TU WIEN Informatics

Depth-Guided Direct 3D Gaussian Splat Initialization

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Attila Deák
Registration Number 12123670

to the Faculty of Informatics

at the TU Wien

Advisor: Projektass. Mag.rer.soc.oec. Stefan Ohrhallinger, PhD

Vienna, January 1, 2001

Attila Deák

Stefan Ohrhallinger

Declaration of Authorship

Attila Deák

I hereby declare that I have written this thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work – including tables, maps and figures – which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, January 1, 2001

Attila Deák

Danksagung

Ich möchte mich bei meinem Betreuer, Stefan Ohrhallinger, für seine konstante Unterstützung und Weisung bedanken, die ich im Laufe dieser Arbeit erhalten durfte.

Ich möchte mich außerdem bei meinem Kollegen und persönlichen Freund Jakub Nawrocki bedanken, der im Rahmen seiner eigenen Arbeit zur Oberflächenrekonstruktion mit Hilfe von Relightable3DGaussian den ursprünglichen Anstoß und die Motivation zu dieser Arbeit gab.

Weiters möchte ich mich bei den Technikern und Technikerinnen des Instituts für Visual Computing und Human-Centered Technology an der Technischen Universität Wien bedanken, die uns Hardware und Systeme zur Verfügung gestellt haben, die wir für unsere Experimente und Evaluierungen verwendet haben.

Zu guter Letzt möchte ich mich bei meiner Familie bedanken, deren Unterstützung mir dieses Studium ermöglicht hat.

Acknowledgements

I want to thank my supervisor, Stefan Ohrhallinger, for his continuous support and guidance throughout this thesis.

I would also like to thank my colleague and personal friend Jakub Nawrocki, who originally suggested the idea for this thesis and whose work on surface reconstruction using Relightable3DGaussian provided the initial motivation for our contribution.

Further on, I would like to thank the technicians of the Institute of Visual Computing and Human-Centered Technology at the Technical University of Vienna for their technical support regarding the hardware and systems on which we conducted our experiments and evaluations.

Finally, I would like to thank my family for their support throughout my studies.

Kurzfassung

3D Gaussian Splatting (3DGS) von Kerbl u. a. [KKLD23] ist ein radianzfeldbasiertes Verfahren, das eine vergleichsweise schnelle Novel-View-Synthese ermöglicht. Konventionelle 3DGS-Rekonstruktionspipelines können jedoch lange Optimierungsphasen erfordern, insbesondere aufgrund der aufwendigen Verdichtung einer initialen Gaussian-Repräsentation, die aus einer spärlichen Punktwolke erzeugt wurde.

Wir zielen darauf ab, dieses Problem zu verringern, indem wir den Direct-Sampling-Ansatz von Meuleman u. a. [MSL+25] isolieren und in die Relightable3DGaussian (R3DGS)-Pipeline von Gao u. a. [GGL+24] integrieren, deren Struktur dem ursprünglichen 3DGS-System ähnlich ist. Zusätzlich erweitern wir die Direct-Sampling-Methode um Strategien wie eine COLMAP-Abdeckungskarte sowie um neue Tiefenkorrekturverfahren mittels k-nächster-Nachbarn-Suche. Anschließend evaluieren wir unsere Lösung anhand von drei Datensätzen, nämlich Neural Radiance Field (NeRF) Synthetic, DTU und Tanks and Temples (TnT).

Unsere Ergebnisse zeigen für alle drei Datensätze im Verlauf der Optimierungsiterationen eine höhere peak signal-to-noise ratio (PSNR) als die Baseline. Über den zeitlichen Verlauf zeigt sich ebenfalls eine höhere PSNR für NeRF Synthetic und DTU, während für TnT keine entsprechende Verbesserung beobachtet wird. Als sekundäres Ergebnis beobachten wir, dass unser Ansatz die Anzahl der Gaussian-Primitiven, die zur Repräsentation bestimmter Szenen erforderlich sind, am Ende des Trainings reduziert und dadurch zu einer kompakteren Repräsentation führt. Dieses Verhalten variiert jedoch stark zwischen den einzelnen Szenen und kann, wie im Fall von TnT, auch den gegenteiligen Effekt haben, sodass im Vergleich zur Baseline mehr Gaussian-Primitiven benötigt werden.

