Matrix-free Second-order Optimization of Gaussian Splats with Residual Sampling

Hamza Pehlivan Andrea Boscolo Camiletto Lin Geng Foo Marc Habermann
Christian Theobalt

Max Planck Institute for Informatics, Saarland Informatics Campus

Abstract

3D Gaussian Splatting (3DGS) is widely used for novel view synthesis due to its high rendering quality and fast inference time. However, 3DGS predominantly relies on first-order optimizers such as Adam, which leads to long training times. To address this limitation, we propose a novel second-order optimization strategy based on Levenberg-Marquardt (LM) and Conjugate Gradient (CG), specifically tailored towards Gaussian Splatting. Our key insight is that the Jacobian in 3DGS exhibits significant sparsity since each Gaussian affects only a limited number of pixels. We exploit this sparsity by proposing a matrix-free and GPU-parallelized LM optimization. To further improve its efficiency, we propose sampling strategies for both camera views and loss function and, consequently, the normal equation, significantly reducing the computational complexity. In addition, we increase the convergence rate of the second-order approximation by introducing an effective heuristic to determine the learning rate that avoids the expensive computation cost of line search methods. As a result, our method achieves a $4\times$ *speedup over standard LM and outperforms Adam by* $5\times$ *when the Gaussian count is low while providing* $\approx 1.3x$ *speed in moderate counts. In addition, our matrix-free implementation achieves* $2\times$ *speedup over the concurrent second-order optimizer 3DGS-LM, while using* $3.5\times$ *less memory.*
Project Page: https://vcai.mpi-inf.mpg.de/projects/LM-RS

1. Introduction

Photoreal novel view synthesis from multi-view images or video has attracted significant attention in recent years due to widely applicable downstream tasks in content creation, VR/XR, gaming, and the movie industry, to name a few. Here, Neural Radiance Fields (NeRF) [36] and 3D Gaussian Splatting (3DGS) [22] mark a major milestone, due to their unprecedented quality leading to follow ups beyond view synthesis like VR rendering [54], avatar creation [50], simultaneous localization and mapping (SLAM) [1, 35], and scene editing [3, 11].

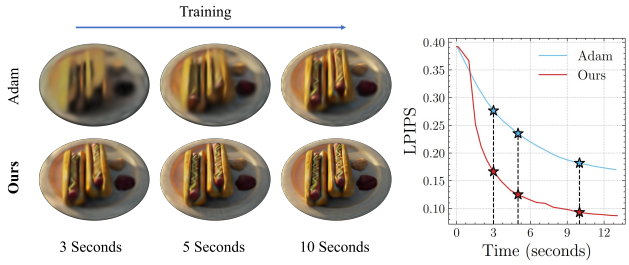


Figure 1. We introduce a matrix-free second-order optimizer for Gaussian Splatting. Notably, our dedicated optimizer converges significantly faster than Adam [24] and already achieves reasonable renderings after very few seconds of training.

However, NeRF-based models often require substantial training time, and researchers have developed various techniques to mitigate this problem. Some of them include neural hashing [39], employing explicit scene modeling [47], improved sampling strategies [21], and tensor factorization methods [10]. 3DGS [22] instead does not rely on coordinate-based representations, but leverages a set of 3D Gaussians, which can be effectively rendered into image space using tile-based rasterization. Nonetheless, optimizing the parameters of each Gaussian can still take hours. Previous work focused on finding better densification strategies [23, 31], quantization and compression of Gaussians [16, 41], and more efficient implementations for the backward pass [31], which led to substantially reduced training times. However, all of them mostly rely on a first-order optimization routine, i.e. gradient descent or Adam [24], and do not explore other (potentially second-order) alternatives.

Second-order optimization is known for having better convergence guarantees compared to first-order methods. However, its adoption in 3DGS remains challenging due to the high memory and computational demands associated with storing and inverting large Jacobian matrices. Notably, the Jacobian size scales quadratically with both the number of Gaussians and image pixels. Unlike NeRFs, which

rely on a dense neural network and a coordinate-based formulation, 3DGS benefits from inherently sparse Jacobians since each Gaussian influences only a small subset of pixels (residuals). Our *key idea* is to leverage this sparsity in order to make second-order optimization for Gaussian Splatting not only tractable, but also potentially more efficient than first-order baselines.

To this end, we formulate fitting the Gaussian parameters to the multi-view images as a non-linear least squares optimization problem and leverage the Levenberg-Marquardt (LM) algorithm for solving it. Concurrent work 3DGS-LM [20] also adopts the LM optimizer with Gaussian Splatting; however, it still suffers from the high storage and computation requirements of the Jacobian matrix. In addition, 3DGS-LM has to run the Adam [24] optimizer first to provide fast convergence, since solving the numerical system arising from the normal equation is computationally expensive.

To overcome these challenges, we propose a GPU-parallelized matrix-free conjugate gradient solver coupled with pixel sampling, which circumvents explicit storage of the Jacobian matrix and solves the matrix inverse iteratively. Firstly, we show that implementing a matrix-free solver naively does not result in a fast optimizer because of the high computational cost of Jacobian-vector products. Therefore, we propose to approximate the full normal equation by an effective view sampling strategy and by sampling individual pixels, which results in significantly faster convergence. These approximations make the Jacobian-vector products cheaper to begin with and obviate the need for the Adam optimizer. We also introduce a heuristic to automatically determine the learning rate, which eliminates the need for line search algorithms that are commonly used in conjunction with second-order optimizers.

