3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

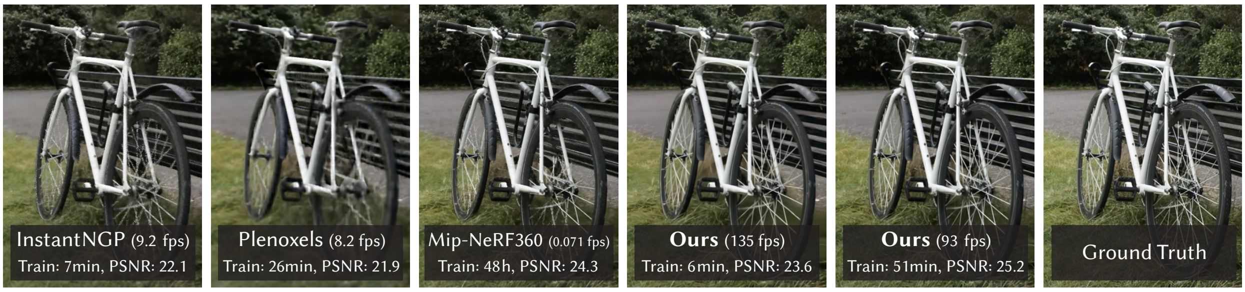

Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].

Radiance Field methods have recently revolutionized novel-view synthesis of scenes captured with multiple photos or videos. However, achieving high visual quality still requires neural networks that are costly to train and render, while recent faster methods inevitably trade off speed for quality. For unbounded and complete scenes (rather than isolated objects) and 1080p resolution rendering, no current method can achieve real-time display rates. We introduce three key elements that allow us to achieve state-of-the-art visual quality while maintaining competitive training times and importantly allow high-quality real-time (≥ 30 fps) novel-view synthesis at 1080p resolution. First, starting from sparse points produced during camera calibration, we represent the scene with 3D Gaussians that preserve desirable properties of continuous volumetric radiance fields for scene optimization while avoiding unnecessary computation in empty space; Second, we perform interleaved optimization/density control of the 3D Gaussians, notably optimizing anisotropic covariance to achieve an accurate representation of the scene; Third, we develop a fast visibility-aware rendering algorithm that supports anisotropic splatting and both accelerates training and allows real-time rendering. We demonstrate state-of-the-art visual quality and real-time rendering on several established datasets.

CCS Concepts: • **Computing methodologies → Rendering**; **Point-based models**; **Rasterization**; **Machine learning approaches**.

*Both authors contributed equally to the paper.

Authors' addresses: Bernhard Kerbl, bernhard.kerbl@inria.fr, Inria, Université Côte d'Azur, France; Georgios Kopanas, georgios.kopanas@inria.fr, Inria, Université Côte d'Azur, France; Thomas Leimkühler, thomas.leimkuehler@mpi-inf.mpg.de, Max-Planck-Institut für Informatik, Germany; George Drettakis, george.drettakis@inria.fr, Inria, Université Côte d'Azur, France.

0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592433

Additional Key Words and Phrases: novel view synthesis, radiance fields, 3D gaussians, real-time rendering

ACM Reference Format:
Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 14 pages. https://doi.org/10.1145/3592433

1 INTRODUCTION

Meshes and points are the most common 3D scene representations because they are explicit and are a good fit for fast GPU/CUDA-based rasterization. In contrast, recent Neural Radiance Field (NeRF) methods build on continuous scene representations, typically optimizing a Multi-Layer Perceptron (MLP) using volumetric ray-marching for novel-view synthesis of captured scenes. Similarly, the most efficient radiance field solutions to date build on continuous representations by interpolating values stored in, e.g., voxel [Fridovich-Keil and Yu et al. 2022] or hash [Müller et al. 2022] grids or points [Xu et al. 2022]. While the continuous nature of these methods helps optimization, the stochastic sampling required for rendering is costly and can result in noise. We introduce a new approach that combines the best of both worlds: our 3D Gaussian representation allows optimization with state-of-the-art (SOTA) visual quality and competitive training times, while our tile-based splatting solution ensures real-time rendering at SOTA quality for 1080p resolution on several previously published datasets [Barron et al. 2022; Hedman et al. 2018; Knapitsch et al. 2017] (see Fig. 1).