Diese Ergebnisse deuten darauf hin, dass unser von Meuleman u. a. [MSL+25] adaptierter Direct-Sampling-Initialisierungsansatz eine vielversprechende Strategie zur Initialisierung von Gaussian-Rekonstruktionen darstellt, deren praktischer Nutzen jedoch stark von den Eigenschaften der jeweiligen Datensätze und Szenen abhängt.

Abstract

3DGS by Kerbl et al. [KKLD23] is a radiance-field solution that enables comparatively fast novel-view synthesis, but conventional 3DGS reconstruction pipelines sometimes rely on lengthy optimization stages, particularly the costly densification of an initial Gaussian representation created from a sparse point cloud. We aim to alleviate this by isolating and integrating the direct sampling approach by Meuleman et al. [MSL+25] into the R3DGS pipeline of Gao et al. [GGL+24], which has a structure similar to the original 3DGS system. We further refine the direct sampling method using strategies such as a COLMAP coverage map and new depth correction approaches via k-nearest-neighbor search. We then evaluate our solution on three datasets, namely NeRF Synthetic, DTU, and TnT. Our results show an increased PSNR over the course of optimization steps for all three datasets compared to the baseline and an increased PSNR over time for NeRF Synthetic and DTU in contrast to TnT, where no improvement in PSNR over time is observed for our implementation. Furthermore, we made a secondary finding that our approach reduced the number of Gaussians required to represent certain scenes at the end of training, thereby making the representation more compact. This phenomenon varies widely from scene to scene and sometimes, as in the case of TnT, has the opposite effect, requiring more Gaussians to represent the scene compared to the baseline.

These results indicate that our direct sampling initialization approach, adapted from Meuleman et al. [MSL+25], is a promising Gaussian initialization strategy. However, its practicality strongly depends on the input characteristics of the datasets and their scenes.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Motivation 1
1.2 Contribution 2
1.3 Thesis structure 4

2 Background **5**
2.1 Gaussian Splatting 5
2.2 Monocular Depth Estimation 9
2.3 Structure from Motion 9
2.4 On-the-fly Reconstruction for Large-Scale Novel View Synthesis from Unposed Images 11

3 Methods **15**
3.1 System overview 15
3.2 Placing of Gaussian Primitives 15
3.3 Depth estimation 19
3.4 Covariance estimation 22

4 Evaluation **25**
4.1 Setup 25
4.2 Results. 26

5 Conclusion **33**
5.1 Discussion 34
5.2 Limitations 35
5.3 Future Work 35

A Evaluation Results **39**

A.1 PSNR Results 39
A.2 SSIM and LPIPS Results 49
A.3 Gaussian Count Results 50
A.4 Runtime Breakdown 55

B Evaluation Parameters **59**

C Experimental Environment **61**

Overview of Generative AI Tools Used **63**

List of Figures **65**

List of Tables **67**

List of Algorithms **69**

Acronyms **71**

Bibliography **73**

CHAPTER 1

Introduction

1.1 Motivation

Novel view synthesis aims to reconstruct a scene from a set of images that capture it, so that newly rendered images from viewpoints different from the original set correctly represent the scene.

Recent developments like Neural Radiance Fields (NeRFs) have successfully tackled this issue but remain infeasible due to their high computational cost. On the other hand, 3D Gaussian Splatting (3DGS) by Kerbl et al. [KKLD23] introduces an explicit radiance-field representation based on anisotropic three-dimensional Gaussian primitives, which, thanks to their fast and differentiable rasterizer, provides a feasible solution to the Novel view synthesis problem, with comparatively short optimization times and real-time rendering of the final reconstruction. This approach consists of a pipeline that typically begins by initializing a set of Gaussian primitives from a sparse point cloud obtained via Structure-from-Motion (SfM), whose properties, such as color, scaling, orientation, and position, are subsequently optimized to most accurately represent the scene. A central component of the optimization stage of 3DGS is its adaptive density control, where, according to certain criteria, Gaussian splats can be duplicated, split, or, if misplaced, completely removed from the scene. This allows Kerbl et al. [KKLD23] to go from the aforementioned initial sparse cloud of Gaussian splats to a much denser distribution of Gaussian primitives, which can represent the target scene with meaningful quality. This also allows them to circumvent dense point clouds produced by multi-view stereo, avoiding its shortcomings.