Overall, with these improvements, our proposed second-order optimizer is both memory and compute efficient compared to 3DGS-LM [20], suggesting that second-order optimization for 3DGS is a highly promising direction for further study. Our optimizer demonstrates improvements in settings with a low and moderate number of Gaussians over first-order optimizers as well. In summary, we propose a novel second-order optimizer for 3DGS with the following features:

- Formulating Gaussian Splatting as a non-linear least squares optimization problem that is solved using a memory and computationally efficient matrix-free Levenberg-Marquardt and conjugate gradient solver specifically tailored towards 3DGS.
- A view and importance sampling strategy over the pixels (residuals) to effectively approximate the loss, leading to a significant decrease in computational complexity.
- An effective heuristic to determine the learning rate, which eliminates the need for expensive line search methods while providing stable convergence for 3DGS optimization.

2. Related Work

3D Scene Representation. Neural Radiance Fields (NeRFs) [36] represent the 3D geometry of a scene implicitly using a coordinate-based neural network. Subsequently, several variations of NeRFs have been proposed to improve visual quality [4] and to improve computational efficiency [18, 40, 49]. However, despite the significant advancements, NeRFs still face limitations, particularly in terms of their training and rendering efficiency. More recently, 3D Gaussian Splatting (3DGS) [22] has been proposed, which employs a fully explicit representation, unlocking significant enhancements in rendering speed while maintaining better qualitative results than NeRF-based methods. Recent advancements have further improved the rendering quality of 3DGS [30, 56], or its efficiency [14, 15, 28, 31, 44]. Notably, LightGaussian [14] and C3DGS [28] focus on pruning Gaussians, directly leading to improvements in rendering speed. Another related approach is to constrain the densification process, efficiently modeling the scene with less Gaussians, such as in Mini-Splatting [15] and 3DGS-MCMC [23]. Some other works aim to compress or quantize Gaussians [16, 41]. Taming3DGS [31] steers the densification process in a controlled manner, while also devising a parallelized implementation for more efficient backpropagation in CUDA. Orthogonally to these works, we explore second-order optimizers for 3DGS optimization, which is challenging due to the high memory and computational demands associated with storing and inverting large Jacobian matrices. To achieve this, we exploit the sparsity of the Jacobian matrices of 3DGS.

First-order Optimizers. Stochastic Gradient Descent (SGD) [43] was developed as a general optimization method and has been widely used in modern multi-dimensional deep learning tasks. With the addition of momentum [46], SGD can overcome local optima, a crucial feature in stochastic settings. However, a key limitation is that SGD applies the same update to all dimensions, which makes it difficult to adapt to complex problems. To address this, researchers have developed various preconditioners for the gradient information. Adagrad [13] accumulates the squares of the gradients and preconditions the gradient with the square root of the accumulated values. RMSprop [51] replaces accumulation with an exponentially moving average, which leads to more stable training. Adam [24], arguably the most popular optimizer today, combines momentum with the moving average of second-order moments. It has also become the default optimizer for many 3D tasks, including NeRF and 3DGS, and provides a strong baseline. However, despite the efficacy of first-order optimizers, there

is strong incentive to explore second-order optimizers due to their attractive properties, as discussed next.

Second-order Optimizers. Second-order optimizers often approximate the inverse of the Hessian matrix to precondition the gradient, and can offer significant advantages over first-order methods. They require far fewer iterations because they can obtain the second-order approximation of the loss landscape into account [32–34]. Additionally, second-order methods are often not as sensitive to learning rate values, and can often estimate it locally through line search algorithms [2, 20, 38], or trust region methods [12, 42].

A subset of second-order optimizers, particularly those deriving from Gauss-Newton, leverage curvature information to refine parameter updates. The general update rule for this family of optimizers follows:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \mathbf{H}^{-1}\mathbf{g} \tag{1}$$

where $\mathbf{g} \in \mathbb{R}^n$ is the gradient, $\mathbf{H} \in \mathbb{R}^{n\times n}$ is the Hessian of the loss function and $\boldsymbol{\beta} \in \mathbb{R}^n$ are the parameters to be updated, and n is the number of parameters. In a naive implementation, storing the Hessian requires $O(n^2)$ space, and computing its inverse takes $O(n^3)$ time, making it impractical for large-scale optimization tasks. Instead, a numerical approximation of the inverse can be obtained using iterative solvers like conjugate gradient (CG) [19]. This method can also eliminate the need to store the Hessian matrix, as CG only requires the results of matrix-vector multiplications [32, 34]. In the literature, this class of methods is referred to as Hessian-free or matrix-free optimization.

Specifically, within the class of second-order optimizers, the Gauss-Newton (GN) algorithm [27] is sometimes adopted to approximate the Hessian $\mathbf{H}$ with $\mathbf{J}^\top\mathbf{J}$, where $\mathbf{J}$ is the Jacobian matrix. This method effectively approximates the second-order derivative only using first-order derivatives. Notably, the Levenberg-Marquardt (LM) algorithm [37] is an extension over GN, interpolating between GN and gradient descent, resulting in more stable optimization. Recently, some works [20, 26] also explored second-order optimizers for 3DGS optimization. 3DGS^2 [26] explores an algorithm based on Newton's method, but makes it computationally tractable by limiting the second-order computations to a small locality. Concurrent work 3DGS-LM [20] employs the LM optimizer with a dedicated cache for intermediate Jacobian entries for fast computation. However, caching Jacobians results in quadratic memory requirements, as it scales with respect to both pixels and parameters. In addition, 3DGS-LM heavily relies on Adam optimizer for performance since processing a large cache is also expensive.

In our paper, we also adapt the LM optimizer, proposing a fast GPU-parallelized conjugate solver for 3DGS, which is a matrix-free optimization that avoids explicitly storing the Jacobian matrix and solves the matrix inverse iteratively. Our LM optimizer exploits the sparsity property within 3DGS representations for computational efficiency by an effective view sampling strategy and by residual sampling, which enables the matrix-free implementation and provides faster convergence.