Our goal is to allow real-time rendering for scenes captured with multiple photos, and create the representations with optimization times as fast as the most efficient previous methods for typical real scenes. Recent methods achieve fast training [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022], but struggle to achieve the visual quality obtained by the current SOTA NeRF methods, i.e., Mip-NeRF360 [Barron et al. 2022], which requires up to 48 hours of training time. The fast – but lower-quality – radiance field methods can achieve interactive rendering times depending on the scene (10-15 frames per second), but fall short of real-time rendering at high resolution.

Our solution builds on three main components. We first introduce *3D Gaussians* as a flexible and expressive scene representation. We start with the same input as previous NeRF-like methods, i.e., cameras calibrated with Structure-from-Motion (SfM) [Snavely et al. 2006] and initialize the set of 3D Gaussians with the sparse point cloud produced for free as part of the SfM process. In contrast to most point-based solutions that require Multi-View Stereo (MVS) data [Aliev et al. 2020; Kopanas et al. 2021; Rückert et al. 2022], we achieve high-quality results with only SfM points as input. Note that for the NeRF-synthetic dataset, our method achieves high quality even with random initialization. We show that 3D Gaussians are an excellent choice, since they are a differentiable volumetric representation, but they can also be rasterized very efficiently by projecting them to 2D, and applying standard α-blending, using an equivalent image formation model as NeRF. The second component of our method is optimization of the properties of the 3D Gaussians – 3D position, opacity α, anisotropic covariance, and spherical harmonic (SH) coefficients – interleaved with adaptive density control steps, where we add and occasionally remove 3D Gaussians during optimization. The optimization procedure produces a reasonably compact, unstructured, and precise representation of the scene (1-5 million Gaussians for all scenes tested). The third and final element of our method is our real-time rendering solution that uses fast GPU sorting algorithms and is inspired by tile-based rasterization, following recent work [Lassner and Zollhofer 2021]. However, thanks to our 3D Gaussian representation, we can perform anisotropic splatting that respects visibility ordering – thanks to sorting and α-blending – and enable a fast and accurate backward pass by tracking the traversal of as many sorted splats as required.

To summarize, we provide the following contributions:

- The introduction of anisotropic 3D Gaussians as a high-quality, unstructured representation of radiance fields.
- An optimization method of 3D Gaussian properties, interleaved with adaptive density control that creates high-quality representations for captured scenes.
- A fast, differentiable rendering approach for the GPU, which is visibility-aware, allows anisotropic splatting and fast backpropagation to achieve high-quality novel view synthesis.

Our results on previously published datasets show that we can optimize our 3D Gaussians from multi-view captures and achieve equal or better quality than the best quality previous implicit radiance field approaches. We also can achieve training speeds and quality similar to the fastest methods and importantly provide the first *real-time rendering* with high quality for novel-view synthesis.

2 RELATED WORK

We first briefly overview traditional reconstruction, then discuss point-based rendering and radiance field work, discussing their similarity; radiance fields are a vast area, so we focus only on directly related work. For complete coverage of the field, please see the excellent recent surveys [Tewari et al. 2022; Xie et al. 2022].

2.1 Traditional Scene Reconstruction and Rendering

The first novel-view synthesis approaches were based on light fields, first densely sampled [Gortler et al. 1996; Levoy and Hanrahan 1996] then allowing unstructured capture [Buehler et al. 2001]. The advent of Structure-from-Motion (SfM) [Snavely et al. 2006] enabled an entire new domain where a collection of photos could be used to synthesize novel views. SfM estimates a sparse point cloud during camera calibration, that was initially used for simple visualization of 3D space. Subsequent multi-view stereo (MVS) produced impressive full 3D reconstruction algorithms over the years [Goesele et al. 2007], enabling the development of several view synthesis algorithms [Chaurasia et al. 2013; Eisemann et al. 2008; Hedman et al. 2018; Kopanas et al. 2021]. All these methods *re-project* and *blend* the input images into the novel view camera, and use the geometry to guide this re-projection. These methods produced excellent results in many cases, but typically cannot completely recover from unreconstructed regions, or from "over-reconstruction", when MVS generates inexistent geometry. Recent neural rendering algorithms [Tewari et al. 2022] vastly reduce such artifacts and avoid the overwhelming cost of storing all input images on the GPU, outperforming these methods on most fronts.

