
TAMING 3DGS: HIGH-QUALITY RADIANCE FIELDS WITH LIMITED RESOURCES

Saswat Subhajyoti Mallick^{*1} Rahul Goel^{*3} Bernhard Kerbl¹
Francisco Vicente Carrasco¹ Markus Steinberger² Fernando De La Torre¹

¹Carnegie Mellon University ²Graz University of Technology

³International Institute of Information Technology, Hyderabad

ABSTRACT

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 \times 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.

Keywords Rasterization · Reconstruction · Interest point and salient region detections · Scene understanding

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 [1]. 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

^{*}Equal contribution

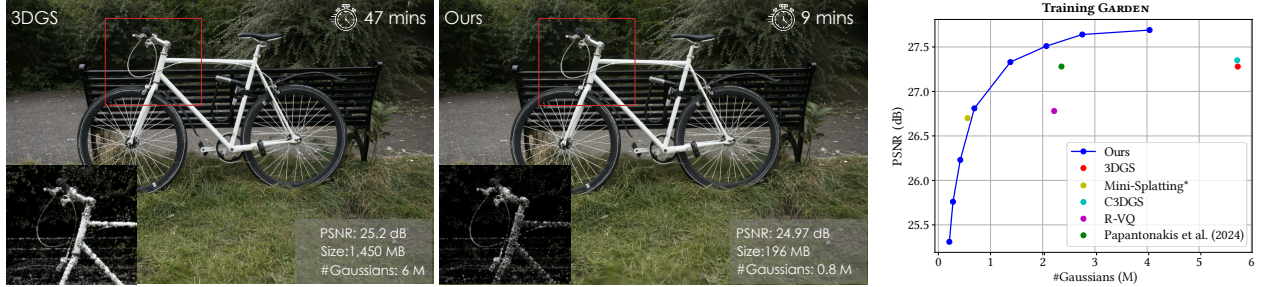


Figure 1: Our method makes 3DGS optimization fast and flexible, achieving high rendering quality on a budget. Left and middle: model size and training time are reduced by more than $5\times$. Right: Our method produces models with an exact, user-specified target size, surpassing 3DGS quality as the target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

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 that require a fixed number of inputs (e.g., classifier networks), preventing them from exploiting an otherwise efficient and 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 (less than $5\times$ on average), we tackle the suboptimal distribution and high redundancy of the original method. We propose an alternate 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.

Redundancy in 3DGS is not limited to its eventual primitive distribution. Therefore, we closely 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. We achieve a significant speedup of $4\text{--}5\times$ on average (see Fig. 1), with training times reduced to just a few minutes on consumer-grade hardware. Our contributions to taming 3DGS can thus be summarized as follows:

1. A *purely constructive, budget-constrained optimization*, 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.

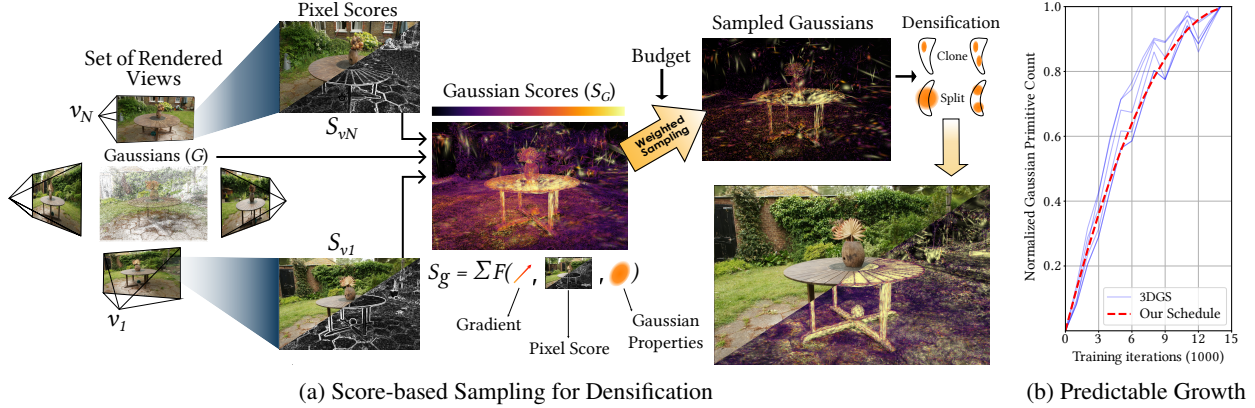


Figure 2: Overview of our method. (a) We propose a systematic redesign of 3DGS densification. To select Gaussians to densify, we sample training views and compute per-pixel saliency. A scoring function F combines gradient, saliency, and primitive properties into a per-Gaussian score S_g . (b) The addition 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.

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 [2, 3, 4] to unbounded environments [5, 6, 7]. In contrast, Neural Radiance Fields (NeRFs) [8] 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 [9, 10, 11, 12]. To address the high rendering times, voxel-based representations [13, 14] have been proposed to complement or replace select components of the NeRF architecture.

Significant breakthroughs for both training and rendering performance were marked by the use of hash grids [15] and space warping [16], at the cost of introducing quality caps. State-of-the-art NeRF-based techniques [17, 18, 19, 20, 21] 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 [1]. 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 [22, 23]; 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. [24] 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. [25] 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. [26] 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 $\approx 2\times$. Other aspects of previously proposed on-disk compression techniques, such as code-booking or entropy minimization, are directly compatible with our method, which would lead to even smaller file sizes due to our higher primitive reduction.

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 [27, 28], resampling points based on their density and distribution.

Learning-based methods introduce task-specific sampling [29] and yield results competitive with heuristic methods, such as farthest point sampling. Nezhadarya et. al. [30] uses a critical points layer, which qualifies the most significant points to the next network layer. Yang et. al. [31] implement Gumbel subset sampling to improve the classification accuracy of a network trained on point cloud data. Lang et. al. [32] introduce a differentiable projection during nearest-neighbor search that "softens" the discrete points. Inspired by these sampling-based approaches to produce compact, yet salient datasets, we redesign 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). 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 [1] is a point-based approach that models scenes using a set of 3D Gaussians, parameterized by position (μ), covariance (Σ), and opacity o . Ignoring inter-primitive overlap, the theoretical contribution of a 3D Gaussian at a point x is defined by:

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

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 α -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*:

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

where N is the number of densification steps, B is the final count (budget), and S is the number of SfM points at initialization. Since 3DGS prunes low-opacity Gaussians over time, 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 Denisfication 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 [33]. 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:

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

where \mathcal{L}_1 is the L1 loss, E is a Laplacian filter, $\mathbb{1}_{ROI}$ is a binary matrix indicating a masked region of interest, \odot is the element-wise product, and λ_1, λ_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:

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

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

Here, ∇_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.

∇_g (50): We adopt the magnitude of the positional gradient as a densification criterion from Kerbl et. al. [1]. 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.

\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 accumulated distance of covered pixels to the center of g .