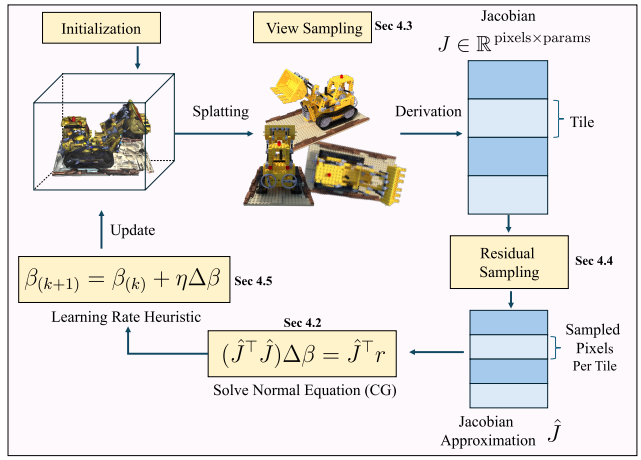


Figure 2. Overview of our method is given. We start from randomly initialized Gaussians and gradually refine them with Levenberg-Marquardt optimizer. Since dealing with the true Jacobian matrix is costly, we approximate it with a tile-aware sampling algorithm. After we solve the normal equations with approximated Jacobians, we update the parameters using a learning rate heuristic. Note that the Jacobians are never materialized in memory, and the normal equation is solved with our matrix-free algorithm.

3. Background - 3D Gaussian Splatting (3DGS)

3DGS is a point-based representation that can reconstruct 3D scenes with high fidelity. The 3D scene is represented using a set of Gaussians, where each Gaussian has an optimizable mean $\mathcal{X}$ and covariance $\boldsymbol{\Sigma}$:

$$G(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\mathcal{X})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathcal{X})} \tag{2}$$

To ensure the covariance matrix remains positive-definite, it is decomposed into rotation $\mathbf{R}$ and scale $\mathbf{S}$ matrices:

$$\Sigma = \mathbf{R}\mathbf{S}\mathbf{S}^\top\mathbf{R}^\top. \tag{3}$$

To render a pixel, the Gaussians are first projected onto the 2D image plane, and sorted according to their depth. A pixel value $\hat{I}_i$ is computed using α-blending, which combines the color c and per-pixel opacity α of the projected Gaussians:

$$\hat{I}_i = \sum_{n\leq\mathcal{S}} c_n \cdot \alpha_n \cdot \prod_{m<n}(1-\alpha_m) \quad \text{, where} \quad \alpha_n = o_n \boldsymbol{\pi}_{\text{cam}}(G_n) \tag{4}$$

where $\mathcal{S}$ represents the total number of depth-sorted Gaussians projected onto pixel i, and opacity contribution α_n is

obtained by multiplying Gaussian base opacity o_n with its 2D projection (via camera projection $\boldsymbol{\pi}_{\text{cam}}$).

The parameters of the Gaussians are updated by optimizing a loss function between the rendered image $\hat{\mathcal{I}}$ and ground-truth image $\mathcal{I}$. This optimization is done predominantly using the Adam optimizer:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_f(\hat{\mathcal{I}}, \mathcal{I}), \tag{5}$$

where $\mathcal{L}_f(\cdot)$ is a loss function that quantifies image differences. In this paper, we experiment with mean squared error (MSE) and structural similarity (SSIM) [52] loss functions. Additionally, 3DGS models the view-dependent effects with spherical harmonics. In this work, however, we assume each scene has Lambertian surface and disable the corresponding spherical harmonics levels.

4. Method

We first introduce the LM optimizer in Sec. 4.1, which we adopt in this work. Then, we derive why a naive implementation is not feasible for Gaussian Splatting. To resolve this issue, in Sec. 4.2, we discuss our matrix-free approach to adapt the LM optimizer. In detail, we propose a GPU-parallelized conjugate gradient solver for our second-order 3DGS optimizer, which circumvents explicit storage of the Jacobian matrix, and solves the matrix inverse iteratively. Next, in Sec. 4.3, we introduce a new view sampling strategy to effectively approximate the full normal equation, thereby providing reliable update step directions by integrating information from multiple views. In Sec. 4.4, we present our residual sampling, providing an approximate loss function, which results in significantly faster convergence. Lastly, in Sec. 4.5, we introduce a heuristic to automatically determine the learning rate, which eliminates the need for line search algorithms that are commonly used in conjunction with second-order optimizers. The overview of all the components is given in Fig. 2.

4.1. Levenberg-Marquardt Optimizer for 3DGS

We use the LM optimizer with a fixed damping parameter to compute an update step by minimizing the loss over a mini-batch B, which includes images of the shape height H, width W, and channels C. The optimizable parameters for each Gaussian are opacity $o \in \mathbb{R}$, color $c \in \mathbb{R}^3$, mean value $\mathcal{X} \in \mathbb{R}^3$, scale $s \in \mathbb{R}^3$, and quaternion rotation $q \in \mathbb{R}^4$. We represent all Gaussian parameters with $\boldsymbol{\beta} \in \mathbb{R}^P$, where we denote the total number of optimizable parameters with P.

The rendering loss function in Eq. 5 can be rewritten as a nonlinear least squares objective:

$$\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M} (r_i)^2 \tag{6}$$

Algorithm 1 One step of LM optimizer with matrix-free PCG solver.