2.2 Neural Rendering and Radiance Fields

Deep learning techniques were adopted early for novel-view synthesis [Flynn et al. 2016; Zhou et al. 2016]; CNNs were used to estimate blending weights [Hedman et al. 2018], or for texture-space solutions [Riegler and Koltun 2020; Thies et al. 2019]. The use of MVS-based geometry is a major drawback of most of these methods; in addition, the use of CNNs for final rendering frequently results in temporal flickering.

Volumetric representations for novel-view synthesis were initiated by Soft3D [Penner and Zhang 2017]; deep-learning techniques coupled with volumetric ray-marching were subsequently proposed [Henzler et al. 2019; Sitzmann et al. 2019] building on a continuous differentiable density field to represent geometry. Rendering using volumetric ray-marching has a significant cost due to the large number of samples required to query the volume. Neural Radiance Fields (NeRFs) [Mildenhall et al. 2020] introduced importance sampling and positional encoding to improve quality, but used a large Multi-Layer Perceptron negatively affecting speed. The success of NeRF has resulted in an explosion of follow-up methods that address quality and speed, often by introducing regularization strategies; the current state-of-the-art in image quality for novel-view synthesis is Mip-NeRF360 [Barron et al. 2022]. While the rendering quality is outstanding, training and rendering times remain extremely high; we are able to equal or in some cases surpass this quality while providing fast training and real-time rendering.

The most recent methods have focused on faster training and/or rendering mostly by exploiting three design choices: the use of spatial data structures to store (neural) features that are subsequently interpolated during volumetric ray-marching, different encodings, and MLP capacity. Such methods include different variants of space discretization [Chen et al. 2022b,a; Fridovich-Keil and Yu et al. 2022; Garbin et al. 2021; Hedman et al. 2021; Reiser et al. 2021; Takikawa et al. 2021; Wu et al. 2022; Yu et al. 2021], codebooks [Takikawa et al. 2022], and encodings such as hash tables [Müller et al. 2022], allowing the use of a smaller MLP or foregoing neural networks completely [Fridovich-Keil and Yu et al. 2022; Sun et al. 2022].

Most notable of these methods are InstantNGP [Müller et al. 2022] which uses a hash grid and an occupancy grid to accelerate computation and a smaller MLP to represent density and appearance; and Plenoxels [Fridovich-Keil and Yu et al. 2022] that use a sparse voxel grid to interpolate a continuous density field, and are able to forgo neural networks altogether. Both rely on Spherical Harmonics: the former to represent directional effects directly, the latter to encode its inputs to the color network. While both provide outstanding results, these methods can still struggle to represent empty space effectively, depending in part on the scene/capture type. In addition, image quality is limited in large part by the choice of the structured grids used for acceleration, and rendering speed is hindered by the need to query many samples for a given ray-marching step. The unstructured, explicit GPU-friendly 3D Gaussians we use achieve faster rendering speed and better quality *without* neural components.

2.3 Point-Based Rendering and Radiance Fields

Point-based methods efficiently render disconnected and unstructured geometry samples (i.e., point clouds) [Gross and Pfister 2011]. In its simplest form, point sample rendering [Grossman and Dally 1998] rasterizes an unstructured set of points with a fixed size, for which it may exploit natively supported point types of graphics APIs [Sainz and Pajarola 2004] or parallel software rasterization on the GPU [Laine and Karras 2011; Schütz et al. 2022]. While true to the underlying data, point sample rendering suffers from holes, causes aliasing, and is strictly discontinuous. Seminal work on high-quality point-based rendering addresses these issues by "splatting" point primitives with an extent larger than a pixel, e.g., circular or elliptic discs, ellipsoids, or surfels [Botsch et al. 2005; Pfister et al. 2000; Ren et al. 2002; Zwicker et al. 2001b].