To avoid the time-consuming process of densifying the reconstruction scene from sparse point clouds, Meuleman et al. [MSL+25] introduce their on-the-fly solution. In this alternative approach, no prior sparse point cloud is required, and Gaussian primitives are directly sampled in high-frequency regions of the training images. This is done by

computing the Laplacian of Gaussian (LoG) response of the input image and treating the result as the probability that a Gaussian will be spawned at the particular pixel. These two-dimensionally sampled Gaussians are then unprojected into scene space using a Monocular Depth Estimator, Depth Anything V2 (DAV2) [YKH+24]. In contrast to conventional offline approaches, Meuleman et al. [MSL+25] is designed to handle an incoming sequential stream of unposed training images, jointly placing and incrementally optimizing their corresponding Gaussians.

The above-mentioned direct sampling approach by Meuleman et al. [MSL+25] is one component of their overall system and is strongly interleaved with their incremental joint estimation of poses and the radiance field. Inspecting this overall procedure, we made some observations. First, their approach introduces redundant steps in conventional offline 3DGS pipelines, where poses are known before optimization and sparse point clouds are available, thereby enabling correction of depth errors produced by depth estimators; second, Meuleman et al. [MSL+25]'s on-the-fly approach relies on a sequential stream of input images, due to its incremental pose and reconstruction pipeline, which limits its use.

From the aforementioned approaches of Kerbl et al. [KKLD23] and Meuleman et al. [MSL+25], we can compare two procedures with certain characteristics. On the one hand, we have the on-the-fly approach of Meuleman et al. [MSL+25], which uses direct sampling and joint pose estimation. On the other hand, we have the conventional reconstruction pipeline of Kerbl et al. [KKLD23], which requires known prior poses along with a sparse point cloud. The latter approach could greatly benefit from an adapted direct sampling approach from Meuleman et al. [MSL+25], without the unnecessary stages that involve joint pose estimation alongside the initial fast pose estimates. A successful, isolated integration of the direct sampling approach into a conventional 3DGS pipeline should lead to a measurable improvement in reconstruction quality achieved in fewer time steps and iterations. This potential separation of the direct sampling method from the rest of the incremental on-the-fly pipeline would also make it possible to use unordered sequences of input images for training, removing a limitation from Meuleman et al. [MSL+25]'s approach.

1.2 Contribution

The goal of this thesis is to adapt and integrate the direct sampling method of the on-the-fly approach by Meuleman et al. [MSL+25] into a more general Gaussian reconstruction pipeline, namely the Relightable3DGaussian (R3DGS) [GGL+24] pipeline, which extends the original 3DGS process by the aforementioned properties, but is nonetheless in structure similar to the original 3DGS pipeline by Kerbl et al. [KKLD23]. We do this by introducing a new initial Gaussian spawning stage, replacing the common sparse point cloud initialization or, in the absence of background, the uniform distribution of Gaussian primitives across the reconstruction scene. By doing this, we avoid unnecessary stages of the on-the-fly approach, such as initial pose estimation and joint optimization of poses

and the radiance field.

The Gaussians are sampled according to the LoG, as in Meuleman et al. [MSL+25], thereby favoring high-frequency regions, which are placed into the three-dimensional scene using relative DAV2 depth estimates. As in the original paper of Meuleman et al. [MSL+25], we adjust these relative depth estimates using a procedure described in [KMK+24], which we refer to as mean absolute deviation from the median (MADM) in this thesis. This depth adjustment procedure requires discrete depth values as close to ground truth as possible. In contrast to Meuleman et al. [MSL+25], who reconstruct scene points on the fly, we use previously reconstructed COLMAP scene points for this purpose. To further enhance the quality of the depth adjustment, we introduce an additional step in which we perform a nearest-neighbor search on the aforementioned COLMAP scene points for each Gaussian, thereby correcting its assumed depth error. This allows us to unproject the sampled Gaussians and place them correctly in the scene. After this initialization stage, we continue with the Gaussian optimization as is state of the art. By doing this, we hope to accelerate the convergence of peak signal-to-noise ratio (PSNR) across optimization iterations and physical time. We evaluate our results against a baseline produced by an R3DGS implementation on three datasets commonly used in the novel view synthesis community: NeRF Synthetic [MST+21], DTU [JDV+14], and Tanks and Temples (TnT) [KPZK17].