Input: Gaussians $\boldsymbol{\beta}_k$, cameras $\mathcal{C}$
Output: Updated Gaussians $\boldsymbol{\beta}_{k+1}$

$\mathcal{I}, \mathcal{C}_B = \text{getBatch}(\mathcal{C})$ ▷ View Sampling, Sec. 4.3
$\hat{\mathcal{I}} = \text{splatting}(\boldsymbol{\beta}_k, \mathcal{C}_B)$
$\mathbf{r} = \text{getResiduals}(\mathcal{I}, \hat{\mathcal{I}})$ ▷ Eq. 7
▷ Beginning of matrix-free PCG solver (Sec 4.2)
$\mathbf{r}_0 = \mathbf{J}^\top \mathbf{r}$ ▷ Estimated in Sec. 4.4
$\mathbf{M}^{-1} = 1/\text{Diag}(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I})$ ▷ Estimated in Sec. 4.4
$\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 = \mathbf{z}_0$
$\mathbf{x}_0 = \mathbf{0}$
for $i = 0$ to $\text{PCG}_{\text{iters}}$ **do**
  $\mathbf{u} = (\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I})\mathbf{p}_i$ ▷ Estimated in Sec. 4.4
  $\alpha = \frac{\mathbf{r}_i^T \mathbf{z}_i}{\mathbf{p}_i^T \mathbf{u}}$
  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha \mathbf{p}_i$
  $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha \mathbf{u}$
  $\mathbf{z}_{i+1} = \mathbf{M}^{-1}\mathbf{r}_{i+1}$
  $\beta = \frac{\mathbf{r}_{i+1}^T \mathbf{z}_{i+1}}{\mathbf{r}_i^T \mathbf{z}_i}$
  $\mathbf{p}_{i+1} = \mathbf{z}_{i+1} + \beta \mathbf{p}_i$
end for
$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \eta\, x_{i+1}$ ▷ Dynamic LR Sched., Sec. 4.5
return $\boldsymbol{\beta}_{k+1}$

where r is a *residual* and $M = BHWC$. The residuals are defined as pixel and structural similarity losses; however, to keep the derivation concise, we omit the structural similarity term:

$$r_i = \hat{\mathcal{I}}_i - \mathcal{I}_i \tag{7}$$

Then, the update vector $\Delta\boldsymbol{\beta} \in \mathbb{R}^P$ is retrieved by solving the following *normal equation*:

$$\left(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}\right) \Delta\boldsymbol{\beta} = \mathbf{J}^T \mathbf{r} \tag{8}$$

where λ is the fixed damping parameter, $\mathbf{r} \in \mathbb{R}^M$ is the vectorized form of the residuals r_i, and $\mathbf{J} \in \mathbb{R}^{M \times P}$ is the Jacobian of $\mathbf{r}$.

One iteration of the LM optimizer is completed after we update all the Gaussian parameters $\boldsymbol{\beta}$ with the learning rate η:

$$\boldsymbol{\beta}_{(k+1)} = \boldsymbol{\beta}_{(k)} + \eta \cdot \Delta\boldsymbol{\beta} \tag{9}$$

Note that in the original 3DGS, for each parameter group (opacity, color, mean, scale, and rotation), a different learning rate is used. On the other hand, we use a uniform learning rate across parameters because Gauss-Newton-type methods inherently incorporate parameter scaling through the Hessian approximation $\mathbf{J}^\top \mathbf{J}$. In Sec. 4.5, we discuss a heuristic to determine the learning rate at every iteration.

In practice, naively solving the normal equation is not feasible, especially in large-scale numerical systems. First, the Jacobians scale quadratically with both the number of

Gaussians and the number of pixels, and storing the Jacobians explicitly in memory quickly becomes infeasible. For example, using 100 images at a resolution of 800×800 pixels as training data, along with 10 000 Gaussians to model the 3D scene, would result in Jacobians exceeding 100 TB in size. Even assuming a sparse storage format with a 99% sparsity, the storage requirement would still exceed 1 TB for a true LM optimizer. In addition, explicitly inverting the system matrix has $O(P^3)$ time complexity when implemented naively. Thus, computing the explicit inverse is often infeasible, and previous work has focused on solving the normal equation iteratively [9, 32, 34], but they still remain computationally expensive, as discussed in the next section.

4.2. Matrix-free Preconditioned Conjugate Gradient (PCG) Solver for 3DGS

We first discuss our matrix-free PCG solver to adapt the LM optimizer. In this subsection, we present a GPU-parallelized conjugate gradient solver, which does not need to store the Jacobian matrix explicitly, avoiding the memory issues mentioned in Sec. 4.1. To achieve this, we solve Eq. 8 via the preconditioned conjugate gradient algorithm [19], which only needs results of Jacobian-vector products, and finds the solution $\Delta\boldsymbol{\beta}$ iteratively.

To improve the condition number of the linear system in Eq. 8, we used the Jacobi preconditioner due to its simplicity and effectiveness. More specifically, we use $\frac{1}{\mathrm{diag}(\mathbf{J}^\top\mathbf{J}+\lambda\mathbf{I})} \in \mathbb{R}^P$, as the precondition. We provide the implementation of the solver in CUDA, together with efficient Jacobian-vector product kernel, which is implemented with forward mode differentiation and dual numbers [5, 53]. See the supplementary material for details about the kernel design. The pseudocode of the optimizer is given in Alg. 1.

Although this optimizer is able to converge to the final solution in a limited number of steps, it is $4\times$ slower than our final method, as shown in Tab. 3. The main reason for this is that the conjugate gradient algorithm needs to run several iterations, therefore, we need to repeatedly compute the $\mathbf{J}^\top\mathbf{J}\mathbf{p}$ product arising on line 10 of Alg. 1. This kind of computational complexity commonly arises when a second-order optimizer is used in large-scale numerical systems, and usually, an approximation method is used for efficiency. The common approximation methods are diagonal [6, 29, 48, 55] and block-diagonal [7, 33, 45] approximation of the Hessian or $\mathbf{J}^\top\mathbf{J}$ matrix.

We observe that the Hessian approximation $\mathbf{J}^\top\mathbf{J}$ in 3DGS does not exhibit a *dominant* diagonal or block-diagonal structure, as illustrated in Fig. 3. This arises because each pixel is rendered through the interaction of multiple Gaussians, as described in Eq. 4. In other words, no single Gaussian independently represents a surface; instead, it relies on the collective contribution of surrounding Gaussians to accurately capture the ground truth. This leads to many off-diagonal and off-block-diagonal entries in $\mathbf{J}^\top\mathbf{J}$. Therefore, the common diagonal and block-diagonal approximations do not provide good approximations. To this end, we propose to estimate the loss function with our proposed view and residual sampling. The sampling techniques enable the matrix-free implementation as the cost of solving Eq. 8 is drastically reduced.