There has been recent interest in *differentiable* point-based rendering techniques [Wiles et al. 2020; Yifan et al. 2019]. Points have been augmented with neural features and rendered using a CNN [Aliev et al. 2020; Rückert et al. 2022] resulting in fast or even real-time view synthesis; however they still depend on MVS for the initial geometry, and as such inherit its artifacts, most notably over- or under-reconstruction in hard cases such as featureless/shiny areas or thin structures.

Point-based α-blending and NeRF-style volumetric rendering share essentially the same image formation model. Specifically, the color C is given by volumetric rendering along a ray:

$$C = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \quad \text{with} \quad T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right), \quad (1)$$

where samples of density σ, transmittance T, and color $\mathbf{c}$ are taken along the ray with intervals δ_i. This can be re-written as

$$C = \sum_{i=1}^{N} T_i\alpha_i\mathbf{c}_i, \quad (2)$$

with

$$\alpha_i = (1 - \exp(-\sigma_i\delta_i)) \text{ and } T_i = \prod_{j=1}^{i-1}(1 - \alpha_i).$$

A typical neural point-based approach (e.g., [Kopanas et al. 2022, 2021]) computes the color C of a pixel by blending $\mathcal{N}$ ordered points overlapping the pixel:

$$C = \sum_{i\in\mathcal{N}} c_i\alpha_i\prod_{j=1}^{i-1}(1 - \alpha_j), \quad (3)$$

where $\mathbf{c}_i$ is the color of each point and α_i is given by evaluating a 2D Gaussian with covariance Σ [Yifan et al. 2019] multiplied with a learned per-point opacity.

From Eq. 2 and Eq. 3, we can clearly see that the image formation model is the same. However, the rendering algorithm is very different. NeRFs are a continuous representation implicitly representing empty/occupied space; expensive random sampling is required to find the samples in Eq. 2 with consequent noise and computational expense. In contrast, points are an unstructured, discrete representation that is flexible enough to allow creation, destruction, and displacement of geometry similar to NeRF. This is achieved by optimizing opacity and positions, as shown by previous work [Kopanas et al. 2021], while avoiding the shortcomings of a full volumetric representation.

Pulsar [Lassner and Zollhofer 2021] achieves fast *sphere* rasterization which inspired our tile-based and sorting renderer. However, given the analysis above, we want to maintain (approximate) conventional α-blending on sorted splats to have the advantages of volumetric representations: Our rasterization respects visibility order in contrast to their order-independent method. In addition, we backpropagate gradients on all splats in a pixel and rasterize anisotropic splats. These elements all contribute to the high visual quality of our results (see Sec. 7.3). In addition, previous methods mentioned above also use CNNs for rendering, which results in temporal instability. Nonetheless, the rendering speed of Pulsar [Lassner and Zollhofer 2021] and ADOP [Rückert et al. 2022] served as motivation to develop our fast rendering solution.

While focusing on specular effects, the diffuse point-based rendering track of Neural Point Catacaustics [Kopanas et al. 2022] overcomes this temporal instability by using an MLP, but still required MVS geometry as input. The most recent method [Zhang et al. 2022] in this category does not require MVS, and also uses SH for directions; however, it can only handle scenes of one object and needs masks for initialization. While fast for small resolutions and low point counts, it is unclear how it can scale to scenes of typical datasets [Barron et al. 2022; Hedman et al. 2018; Knapitsch et al. 2017]. We use 3D Gaussians for a more flexible scene representation, avoiding the need for MVS geometry and achieving real-time rendering thanks to our tile-based rendering algorithm for the projected Gaussians.