Algorithm 1 Proposed Steerable Densification Method

```

1:  $T \leftarrow$  Target Gaussian count at current iteration
2:  $\mathcal{G} \leftarrow$  All Gaussians  $\{g_1, g_2, \dots, g_{|\mathcal{G}|}\}$ 
3:  $\mathcal{G}_t \leftarrow$  Gradient threshold
4:  $\mathcal{R}_t \leftarrow$  Radius threshold
5: for image  $i \in$  sampled views( $N$ ) do
6:    $P_i \leftarrow$  Photometric loss
7:   Initialise:  $c_g^i = 0$ ;  $\mathbf{D}_g^i = 0$ ;  $\mathbf{s}_g^i = 0$ ;  $\mathbf{B}_g^i = 0$ 
8:   for pixel  $p \in i$  do
9:     for  $g \in$  Gaussians contributing to  $p$  do
10:       $c_g^i += 1$ 
11:       $\mathbf{D}_g^i +=$  Distance from center of  $g$  to  $p$ 
12:       $\mathbf{s}_g^i += \mathbf{S}_v^i(p)$ 
13:       $\mathbf{B}_g^i +=$  Blending weight of  $g$  on  $p$ 
14:    end for
15:  end for
16:   $S_g = S_g + P_i \cdot F(\nabla_g, c_g^i, \mathbf{D}_g^i, \mathbf{s}_g^i, \mathbf{B}_g^i, z_g^i, o_g, s_g)$ 
17: end for
18:  $\mathbf{S}_G = [S_{g_1}, \dots, S_{g_M}]^T, \quad g_i \in \mathcal{G}$ 
19:  $B = T - |\mathcal{G}|$  ▷ #Gaussians to add
20: Top gaussian indices:  $G' \sim (\mathcal{G}, \mathbf{S}_G, B)$ 
21: for  $g \in G'$  do
22:   if  $(\nabla_g > \mathcal{G}_t) \ \& \ (\text{radius}_g > \mathcal{R}_t)$  then
23:     SPLIT
24:   else if  $(\nabla_g > \mathcal{G}_t) \ \& \ (\text{radius}_g \leq \mathcal{R}_t)$  then
25:     CLONE
26:   end if
27: end for

```

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

\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 visible 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 results in 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.

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 [34]. To remedy this, Kerbl et al. [35] 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 PNSR.

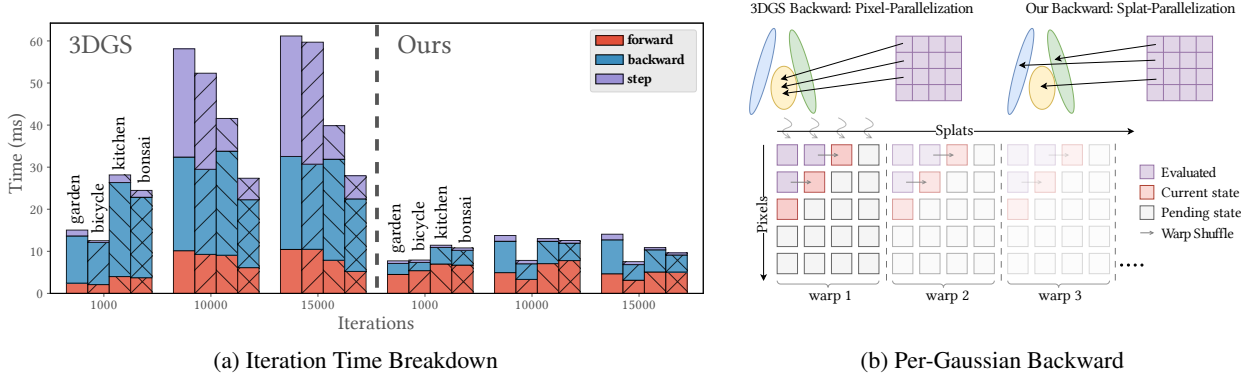


Figure 3: (a) Time spent in different parts (forward pass, backward pass, optimizer step) of one 3DGS iteration in four scenes (GARDEN, BICYCLE, KITCHEN, BONSAI). Left: analysis of original 3DGS at different stages of training. Right: using our budgeted densification and performance optimizations. (b) 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 \leftrightarrow pixel state table. During the forward, we store the pixel states for every 32^{nd} splat in the sorted list. 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.

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. 3a. 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.

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. [1] 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. 3b. The fact that each Gaussian splat maintains *a multitude* of gradients for its attributes further aggravates the overhead of this reduction [36].

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 frontmost 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 $(0, j), (N, j), \dots \forall j$. 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. 3b.

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 \leftrightarrow tile pairings. Finally, we reduce the overall rasterization workload via tighter culling as proposed by Radl et al. [37], minimizing redundant splats in the forward and backward pass.