4.3. View Sampling

Second-order methods like LM are typically used in deterministic settings where the full objective is evaluated at every iteration. If this is not the case, estimating local curvature can be problematic and become unreliable [9]. This poses a challenge in the 3DGS setting, where the number of views can exceed hundreds, as incorporating all of them at the same time is infeasible. Yet, to compute meaningful gradients, we must find an effective way to approximate the full normal equation in Eq. 8.

To this end, we introduce a camera sampling approach that allows us to get a diverse set of views in each batch. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ be the set of all cameras. For each camera c_i, we form the feature vector:

$$f_i = [x_i, y_i, z_i,\ dx_i, dy_i, dz_i] \tag{10}$$

where (x, y, z) is the cameras' normalized location and (dx, dy, dz) is the viewing direction. We then run K-Means clustering on $\{f_i\}_{i=1}^N$ to partition the cameras into batch size number of clusters (e.g., 8 clusters created for a batch size of 8). A camera is then randomly selected from every cluster and collected in the set $\mathcal{C}_B$, which serves as training input. This approach ensures that each batch captures a more balanced and diverse set of views, resulting in a more accurate estimation of the curvature information. As evidenced by Tab. 2, this method converges to higher scores compared to the random sampling of the cameras.

4.4. Estimation of Loss Function with Residual Sampling

Additionally, to improve the efficiency and feasibility of our second-order optimizer for 3DGS, we further exploit the sparsity property within the 3DGS representation, which has been discussed in Sec. 4.2. Yet, at the same time, as observed in Fig. 3, there does not exist a well-organized structure to the sparsity. Instead, in this paper, we propose to adopt a residual sampling approach.

We begin our derivation by rewriting our least squares loss function in Eq. 6 as an integral: Then, the loss function is given as:

$$L = \int_\Omega r(\mathbf{p})^2 \, d\mathbf{p} \tag{11}$$

where the integral is defined over Ω, the union of all image domains of all cameras, and calculated at the coordinates

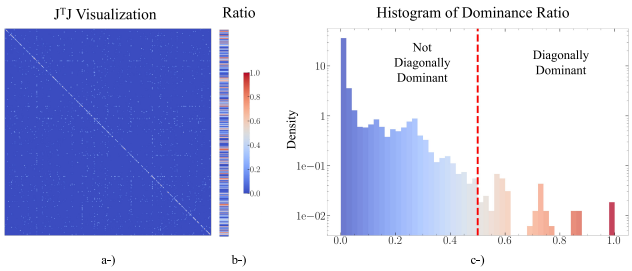


Figure 3. We visualize a normalized $\mathbf{J}^\top\mathbf{J}$ matrix for one downsampled training image (a). While elements in the diagonal are common, we compute the *dominance ratio* as the normalized ratio between the diagonal element over the sum of other elements in the same row (b) and show how only a limited number of parameters lead to a diagonally dominant linear system (c).

$\mathbf{p} \in \mathbb{R}^2$ of the image plane. To estimate the loss function L, we express it as an expectation with respect to an arbitrary probability distribution $q(\mathbf{p})$:

$$L = \int_\Omega r(\mathbf{p})^2 \frac{q(\mathbf{p})}{q(\mathbf{p})} d\mathbf{p} = \mathbb{E}_{\mathbf{p}\sim q(\mathbf{p})}\left[\frac{r(\mathbf{p})^2}{q(\mathbf{p})}\right]. \tag{12}$$

This formulation allows us to interpret $q(\mathbf{p})$ as a sampling distribution over the domain Ω. Consequently, we can approximate the expectation numerically using Monte Carlo estimation. In particular, by independently sampling N pixels $\{\mathbf{p}_i\}_{i=1}^N$ from $q(\mathbf{p})$, the loss function can be approximated as:

$$L \approx \hat{L} = \frac{1}{N}\sum_{i=1}^{N} r(i)^2 \frac{1}{q(i)}. \tag{13}$$

The estimated loss function allows us to derive the estimated versions of Jacobian-vector products arising in the PCG solver. The derivative of the estimated loss function with respect to a parameter becomes:

$$\frac{\partial \hat{L}}{\partial \boldsymbol{\beta}_j} = \frac{2}{N}\sum_{i=1}^{N} \frac{\partial r_i}{\partial \beta_j} r_i \frac{1}{q(i)} \tag{14}$$

This gradient estimates the right-hand side of the normal equation given in Eq. 8 since $\frac{\partial r_i}{\partial \beta_j} = J_{ij}$.

Similarly, we can calculate the Hessian of the estimated loss function given in Eq. 13. To do so, we take the derivative of Eq. 14 with respect to parameter $\boldsymbol{\beta}_k$:

$$\hat{H}_{jk} = \frac{2}{N}\sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial \beta_j}\frac{\partial r_i}{\partial \beta_k} + r_i \frac{\partial^2 r_i}{\partial \beta_j \partial \beta_k}\right)\frac{1}{q(i)} \tag{15}$$

When we apply the Gauss-Newton approximation to the Hessian matrix, the term with second derivative in Eq. 15 is ignored, and we are left with an estimation of $\mathbf{J}^\top\mathbf{J}$.

Note that this approximation preserves the symmetry and positive semi-definiteness of $\mathbf{J}^\top\mathbf{J}$, a property required for the convergence of the conjugate gradient algorithm.

In this work, we evaluated a range of candidate distributions q and found that simple uniform distribution works well. For a comparison of alternative distributions and their respective results, please refer to the Supplementary.