A recent approach [Xu et al. 2022] uses points to represent a radiance field with a radial basis function approach. They employ point pruning and densification techniques during optimization, but use volumetric ray-marching and cannot achieve real-time display rates.

In the domain of human performance capture, 3D Gaussians have been used to represent captured human bodies [Rhodin et al. 2015; Stoll et al. 2011]; more recently they have been used with volumetric ray-marching for vision tasks [Wang et al. 2023]. Neural volumetric primitives have been proposed in a similar context [Lombardi et al. 2021]. While these methods inspired the choice of 3D Gaussians as our scene representation, they focus on the specific case of reconstructing and rendering a single isolated object (a human body or face), resulting in scenes with small depth complexity. In contrast, our optimization of *anisotropic* covariance, our interleaved optimization/density control, and efficient depth sorting for rendering allow us to handle complete, complex scenes including background, both indoors and outdoors and with large depth complexity.

3 OVERVIEW

The input to our method is a set of images of a static scene, together with the corresponding cameras calibrated by SfM [Schönberger and Frahm 2016] which produces a sparse point cloud as a side-effect. From these points we create a set of 3D Gaussians (Sec. 4), defined by a position (mean), covariance matrix and opacity α, that allows a very flexible optimization regime. This results in a reasonably compact representation of the 3D scene, in part because highly anisotropic volumetric splats can be used to represent fine structures compactly. The directional appearance component (color) of the radiance field is represented via spherical harmonics (SH), following standard practice [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Our algorithm proceeds to create the radiance field representation (Sec. 5) via a sequence of optimization steps of 3D Gaussian parameters, i.e., position, covariance, α and SH coefficients interleaved with operations for adaptive control of the Gaussian density. The key to the efficiency of our method is our tile-based rasterizer (Sec. 6) that allows α-blending of anisotropic splats, respecting visibility order thanks to fast sorting. Out fast rasterizer also includes a fast backward pass by tracking accumulated α values, without a limit on the number of Gaussians that can receive gradients. The overview of our method is illustrated in Fig. 2.

4 DIFFERENTIABLE 3D GAUSSIAN SPLATTING

Our goal is to optimize a scene representation that allows high-quality novel view synthesis, starting from a sparse set of (SfM) points without normals. To do this, we need a primitive that inherits the properties of differentiable volumetric representations, while at the same time being unstructured and explicit to allow very fast rendering. We choose 3D Gaussians, which are differentiable and can be easily projected to 2D splats allowing fast α-blending for rendering.

Our representation has similarities to previous methods that use 2D points [Kopanas et al. 2021; Yifan et al. 2019] and assume each point is a small planar circle with a normal. Given the extreme sparsity of SfM points it is very hard to estimate normals. Similarly, optimizing very noisy normals from such an estimation would be very challenging. Instead, we model the geometry as a set of 3D Gaussians that do not require normals. Our Gaussians are defined by a full 3D covariance matrix Σ defined in world space [Zwicker et al. 2001a] centered at point (mean) μ:

$$G(x) = e^{-\frac{1}{2}(x)^T\Sigma^{-1}(x)} \tag{4}$$

. This Gaussian is multiplied by α in our blending process.

However, we need to project our 3D Gaussians to 2D for rendering. Zwicker et al. [2001a] demonstrate how to do this projection to image space. Given a viewing transformation W the covariance matrix Σ' in camera coordinates is given as follows:

$$\Sigma' = JW\,\Sigma\,W^TJ^T \tag{5}$$

where J is the Jacobian of the affine approximation of the projective transformation. Zwicker et al. [2001a] also show that if we skip the third row and column of Σ', we obtain a 2×2 variance matrix with the same structure and properties as if we would start from planar points with normals, as in previous work [Kopanas et al. 2021].

An obvious approach would be to directly optimize the covariance matrix Σ to obtain 3D Gaussians that represent the radiance field. However, covariance matrices have physical meaning only when they are positive semi-definite. For our optimization of all our parameters, we use gradient descent that cannot be easily constrained to produce such valid matrices, and update steps and gradients can very easily create invalid covariance matrices.