We used the R3DGS project [GGL+24] as our codebase, since the motivation for this thesis stemmed from an existing R3DGS implementation that exhibited bottlenecks due to long densification times. The original 3DGS paper was limited to pure novel view synthesis and did not allow relighting the scene under different conditions. R3DGS expands on the idea by introducing additional properties such as bidirectional reflectance distribution function (BRDF) values and normals to use physically based rendering (PBR) to obtain the actual colors of each Gaussian before the α-blending.

To summarize, the main contributions of this thesis are:

1. Adapt the direct image space-based Gaussian sampling approach of Meuleman et al. [MSL+25], decoupled from other components of their on-the-fly reconstruction framework, into the R3DGS pipeline by Gao et al. [GGL+24], which retains the conventional Gaussian reconstruction structure of Kerbl et al. [KKLD23] and thus allows us to use unordered sequences of images for training.

2. Extend the methodology with COLMAP-supported depth adjustments and additional sampling strategies to increase PSNR across optimization steps and physical time.

We quantitatively evaluate whether our additional initialization stage improves the convergence of PSNR across optimization steps and physical time.

1.3 Thesis structure

The remainder of the thesis is structured as follows: Chapter 2 introduces the papers on which this thesis is based. First, we have 3DGS [KKLD23] and R3DGS [GGL+24]. These are the fundamental processes that we want to modify. Next, we have the Monocular Depth Estimator DAV2 [YKH+24], an essential tool for the correct placement of sampled Gaussians. This is followed by a SfM section of COLMAP [SF16]. In the final section, we introduce the on-the-fly approach by Meuleman et al. [MSL+25]. Chapter 3 describes our proposed direct sampling initialization in detail, based on the work of Meuleman et al. [MSL+25]. Chapter 4 sets clear evaluation details and compares our direct sampling initialization approach to the R3DGS baseline. Chapter 5 discusses findings and limitations of our solution and presents possibilities for future work.

CHAPTER 2

Background

In this chapter, we examine the already existing work that forms the foundation of our contribution to the topic. We split it into roughly four sections.

The first section handles Radiance Field solutions for novel-view synthesis, where we present the 3DGS approach by Kerbl et al. [KKLD23], which, at its core, is the process we want to speed up. This section also contains the extended 3DGS approach by Gao et al. [GGL+24], called R3DGS, which aims to incorporate physically based properties into the training process and is the codebase we extend.
An overview of the original Gaussian splatting pipeline can be seen in Figure 2.1

Next, we introduce DAV2 [YKH+24], a powerful monocular depth estimator, which we use to estimate the depth of the initially spawned Gaussians.

The third section discusses SfM with COLMAP [SF16]. SfM is necessary to create a point cloud containing key features from the training images, which we can later use to adjust our relative depth maps generated by DAV2.

Finally, we introduce the approach of Meuleman et al. [MSL+25], an on-the-fly method for reconstructing large-scale novel view synthesis from unposed images. Their idea of sampling Gaussian means according to high- and low-frequency regions is the core idea of this thesis, applying the idea to the conventional 3DGS structure in the form of R3DGS [GGL+24], requiring known camera poses and key features before the sampling.