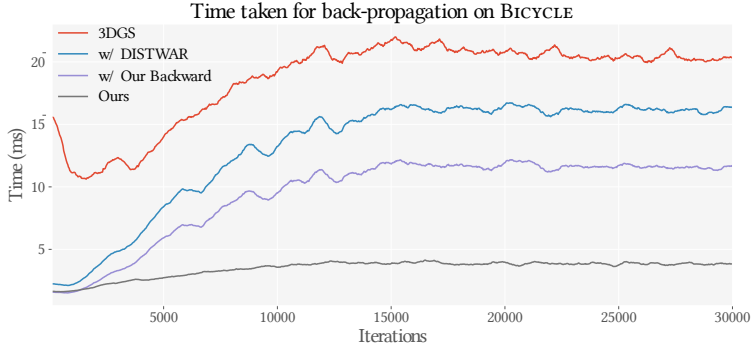


Figure 4: Backward pass duration in training of BICYCLE using 3DGS, DISTWAR [36] and our variants. For our approach, we plot the times when used with original 3DGS densification and with our more tightly budgeted schedule.

Fig. 4 compares the time taken for the backward methods of 3DGS, concurrent work DISTWAR [36], and Ours both with the original 3DGS and our budgeted optimization schedule.

4.2 Accelerated SH and Differentiable Loss Computation

Fig. 3a 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.

These measures—along with our per-splat backward—serve as drop-in replacements to the original 3DGS. The mentioned changes, except for the modified SH update schedule, yield equivalent results to 3DGS. Our experiments show that these considerably reduce 3DGS training times and can outperform the recently released version 1.0 of gsplat [38] by $1.5 \times - 2 \times$. We provide the source code for our optimizations at <https://github.com/nullptr81/3dgs-accel>.

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 [1]. 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 [39], Deep Blending [6], and MipNeRF360 [11], 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) [40], 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.

	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
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
[26]	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	0.50	4.32	0.909	30.04	0.244	24 m	<u>0.56</u>	4.51
INGP-Big	0.745	21.92	0.305	7 m	-	-	0.699	25.59	0.331	8 m	-	-	0.817	24.96	0.390	<u>8 m</u>	-	-
Ours	0.837	23.95	0.201	7 m	0.29	0.29	0.801	27.31	0.252	11 m	0.63	0.63	0.904	29.82	0.260	7 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	<u>0.847</u>	<u>23.65</u>	<u>0.176</u>	28 m	1.84	1.84	0.815	27.46	0.215	43 m	3.31	3.31	<u>0.904</u>	<u>29.64</u>	<u>0.243</u>	39 m	2.81	2.81
Ours (#3DGS)	0.851	24.04	0.170	20 m	1.84	1.84	<u>0.822</u>	<u>27.79</u>	<u>0.205</u>	<u>32 m</u>	3.31	3.31	0.907	30.14	0.235	22 m	2.81	2.81

Table 1: Quantitative comparison with other methods in two scenarios (top half & bottom half). For quality, we compare PSNR, SSIM, and LPIPS for quality. 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.

5.2 Results

We evaluate our method in two separate, budgeted scenarios (top/bottom of Table 1). For qualitative results, see Fig. 5.

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 $2\times$ the SfM points. For the larger, full-room indoor captures of Deep Blending, we use $5\times$, and for unbounded outdoor scenes, we use $15\times$. For the outdoor Tanks&Temples, the initial SfM point count is significantly higher, thus we set the budget to $2\times$ 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) [24], Compact-3DGS (R-VQ) [23], and [26]. Due to its exceptionally fast training, we also compare with the high-quality version of Instant-NGP (INGP-Big) [15]. To perform a thorough evaluation and provide comprehensive context, we also evaluate the concurrent pre-print for Mini-Splatting [41]. 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 budgeted 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 $10\times$ 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.

