

These properties generate new opportunities for optimizing novel-view synthesis in various environments, e.g., hardware-constrained and edge devices. Other applications include latency-constrained streaming services, where on-the-fly, interactive 3D reconstructions could be steered towards prioritizing salient regions of interest, such as faces. Another, exciting avenue that our sample guiding could facilitate is 3DGS optimization using pre-trained generative models, e.g., by steering optimization based on score distillation [Chen et al. 2024]. Our contributions are complementary to ongoing 3DGS compression efforts, many of which could be applied to our reduced-size models to even greater effect.

While our approach is an important step towards low-cost, high-quality radiance fields, achieving optimal quality still requires a substantial sample count. Strongly budgeted settings can lead to noticeable blurriness, especially for background objects (see accompanying video). We leave more elaborate predictions and resolution of blind spots in scene exploration to future work. Our code is available at https://github.com/humansensinglab/taming-3dgs.

## ACKNOWLEDGMENTS

We thank Srinath R. and Junkai Huang for helpful discussions.

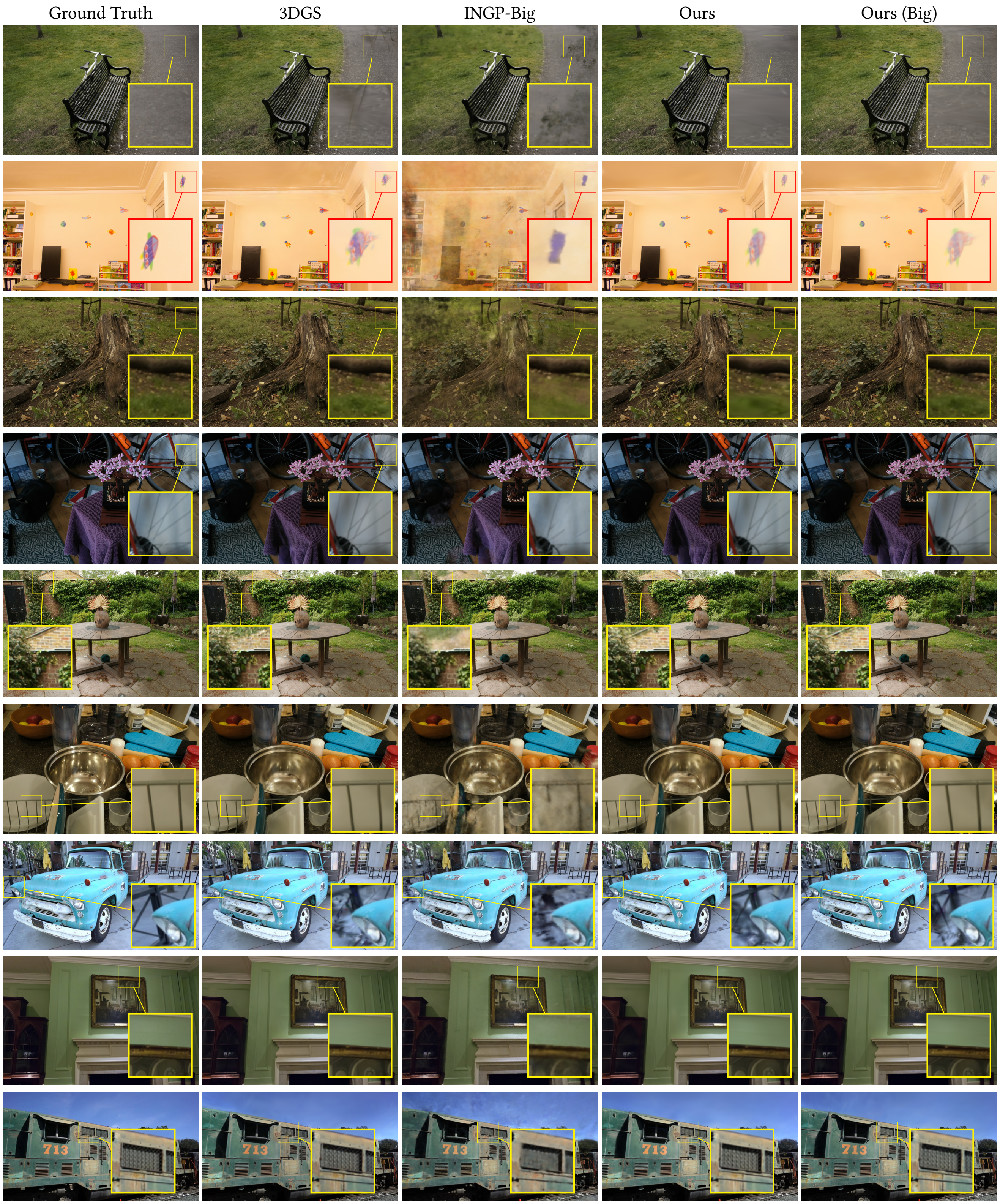



Fig. 6. Qualitative comparison of results produced with our method in two budgeted scenarios to 3DGS, as well as Instant-NGP, whose training times match those of Ours. While the strictly budgeted scenario produces highly competitive results, a higher budget resolves occasional remaining blurry Gaussians.

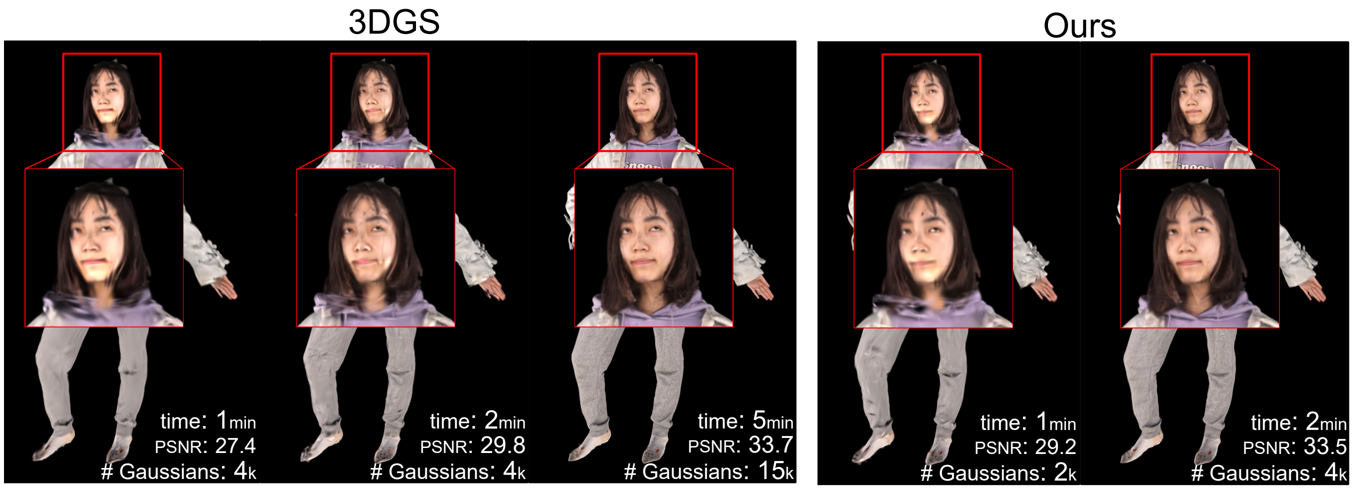



Fig. 7. Demonstrating prioritization for guiding densification to regions of interest. We assign higher scores to face masks detected with SegmentAnything [Kirillov et al. 2023] in the computation of $S_v$. The above figure displays the quality of the facial region as measured via PSNR. We achieve competitive metrics faster and with fewer Gaussians than 3DGS. This demonstrates use cases of our approach for latency-constrained live settings. In a telepresence scenario, we could prioritize the quality of the most frequently observed image regions and leave others under-sampled without impacting the user experience.