2.1 Gaussian Splatting

Gaussian Scene Representation. 3DGS by Kerbl et al. [KKLD23] represents a reconstructed scene by three-dimensional normal distributions, called Gaussian splats, Gaussian primitives, or simply Gaussians. These Gaussian primitives are centered at a mean μ, with their orientations and axis-wise alignments determined by their anisotropic

covariance matrix Σ. For a displacement x from the Gaussian mean μ we thus get the Gaussian function $G(x)$:

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)} \tag{2.1}$$

Using this representation, an issue arises linked to the covariance matrix. It only has a geometric meaning when it is positive semidefinite, and optimizing the covariance matrix directly via gradient descent, for which there are no obvious constraints at hand in this case, does not guarantee it staying that way. Kerbl et al. [KKLD23] opt for an equivalent expression of the covariance matrix by decomposing it into a rotation matrix R and scaling matrix S. Given these matrices, one can obtain the corresponding Σ via:

$$\Sigma = RSS^T R^T \tag{2.2}$$

The rotation is stored and optimized as a quaternion q and the scale as a three-dimensional vector s. Both forms of representation can be easily converted back to their respective matrix forms, which are needed for the actual covariance matrix, and are optimized separately along the way.

In addition to these spatial criteria, Gaussians are assigned an alpha value for opacity together with a directional appearance component, also called color, represented by spherical harmonics (SH), as is standard practice by Yu et al. [YFT+21] and Müller et al. [MESK22].

Differentiable Gaussian Rasterization. To meaningfully use Gaussian Splats, there needs to be a way to project them from their three-dimensional space into image space for rendering. Kerbl et al. [KKLD23] make use of the approach by Zwicker et al. [ZPBG01], where W is the viewing transformation and Σ' is the covariance matrix in camera coordinates, which we get as follows:

$$\Sigma' = JW\,\Sigma\,W^T J^T \tag{2.3}$$

Here J denotes the Jacobian of the affine approximation of the projective transformation matrix.

Projecting Gaussians enables Kerbl et al. [KKLD23] to introduce a concept central to the 3DGS approach: their fast, differentiable rasterizer with approximate alpha-blending. Previous alpha-blending solutions such as Kopanas et al. [KPLD21] and Kopanas et al. [KLR+22] sorted their primitives for pixel-wise alpha-blending, incurring considerable overhead. 3DGS, in contrast, splits the screen into 16×16 tiles and assigns each Gaussian a key consisting of the tile ID it belongs to and its depth. Gaussians falling into two tiles are duplicated in the sense that they get two keys assigned. After some standard procedures, such as clipping the view frustum, the Gaussians are sorted in a single global

GPU-based Radix sort using the keys we described. This ensures Gaussians are grouped in their respective tiles, sorted from front to back. Without any additional pixel-wise sorting, Kerbl et al. [KKLD23] directly resume with their α-blending, where, for each pixel, the color and α values are accumulated until a target saturation (e.g., α approaches 1) is reached. The missing second sorting on the pixel level could be causing artifacts due to Gaussians overlapping and having different depths in multiple pixels, but Kerbl et al. [KKLD23] point out that these artifacts are negligible for the reconstruction quality. This rasterizer promises fast rendering and allows gradients to flow back into training without any major issues, contributing strongly to the success of 3DGS in the novel view synthesis community.

Initialization and Optimization. The Gaussian pipeline cannot start reconstruction from an empty scene and thus needs initial Gaussian primitives present. A naive approach would be to uniformly distribute Gaussian Splats across the scene inside a cube or any other primitive. As Kerbl et al. [KKLD23] pointed out, this leads to failure to reconstruct scenes with a background, causing the reconstruction to enter an unrecoverable state. The usual approach, as in the original 3DGS paper, is to use a sparse point cloud from SfM (typically COLMAP) and take the coordinates of the scene points as the means for the Gaussian primitives. These spawned Gaussians are all isotropic, making the initial orientation irrelevant. This still requires determining the scale and color of each Gaussian. Kerbl et al. [KKLD23] use the average distance to the three closest scene points for the initial scale and reuse the color of the COLMAP scene point for the degree-zero (DC) SH coefficient.