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 [42], and two sophisticated NeRF methods, MipNeRF360 [11] and ZipNeRF [17]. Finally, we consider the original 3DGS technique [1]. 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 to match their model size exactly. The achieved quality easily surpasses 3DGS and MipNeRF360, 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 on the scenes from Tanks&Temples. 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 fitting speed performance drop of up to 50%. Replacing our per-splat backward pass with the original has an even higher performance cost, indicating the effectiveness of our optimizations. As an

	Truck			Train		
	PSNR	LPIPS	Time	PSNR	LPIPS	Time
Ours	<u>25.20</u>	<u>0.165</u>	7 m	22.69	<u>0.238</u>	9 m
-score-based sampling	24.92	0.189	6 m	22.24	0.246	8 m
-image loss	24.94	0.187	7 m	22.08	0.242	9 m
-high opacity	25.01	0.174	7 m	22.29	0.239	9 m
-reduce SH frequency	25.39	0.161	9 m	22.75	0.235	12 m
-per splat backward	<u>25.20</u>	<u>0.165</u>	14 m	<u>22.69</u>	<u>0.238</u>	17 m

Table 2: Ablations on Tanks&Temples.

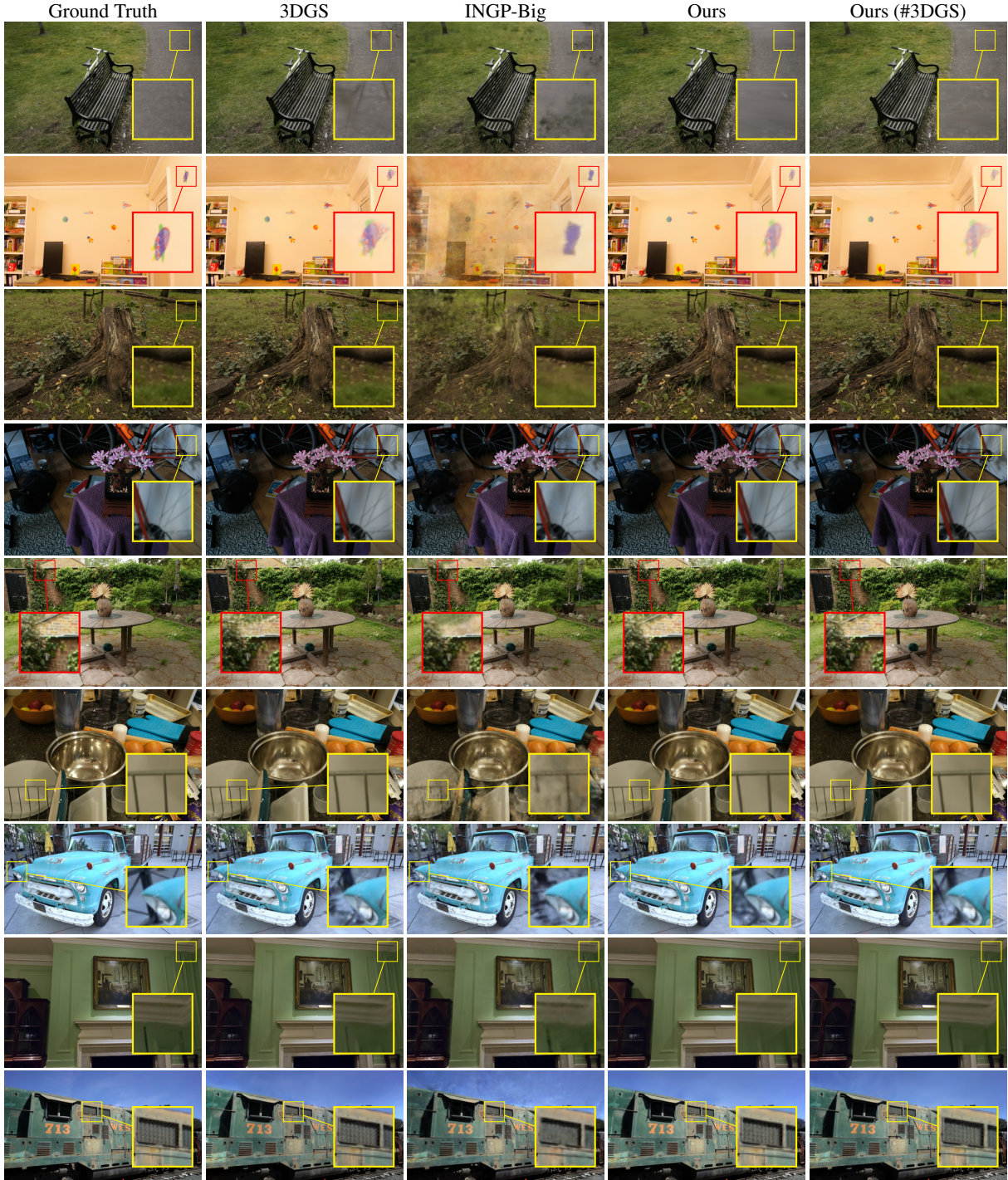