In practice, sampling pixels over the image does not efficiently integrate with GPU programming and the backward pass of 3DGS. The reason is that an image is divided into 16×16 tiles and when the pixels are selected randomly, some tiles get more samples than others, causing an unbalanced workload among thread blocks. Moreover, the random nature of the sampling does not allow fixed thread assignment per tile. Therefore, we use a stratified sampling strategy by distributing samples among tiles, and we perform sampling inside them. In other words, instead of sampling N pixels from the entire image, we sample $N/\mathcal{T}$ pixels from each tile, where $\mathcal{T}$ is the total number of tiles in the image.

The sampling mechanism allows us to achieve $4\times$ speedup over the non-approximated LM optimizer, while maintaining similar performance, as shown in Tab. 3.

4.5. Dynamic Learning Rate Scheduler

When second-order optimizers are applied to large-scale systems, numerical errors can occur, causing the solution vector $\Delta\boldsymbol{\beta}$ to overshoot the true loss landscape. To determine the optimal learning rate, line search algorithms or trust region methods [2, 8, 12, 20, 38, 42] are usually employed. However, these methods require additional forward or backward passes, increasing the computational overhead and slowing down the overall algorithm. Rather than relying on these methods, we estimate the learning rate by constraining the maximum update of color values. In 3DGS, the color parameters range between ≈ -1.77 and ≈ 1.77 due to the level 0 spherical harmonic coefficient. This gives us a natural bound for the color parameters. We trust the update direction $\Delta\boldsymbol{\beta}$, as long as the resulting color change does not exceed 1. If the change surpasses this threshold, we scale $\Delta\boldsymbol{\beta}$ so that the maximum change remains 1. Notably, this scaling is applied uniformly across all parameters, regardless of their type (e.g., opacity, color, mean, scale, or rotation). The effectiveness of this learning rate heuristic is demonstrated in Tab. 4, and examples of assigned learning rates are given in the Supplementary.

5. Results

Datasets and Metrics. We conduct our main experiments on the synthetic NeRF [36] and real-world Mip-NeRF360 indoor datasets [4]. In the synthetic dataset, we position 10,000 Gaussians at random locations with random colors inside of the cube encapsulating the object. In the real-world scenes, the Gaussians are initialized with structure-from-motion point clouds as in 3DGS [22]. For all experiments, we use the default training and test splits. We report

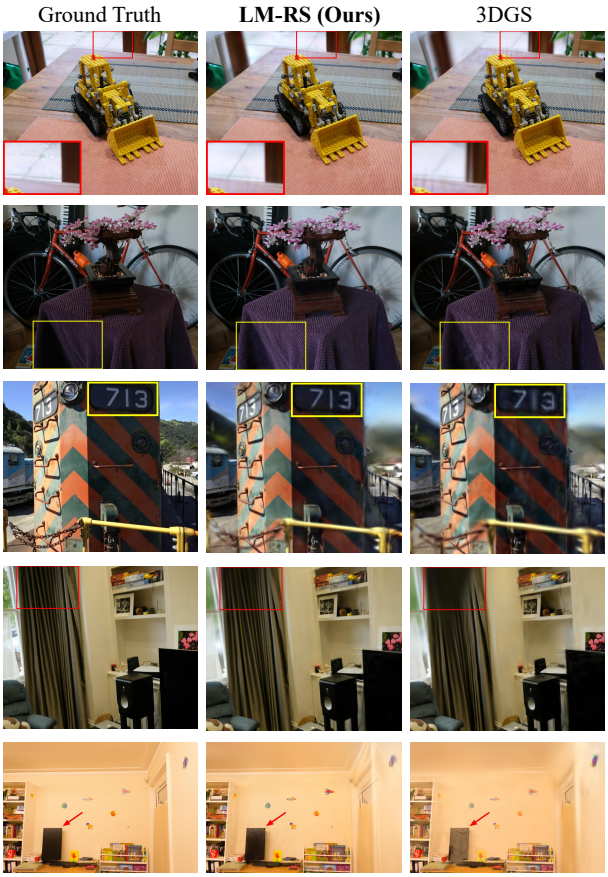


Figure 4. We evaluate the performance of the optimizers in the real-world scenes. Our optimizer attains better quality in shorter training time.

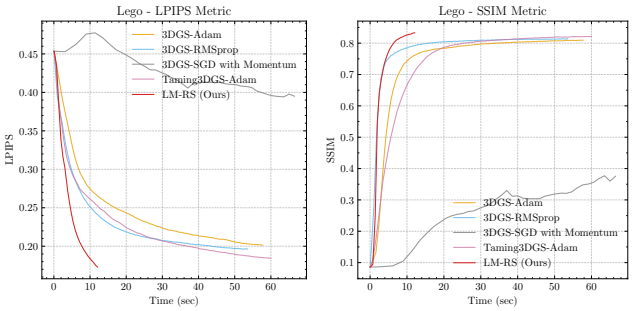
the test set performance based on structural similarity index (SSIM) [52], learned perceptual image patch similarity score (LPIPS) [57], and peak signal-to-noise ratio (PSNR).

Baselines. In the synthetic dataset, we compare our optimizer against Adam [24], RMSprop [51], and SGD with momentum [46], using their respective implementations in PyTorch. The gradients are computed using either the vanilla 3DGS or Taming 3DGS [31], which enhances the efficiency of the backward computation by storing the gradient state at every 32nd splat.

In the real-world scenes, we compare against vanilla 3DGS [22] and 3DGS-LM [20]. For comparisons against 3DGS-LM, we contacted the authors[1], who kindly provided PSNR scores, runtime, and average memory usage across several datasets. In addition, they provided results obtained without the Adam optimizer, which are denoted in the tables as 3DGS-LM (w/o Adam). First-order baselines are run for $10,000$ iterations, and we performed a hyperparameter search to find the best learning rates.

Implementation Details. We conducted synthetic

[1]At the time of our submission, the code for 3DGS-LM had not yet been released.