As a result, we opted for a more intuitive, yet equivalently expressive representation for optimization. The covariance matrix Σ of a 3D Gaussian is analogous to describing the configuration of an ellipsoid. Given a scaling matrix S and rotation matrix R, we can find the corresponding Σ:

$$\Sigma = RSS^TR^T \tag{6}$$

To allow independent optimization of both factors, we store them separately: a 3D vector s for scaling and a quaternion q to represent rotation. These can be trivially converted to their respective matrices and combined, making sure to normalize q to obtain a valid unit quaternion.

To avoid significant overhead due to automatic differentiation during training, we derive the gradients for all parameters explicitly. Details of the exact derivative computations are in appendix A.

This representation of anisotropic covariance – suitable for optimization – allows us to optimize 3D Gaussians to adapt to the geometry of different shapes in captured scenes, resulting in a fairly compact representation. Fig. 3 illustrates such cases.

5 OPTIMIZATION WITH ADAPTIVE DENSITY CONTROL OF 3D GAUSSIANS

The core of our approach is the optimization step, which creates a dense set of 3D Gaussians accurately representing the scene for free-view synthesis. In addition to positions p, α, and covariance Σ, we also optimize SH coefficients representing color c of each Gaussian to correctly capture the view-dependent appearance of the scene. The optimization of these parameters is interleaved with steps that control the density of the Gaussians to better represent the scene.

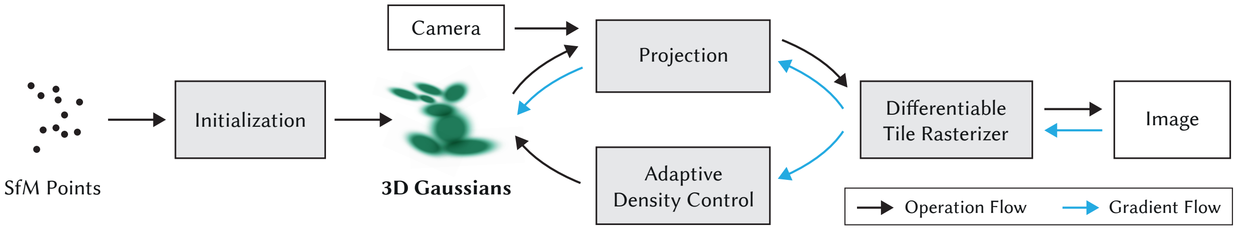

Fig. 2. Optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. We then optimize and adaptively control the density of this set of Gaussians. During optimization we use our fast tile-based renderer, allowing competitive training times compared to SOTA fast radiance field methods. Once trained, our renderer allows real-time navigation for a wide variety of scenes.

Fig. 3. We visualize the 3D Gaussians after optimization by shrinking them 60% (far right). This clearly shows the anisotropic shapes of the 3D Gaussians that compactly represent complex geometry after optimization. Left the actual rendered image.

5.1 Optimization

The optimization is based on successive iterations of rendering and comparing the resulting image to the training views in the captured dataset. Inevitably, geometry may be incorrectly placed due to the ambiguities of 3D to 2D projection. Our optimization thus needs to be able to *create* geometry and also *destroy* or *move* geometry if it has been incorrectly positioned. The quality of the parameters of the covariances of the 3D Gaussians is critical for the compactness of the representation since large homogeneous areas can be captured with a small number of large anisotropic Gaussians.

We use Stochastic Gradient Descent techniques for optimization, taking full advantage of standard GPU-accelerated frameworks, and the ability to add custom CUDA kernels for some operations, following recent best practice [Fridovich-Keil and Yu et al. 2022; Sun et al. 2022]. In particular, our fast rasterization (see Sec. 6) is critical in the efficiency of our optimization, since it is the main computational bottleneck of the optimization.

We use a sigmoid activation function for α to constrain it in the $[0-1)$ range and obtain smooth gradients, and an exponential activation function for the scale of the covariance for similar reasons.