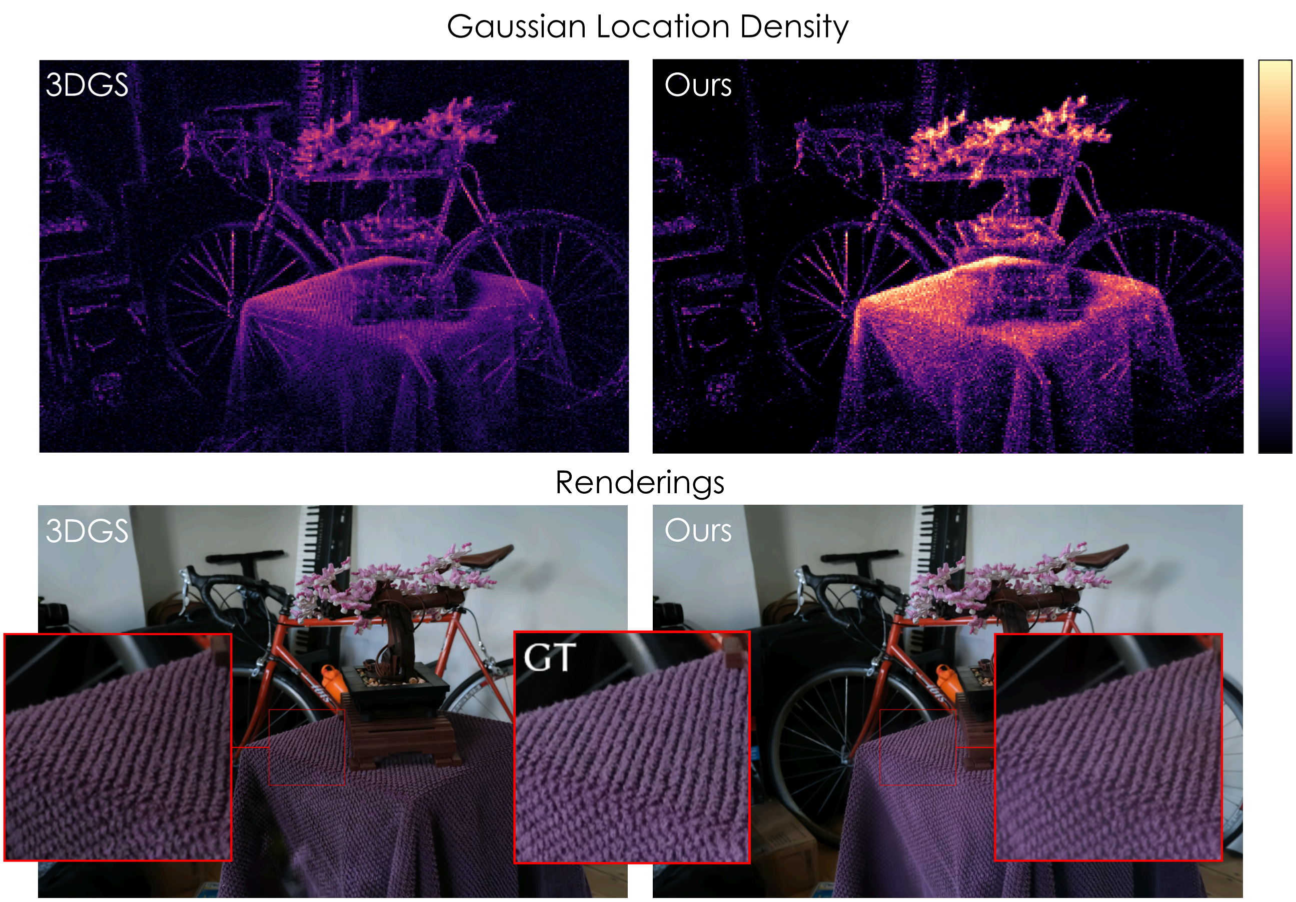



Fig. 8. Prioritizing foreground object reconstruction: we modify the proposed cost function by using a high, *negative* weight for the depth component (-15). When using a budget comparable to 3DGS, more samples are distributed to areas showing the foreground object (top). The higher density of Gaussians allows for more accurate modeling of intricate view-varying effects, such as changes in illumination (bottom).

## REFERENCES


Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV'21*. 5855–5864.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR* (2022).

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV* (2023).

András Bódis-Szomorú, Hayko Riemenschneider, and Luc Van Gool. 2016. Efficient volumetric fusion of airborne and street-side data for urban reconstruction. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 3204–3209.

Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. 2023. Unstructured lumigraph rendering. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 497–504.

Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. 2024. Revising Densification in Gaussian Splatting. arXiv:2404.06109 [cs.CV] https://arxiv.org/abs/2404.06109

Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. 2013. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)* 32, 3 (2013), 1–12.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. *ECCV* (2022).