Figure 5: Qualitative comparison of results produced with our method in two budgeted scenarios to 3DGS, as well as Instant-NGP, whose training times match those of Ours. While the strictly budgeted scenario produces highly competitive results, a higher budget resolves occasional remaining blurry Gaussians.

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.

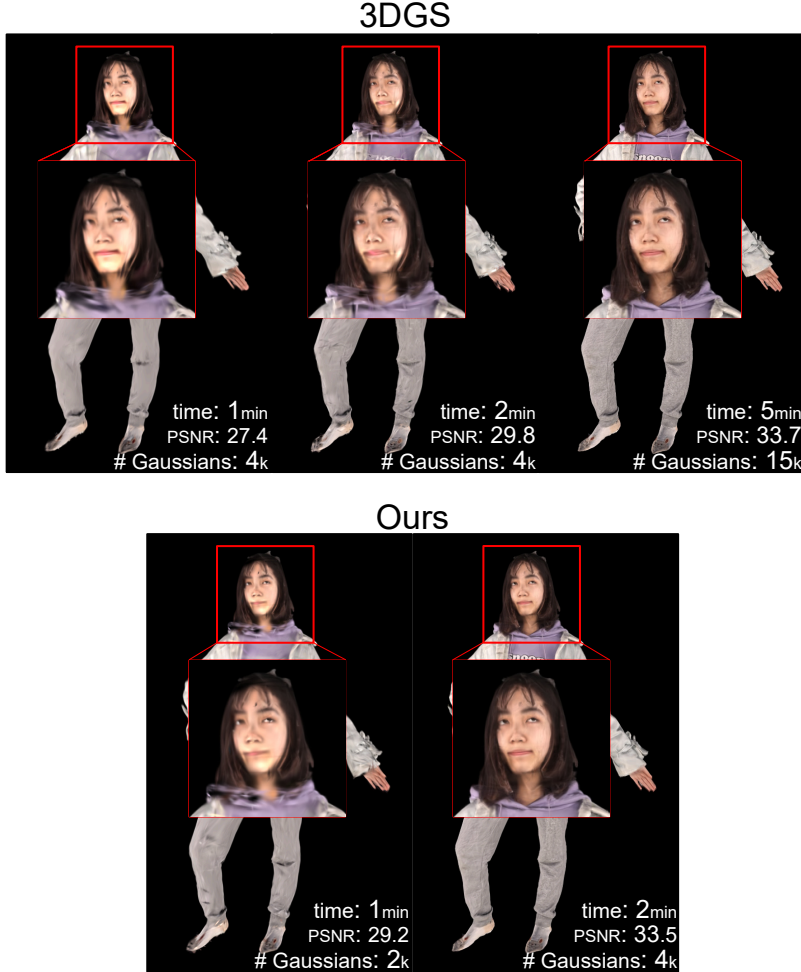


Figure 6: Demonstrating prioritization for guiding densification to regions of interest. We provide a region-of-interest mask for the face (detected with SegmentAnything [43]) on a model from the THuman dataset [44] in the computation of S_o and increase its weight to 10^3 . The above figure displays the quality of the facial region as measured via PSNR. We achieve equal scores much faster and with a fraction of the Gaussians used by 3DGS. This demonstrates the potential of our 3DGS budgeting for latency-constrained live scenarios: In a telepresence setting, we could prioritize the quality of the most frequently observed image regions—e.g., faces—and leave others under-sampled, without significantly degrading the user experience.