Method	LPIPS ↓	SSIM ↑	PSNR ↑	Time (s)	Memory (GB)
3DGS-SGD	0.331	0.488	18.56	164	**2.43**
3DGS-RMSprop	0.159	0.856	25.52	54	**2.43**
3DGS-Adam	0.162	0.855	25.49	58	**2.43**
Taming3DGS-Adam	0.147	0.862	26.01	60	2.54
LM-RS (Ours)	**0.139**	**0.872**	**26.14**	**13**	3.07

Figure 5. The averaged results of competing methods on the synthetic NeRF dataset are presented. Our method achieves similar quality in a shorter amount of time. We also provide convergence plots on the Lego scene.

dataset experiments on a NVIDIA Tesla A40 GPU and real-world experiments on a NVIDIA Tesla A100 GPU. Our method and the baselines optimize the mean squared error (MSE) loss. We set the sampled pixels per tile (N) to 32 in each experiment. We use a fixed damping parameter λ of 0.1 for the synthetic NeRF scenes and Mip-NeRF360 dataset. For the synthetic dataset, the batch size is fixed at 8; for real-world datasets, it is initialized at 16 and increased to 32 after the 50th iteration. The maximum number of conjugate gradient iterations starts at 3 for synthetic datasets and at 5 for real-world datasets, rising to 8 after the 50th iteration.

Table 1. This table presents the comparisons of competing methods averaged in the Mip-NeRF360 indoor dataset.

Method	SSIM↑	PSNR↑	Time (s)	Memory (GB)
3DGS	0.854	28.21	252	**5.81**
3DGS-LM (w/o Adam)	-	26.72	1186	≈ 50
3DGS-LM	-	28.31	398	≈ 50
LM-RS (Ours)	**0.859**	**28.59**	**190**	14.75

5.1. Comparison

Next, we evaluate our method, named Levenberg-Marquardt with Residual Sampling (LM-RS), on synthetic NeRF scenes and present the results in Fig. 5. As shown, our second-order optimizer converges rapidly and achieves a substantial $\approx 5\times$ speedup over Adam. We observed that SGD with momentum consistently underperforms compared to other optimizers, despite our efforts at hyperparameter tuning. This highlights the importance of adaptive learning rates for optimizing 3DGS, a feature incorporated in all other evaluated optimizers. We also did not observe significant speed benefits from the optimizations of Taming 3DGS, which is likely due to a comparatively lower number of Gaussians in this setting that reduces the advantages of

gradient caching.

In addition, we provide results in Mip-NeRF360 dataset in Tab. 1 and Fig. 4. Our method achieves $1.3\times$ speedup over Adam, $2\times$ over 3DGS-LM and $6\times$ speedup over 3DGS-LM (w/o Adam). Thanks to our matrix-free algorithm, we do not rely on the Adam optimizer at all, and our memory requirement is $3.5\times$ less than 3DGS-LM.

Please refer to the Supplementary for the results using SSIM loss, as well as quantitative results in Tanks & Temples [25] and DeepBlending scenes [17].

5.2. Ablation Studies

In this section, we ablate various design choices incorporated into our optimizer.

View Sampling. We propose a view sampling approach in Sec. 4.3 to ensure that our optimizer can effectively estimate the local curvature of the loss landscape while still maintaining a relatively low batch size, through sampling of diverse views in a batch. In Tab. 2, we ablate our design choice, where we observe that our method shows improvements compared to random view sampling. We also report performance with reduced batch size, where we observe that the performance is significantly lower.

Table 2. The table demonstrates the significance of high batch sizes and the effectiveness of our clustering-based camera sampler compared to random sampler. The results are averaged over Tanks & Temples scenes. (MBS= Maximum Batch Size)

View Sampler	LPIPS ↓	SSIM ↑	PSNR ↑
Clustering-Based, MBS=8	0.363	0.640	19.18
Clustering-Based, MBS=16	0.271	0.719	21.37
Random, MBS=32	0.276	0.722	21.68
Clustering-Based, MBS=32	**0.253**	**0.731**	**21.83**

Residual Sampling Size. Our proposed sampling approach in Sec. 4.4 is the core algorithm that provides significant speed benefits, enabling faster processing by effectively reducing the dimensionality of highly redundant Jacobians. Here, we compare our performance against three baselines in Tab. 3. Firstly, when no sampling is used at all, i.e., the full Jacobians are considered every time, the amount of training time required increases significantly, showing the importance of sampling. When we adopt our pixel sampling, but use a high number of sampled pixels per tile (e.g., $N = 128$, $N = 64$), there are some gains over no sampling, but its efficiency is still suboptimal. All these show the efficacy of our residual sampling method.

Dynamic Learning Rate Scheduler. We compare our learning rate heuristic introduced in Sec. 4.5 against uniform learning rate, Armijo line search [2, 42], and a grid search method in Tab. 4. The grid search method selects the learning rate at each iteration by identifying the value that results in the greatest reduction in the objective function. Although this method is stable, the learning rates obtained are too pessimistic, and it cannot reach the quality of our heuristic. Armijo line search improves the grid search method by picking the highest learning rate that results in *sufficient* reduction of the loss. While this approach has been successfully integrated as an improvement in deterministic second-order optimizers, our findings indicate that its effectiveness is limited in stochastic settings, such as ours, where optimization relies on multiple approximations. We also found that the uniform learning rate can diverge (LR=0.1) or behave suboptimally (LR=0.07) compared to our scheduler.

Table 3. We show that sampling fewer pixels per tile (N) has little impact on quantitative performance while substantially increasing efficiency. The results are averaged over the MipNeRF-360 indoor scenes.

Residual Sampler	LPIPS ↓	SSIM ↑	PSNR ↑	Time (s)
LM - No Sampling	**0.143**	**0.868**	**28.68**	810
LM-RS - $N = 128$	0.159	0.860	28.63	470
LM-RS - $N = 64$	0.160	0.860	28.66	286
LM-RS - $N = 32$	0.161	0.859	28.59	**190**

Table 4. We show that our dynamic learning rate algorithm can converge faster, compared to uniform learning rate or search methods. The results are averaged across Mip-NeRF360 indoor scenes.