Zilong Chen, Feng Wang, Yikai Wang, and Huaping Liu. 2024. Text-to-3D using Gaussian Splatting. arXiv:2309.16585 [cs.CV] https://arxiv.org/abs/2309.16585

Oren Dovrat, Itai Lang, and Shai Avidan. 2019. Learning to sample. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2760–2769.

Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T Barron. 2023. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. *arXiv preprint arXiv:2312.07541* (2023).

Sankeerth Durvasula, Adrian Zhao, Fan Chen, Ruofan Liang, Pawan Kumar Sanjaya, and Nandita Vijaykumar. 2023. DISTWAR: Fast Differentiable Rendering on Raster-based Rendering Pipelines. arXiv:2401.05345 [cs.CV]

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2023. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. *arXiv preprint arXiv:2311.17245* (2023).

Guangchi Fang and Bing Wang. 2024. Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. In *Proceedings of the European Conference on Computer Vision (ECCV)*. arXiv:2403.14166 [cs.CV]

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields Without Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5501–5510.

Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. 2004. Neighbourhood components analysis. *Advances in neural information processing systems* 17 (2004).

Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. 2024. GES: Generalized Exponential Splatting for Efficient Radiance Field Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep Blending for Free-viewpoint Image-based Rendering. 37, 6 (2018), 257:1–257:15.

Nishant Jain, Suryansh Kumar, and Luc Van Gool. 2023. Enhanced Stable View Synthesis. In *CVPR'23*.

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. Relu fields: The little non-linearity that could. In *SIGGRAPH 2022*. 1–9.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (2023).

Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. 2024. A Hierarchical 3D Gaussian Representation for Real-Time Rendering of Very Large Datasets. *ACM Transactions on Graphics* 43, 4 (July 2024). https://repo-sam.inria.fr/fungraph/hierarchical-3d-gaussians/

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 2024. 3D Gaussian Splatting as Markov Chain Monte Carlo. arXiv:2404.09591 [cs.CV]

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. 2023. Segment Anything. arXiv:2304.02643 [cs.CV]

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

Itai Lang, Asaf Manor, and Shai Avidan. 2020. Samplenet: Differentiable point cloud sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7578–7588.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2023. Compact 3D Gaussian Representation for Radiance Field. *arXiv preprint arXiv:2311.13681* (2023).

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)* 41, 4 (2022), 1–15.

KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. 2023. Compact3D: Compressing Gaussian Splat Radiance Field Models with Vector Quantization. *arXiv preprint arXiv:2311.18159* (2023).

Ehsan Nezhadarya, Ehsan Taghavi, Ryan Razani, Bingbing Liu, and Jun Luo. 2020. Adaptive hierarchical down-sampling for point cloud classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12956–12964.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. 2023. Compressed 3d gaussian splatting for accelerated novel view synthesis. *arXiv preprint arXiv:2401.02436* (2023).

Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. 2024. RadSplat: Radiance Field-Informed Gaussian Splatting for Robust Real-Time Rendering with 900+ FPS. *arXiv.org* (2024).

Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. 2024. Reducing the Memory Footprint of 3D Gaussian Splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7, 1 (May 2024). https://repo-sam.inria.fr/fungraph/reduced_3dgs/

Tobias Plötz and Stefan Roth. 2018. Neural nearest neighbors networks. *Advances in Neural information processing systems* 31 (2018).

Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. 2024. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. arXiv:2402.00525 [cs.GR]

Gernot Riegler and Vladlen Koltun. 2021. Stable View Synthesis. In *CVPR'21*.

Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-Fast Convergence for Radiance Fields Reconstruction. In *CVPR'22*.

Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. 2023. F2-NeRF: Fast Neural Radiance Field Training with Free Camera Trajectories. *CVPR'23* (2023).

Xiuchao Wu, Jiamin Xu, Zihan Zhu, Hujun Bao, Qixing Huang, James Tompkin, and Weiwei Xu. 2022. Scalable Neural Indoor Scene Rendering. *ACM Transactions on Graphics* (2022).

Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. 2019. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3323–3332.

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492* (2020).

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. 2022. NeRFusion: Fusing Radiance Fields for Large-Scale Scene Reconstruction. *CVPR'22* (2022).

Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao. 2024. Pixel-GS: Density Control with Pixel-aware Gradient for 3D Gaussian Splatting. In *ECCV*.