6 Conclusion

We have presented a highly efficient, splatting-based optimization technique for high-quality radiance fields. Our approach restrains the unpredictable behavior of the recent 3DGS technique, allowing for exact primitive budgeting, flexible sample steering, 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 potential 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 (see Fig. 6).

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 and meandering exploration as Gaussians move across the scene. We consider efficient search paths, occupancy predictions, and resolution of blind spots in scene reconstructions as exciting avenues for future work.

References

- [1] 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] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 32(3):1–12, 2013.
- [3] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 497–504. 2023.
- [4] Nishant Jain, Suryansh Kumar, and Luc Van Gool. Enhanced stable view synthesis. In *CVPR’23*, 2023.
- [5] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *CVPR’21*, 2021.
- [6] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6):257:1–257:15, 2018.
- [7] András Bódis-Szomorú, Hayko Riemenschneider, and Luc Van Gool. Efficient volumetric fusion of airborne and street-side data for urban reconstruction. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3204–3209. IEEE, 2016.
- [8] 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. *Communications of the ACM*, 65(1):99–106, 2021.
- [9] 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. In *ICCV’21*, pages 5855–5864, 2021.
- [10] Kai Zhang, Gernot Riegler, Noah Snaveley, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020.
- [11] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.
- [12] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *ECCV*, 2022.
- [13] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR’22*, 2022.
- [14] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. Relu fields: The little non-linearity that could. In *SIGGRAPH 2022*, pages 1–9, 2022.
- [15] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.
- [16] Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. F2-nerf: Fast neural radiance field training with free camera trajectories. *CVPR’23*, 2023.
- [17] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023.
- [18] Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T Barron. Smerf: Streamable memory efficient radiance fields for real-time large-scene exploration. *arXiv preprint arXiv:2312.07541*, 2023.
- [19] Xiuchao Wu, Jiamin Xu, Zihan Zhu, Hujun Bao, Qixing Huang, James Tompkin, and Weiwei Xu. Scalable neural indoor scene rendering. *ACM Transactions on Graphics*, 2022.
- [20] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. *CVPR’22*, 2022.
- [21] 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.
- [22] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compact3d: Compressing gaussian splat radiance field models with vector quantization. *arXiv preprint arXiv:2311.18159*, 2023.
- [23] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. *arXiv preprint arXiv:2311.13681*, 2023.

- [24] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. *arXiv preprint arXiv:2401.02436*, 2023.
- [25] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023.
- [26] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1), May 2024.
- [27] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17, 2004.
- [28] Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. *Advances in Neural information processing systems*, 31, 2018.
- [29] Oren Dovrat, Itai Lang, and Shai Avidan. Learning to sample. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2760–2769, 2019.
- [30] Ehsan Nezhadarya, Ehsan Taghavi, Ryan Razani, Bingbing Liu, and Jun Luo. Adaptive hierarchical down-sampling for point cloud classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12956–12964, 2020.
- [31] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3323–3332, 2019.
- [32] Itai Lang, Asaf Manor, and Shai Avidan. Samplenet: Differentiable point cloud sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7578–7588, 2020.
- [33] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo, 2024.
- [34] 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)*, 2024.
- [35] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics*, 43(4), July 2024.
- [36] Sankeerth Durvasula, Adrian Zhao, Fan Chen, Ruofan Liang, Pawan Kumar Sanjaya, and Nandita Vijaykumar. Distwar: Fast differentiable rendering on raster-based rendering pipelines, 2023.
- [37] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering, 2024.
- [38] Vickie Ye and Angjoo Kanazawa. Mathematical supplement for the gsp1at library, 2023.
- [39] 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.
- [40] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.
- [41] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians, 2024.
- [42] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5501–5510, June 2022.
- [43] 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. Segment anything, 2023.
- [44] Tao Yu, Zerong Zheng, Kaiwen Guo, Pengpeng Liu, Qionghai Dai, and Yebin Liu. Function4d: Real-time human volumetric capture from very sparse consumer rgbd sensors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR2021)*, June 2021.