We estimate the initial covariance matrix as an isotropic Gaussian with axes equal to the mean of the distance to the closest three points. We use a standard exponential decay scheduling technique similar to Plenoxels [Fridovich-Keil and Yu et al. 2022], but for positions only. The loss function is $\mathcal{L}_1$ combined with a D-SSIM term:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}} \tag{7}$$

We use $\lambda = 0.2$ in all our tests. We provide details of the learning schedule and other elements in Sec. 7.1.

5.2 Adaptive Control of Gaussians

We start with the initial set of sparse points from SfM and then apply our method to adaptively control the number of Gaussians and their density over unit volume[1], allowing us to go from an initial sparse set of Gaussians to a denser set that better represents the scene, and with correct parameters. After optimization warm-up (see Sec. 7.1), we densify every 100 iterations and remove any Gaussians that are essentially transparent, i.e., with α less than a threshold ϵ_α.

Our adaptive control of the Gaussians needs to populate empty areas. It focuses on regions with missing geometric features ("under-reconstruction"), but also in regions where Gaussians cover large areas in the scene (which often correspond to "over-reconstruction"). We observe that both have *large* view-space positional gradients. Intuitively, this is likely because they correspond to regions that are not yet well reconstructed, and the optimization tries to move the Gaussians to correct this.

Since both cases are good candidates for densification, we densify Gaussians with an average magnitude of view-space position gradients above a threshold τ_{pos}, which we set to 0.0002 in our tests.

We next present details of this process, illustrated in Fig. 4.

For small Gaussians that are in under-reconstructed regions, we need to cover the new geometry that must be created. For this, it is preferable to clone the Gaussians, by simply creating a copy of the same size, and moving it in the direction of the positional gradient.

On the other hand, large Gaussians in regions with high variance need to be split into smaller Gaussians. We replace such Gaussians by two new ones, and divide their scale by a factor of $\phi = 1.6$ which we determined experimentally. We also initialize their position by using the original 3D Gaussian as a PDF for sampling.

In the first case we detect and treat the need for increasing both the total volume of the system and the number of Gaussians, while in the second case we conserve total volume but increase the number of Gaussians. Similar to other volumetric representations, our optimization can get stuck with floaters close to the input cameras; in our case this may result in an unjustified increase in the Gaussian density. An effective way to moderate the increase in the number of Gaussians is to set the α value close to zero every $N = 3000$

[1] Density of Gaussians should not be confused of course with density σ in the NeRF literature.

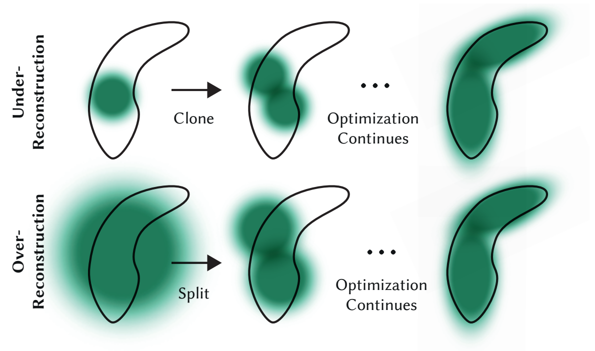

Fig. 4. Our adaptive Gaussian densification scheme. *Top row (under-reconstruction)*: When small-scale geometry (black outline) is insufficiently covered, we clone the respective Gaussian. *Bottom row (over-reconstruction)*: If small-scale geometry is represented by one large splat, we split it in two.

iterations. The optimization then increases the α for the Gaussians where this is needed while allowing our culling approach to remove Gaussians with α less than ϵ_α as described above. Gaussians may shrink or grow and considerably overlap with others, but we periodically remove Gaussians that are very large in worldspace and those that have a big footprint in viewspace. This strategy results in overall good control over the total number of Gaussians. The Gaussians in our model remain primitives in Euclidean space at all times; unlike other methods [Barron et al. 2022; Fridovich-Keil and Yu et al. 2022], we do not require space compaction, warping or projection strategies for distant or large Gaussians.