Learning Rate	LPIPS ↓	SSIM ↑	PSNR ↑	Time (s)
Uniform $LR = 0.07$	0.202	0.842	27.91	**188**
Uniform $LR = 0.1$	0.281	0.669	23.20	**188**
Grid Line Search	0.224	0.821	27.12	397
Armijo Line Search	0.264	0.763	23.66	416
Our Scheduler (Sec. 4.5)	**0.161**	**0.859**	**28.59**	190

6. Limitations

We use only a diagonally approximated SSIM loss for performance reasons, which may affect the convergence behavior. Future work could explore more efficient implementations enabling the use of full SSIM loss. In addition, our matrix-free implementation requires more intermediate vectors than Adam, leading $3\times$ higher memory usage.

7. Conclusion

By leveraging the inherent sparsity of the Jacobian matrix and integrating a GPU-parallelized conjugate gradient solver, our method significantly reduces the computational overhead. Our novel view and pixel-wise sampling further enhance efficiency, enabling rapid convergence by decreasing the per-step overhead. Additionally, our dynamic learning rate scheduler eliminates the need for costly line search procedures, further accelerating training. Our approach achieves up to $5\times$ speedup over Adam, particularly excelling in scenarios with low number of Gaussians. Overall, our results highlight the potential of second-order methods in accelerating optimization for 3D Gaussian Splatting. We anticipate that future work will refine these techniques, particularly by incorporating additional loss terms without incurring runtime overhead, further advancing the efficiency and quality of Gaussian-based scene representations.

References

[1] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2): 4606–4613, 2022. 1

[2] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966. 3, 6, 8

[3] Chong Bao, Yinda Zhang, Bangbang Yang, Tianxing Fan, Zesong Yang, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Sine: Semantic-driven image-based nerf editing with prior-guided editing field. In *The IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 2023. 1

[4] 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. *Internation Conference on Computer Vision (ICCV)*, 2021. 2, 6

[5] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018. 5

[6] Suzanna Becker and Yann Lecun. Improving the convergence of back-propagation learning with second-order methods. In *Proceedings of the 1988 Connectionist Models Summer School, San Mateo*, pages 29–37. Morgan Kaufmann, 1989. 5

[7] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, pages 557–565. PMLR, 2017. 5

[8] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. 6

[9] Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011. 5

[10] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European conference on computer vision*, pages 333–350. Springer, 2022. 1

[11] Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 21476–21485, 2024. 1

[12] Andrew R Conn, Nicholas IM Gould, and Philippe L Toint. *Trust region methods*. SIAM, 2000. 3, 6

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011. 2

[14] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37: 140138–140158, 2025. 2

[15] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, pages 165–181. Springer, 2024. 2

[16] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*, pages 54–71. Springer, 2024. 1, 2

[17] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018. 8

[18] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5875–5884, 2021. 2

[19] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952. 3, 5

[20] Lukas Höllein, Aljaž Božič, Michael Zollhöfer, and Matthias Nießner. 3dgs-lm: Faster gaussian-splatting optimization with levenberg-marquardt. *arXiv preprint arXiv:2409.12892*, 2024. 2, 3, 6, 7

[21] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. Efficientnerf: Efficient neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12902–12911, 2022. 1

[22] 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. 1, 2, 6, 7

[23] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *Advances in Neural Information Processing Systems*, 37:80965–80986, 2025. 1, 2

[24] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 1, 2, 7

[25] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36 (4):1–13, 2017. 8

[26] Lei Lan, Tianjia Shao, Zixuan Lu, Yu Zhang, Chenfanfu Jiang, and Yin Yang. 3dgs^2: Near second-order converging 3d gaussian splatting. *arXiv preprint arXiv:2501.13975*, 2025. 3

[27] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002. 3

[28] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21719–21728, 2024. 2

[29] Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *International Conference on Learning Representations (ICLR)*, 2024. 5

[30] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024. 2

[31] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 1, 2, 7

[32] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 735–742, 2010. 3, 5

[33] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015. 5

[34] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1033–1040, 2011. 3, 5

[35] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18039–18048, 2024. 1

[36] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020. 1, 2, 6

[37] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*, pages 105–116. Springer, 2006. 3

[38] Jorge J Moré and David J Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):286–307, 1994. 3, 6

[39] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1

[40] 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. 2

[41] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *European Conference on Computer Vision (ECCV)*, 2024. 1, 2

[42] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 3, 6, 8

[43] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, pages 400–407, 1951. 2

[44] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting. In *European Conference on Computer Vision*, pages 347–362. Springer, 2024. 2

[45] Nicolas Roux, Pierre-antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2007. 5

[46] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988. 2, 7

[47] Sara Fridovich-Keil and 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)*, 2022. 1

[48] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 343–351, Atlanta, Georgia, USA, 2013. PMLR. 5

[49] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5459–5469, 2022. 2

[50] Kartik Teotia, Hyeongwoo Kim, Pablo Garrido, Marc Habermann, Mohamed Elgharib, and Christian Theobalt. Gaussianheads: End-to-end learning of drivable gaussian head avatars from coarse-to-fine representations. *ACM Transactions on Graphics (TOG)*, 43(6):1–12, 2024. 1

[51] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012. 2, 7

[52] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. 4, 7

[53] Robert Edwin Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964. 5

[54] Linning Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Aljaž Božič, Dahua Lin, Michael Zollhöfer, and Christian Richardt. VR-NeRF: High-fidelity virtualized walkable spaces. In *SIGGRAPH Asia Conference Proceedings*, 2023. 1

[55] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021. 5

[56] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19447–19456, 2024. 2

[57] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 7