6 FAST DIFFERENTIABLE RASTERIZER FOR GAUSSIANS

Our goals are to have fast overall rendering and fast sorting to allow approximate α-blending – including for anisotropic splats – and to avoid hard limits on the number of splats that can receive gradients that exist in previous work [Lassner and Zollhofer 2021].

To achieve these goals, we design a tile-based rasterizer for Gaussian splats inspired by recent software rasterization approaches [Lassner and Zollhofer 2021] to pre-sort primitives for an entire image at a time, avoiding the expense of sorting per pixel that hindered previous α-blending solutions [Kopanas et al. 2022, 2021]. Our fast rasterizer allows efficient backpropagation over an arbitrary number of blended Gaussians with low additional memory consumption, requiring only a constant overhead per pixel. Our rasterization pipeline is fully differentiable, and given the projection to 2D (Sec. 4) can rasterize anisotropic splats similar to previous 2D splatting methods [Kopanas et al. 2021].

Our method starts by splitting the screen into 16×16 tiles, and then proceeds to cull 3D Gaussians against the view frustum and each tile. Specifically, we only keep Gaussians with a 99% confidence interval intersecting the view frustum. Additionally, we use a guard band to trivially reject Gaussians at extreme positions (i.e., those with means close to the near plane and far outside the view frustum), since computing their projected 2D covariance would be unstable. We then instantiate each Gaussian according to the number of tiles they overlap and assign each instance a key that combines view space depth and tile ID. We then sort Gaussians based on these keys using a single fast GPU Radix sort [Merrill and Grimshaw 2010]. Note that there is no additional per-pixel ordering of points, and blending is performed based on this initial sorting. As a consequence, our α-blending can be approximate in some configurations. However, these approximations become negligible as splats approach the size of individual pixels. We found that this choice greatly enhances training and rendering performance without producing visible artifacts in converged scenes.

After sorting Gaussians, we produce a list for each tile by identifying the first and last depth-sorted entry that splats to a given tile. For rasterization, we launch one thread block for each tile. Each block first collaboratively loads packets of Gaussians into shared memory and then, for a given pixel, accumulates color and α values by traversing the lists front-to-back, thus maximizing the gain in parallelism both for data loading/sharing and processing. When we reach a target saturation of α in a pixel, the corresponding thread stops. At regular intervals, threads in a tile are queried and the processing of the entire tile terminates when all pixels have saturated (i.e., α goes to 1). Details of sorting and a high-level overview of the overall rasterization approach are given in Appendix C.

During rasterization, the saturation of α is the only stopping criterion. In contrast to previous work, we do not limit the number of blended primitives that receive gradient updates. We enforce this property to allow our approach to handle scenes with an arbitrary, varying depth complexity and accurately learn them, without having to resort to scene-specific hyperparameter tuning. During the backward pass, we must therefore recover the full sequence of blended points per-pixel in the forward pass. One solution would be to store arbitrarily long lists of blended points per-pixel in global memory [Kopanas et al. 2021]. To avoid the implied dynamic memory management overhead, we instead choose to traverse the per-tile lists again; we can reuse the sorted array of Gaussians and tile ranges from the forward pass. To facilitate gradient computation, we now traverse them back-to-front.

The traversal starts from the last point that affected any pixel in the tile, and loading of points into shared memory again happens collaboratively. Additionally, each pixel will only start (expensive) overlap testing and processing of points if their depth is lower than or equal to the depth of the last point that contributed to its color during the forward pass. Computation of the gradients described in Sec. 4 requires the accumulated opacity values at each step during the original blending process. Rather than trasversing an explicit list of progressively shrinking opacities in the backward pass, we can recover these intermediate opacities by storing only the total accumulated opacity at the end of the forward pass. Specifically, each point stores the final accumulated opacity α in the forward process; we divide this by each point's α in our back-to-front traversal to obtain the required coefficients for gradient computation.

7 IMPLEMENTATION, RESULTS AND EVALUATION

We next discuss some details of implementation, present results and the evaluation of our algorithm compared to previous work and ablation studies.