Following this, Kerbl et al. [KKLD23] use stochastic gradient descent techniques for their optimization stage, taking full advantage of already existing GPU-accelerated frameworks. They combine the loss function $\mathcal{L}_1$ and differentiable structural similarity index measure (D-SSIM) with an interpolation weight λ term and thus get the total loss $\mathcal{L}$:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}, \tag{2.4}$$

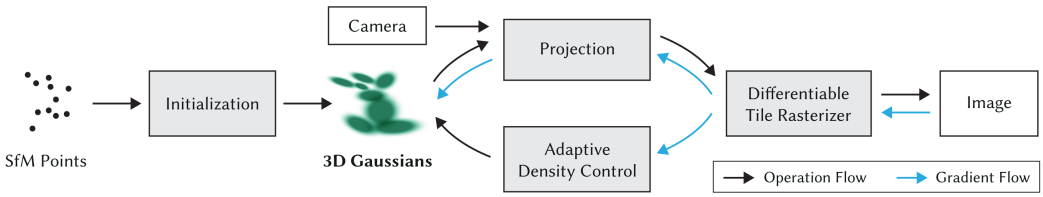


Figure 2.1: The optimizatin pipeline illustrated by Kerbl et al. [KKLD23]

Adaptive Density Control. Kerbl et al. [KKLD23] emphasize that due to the ambiguity of 3D-to-2D projection, Gaussians might be placed incorrectly in the scene. This necessitates creating, moving, or flat-out destroying misplaced Gaussians.

This allows Kerbl et al. [KKLD23] to go from a sparse set of points from SfM to a denser set of Gaussians that better represent the scene, adaptively controlling the number of Gaussians and their density per unit volume in the process. After an initial optimization warm-up, they densify every 100 iterations and prune Gaussians with an α below the ϵ-threshold, since these nearly or fully transparent Gaussians do not meaningfully contribute to the scene.

For empty regions, in other words, they are missing geometry in the form of Gaussian primitives, new Gaussians need to be created or existing ones duplicated. This erroneous state of construction is called *under-construction*. On the other hand, there are regions covered by large Gaussians, not capable of correctly representing the geometry on their own. These regions are *over-constructed* and the giant Gaussians are subsequently split to represent the scene better. Figure 2.2 illustrates this process for both cases. Since both cases, *under-* and *over-constructed* regions, are good candidates for densification, Kerbl et al. [KKLD23] sets a threshold τ_{pos} for the view-space position gradient that determines whether densification shall occur. Intuitively, a large view-space positional gradient means that a rendered image would benefit greatly from changing the position of the projected Gaussian that produced it, indicating that the region influenced by this Gaussian has not yet been sufficiently reconstructed.

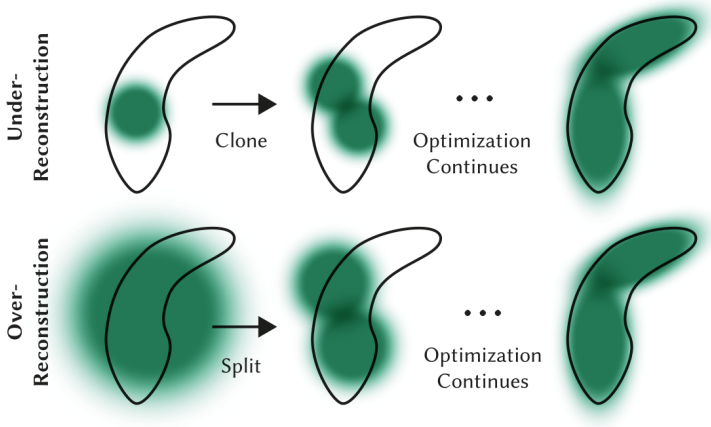


Figure 2.2: A visual example of a clone and a split operation; source: [KKLD23]

Relightable 3D Gaussians. Since the original 3DGS approach by Kerbl et al. [KKLD23] is non-relightable, as the illumination is baked into its learning process, R3DGS [GGL+24] introduces a solution based on 3DGS for fully relightable scenes, which is the codebase for our thesis. They achieve this by introducing multiple new learnable parameters, such as normals, albedo, roughness, indirect incident lighting, and global environment lighting. This is complemented by point-based ray tracing to calculate occlusion and visibility.

These parameters are not optimized simultaneously, but in multiple stages. The first mainly reconstructs the geometric representation of the scene with all conventional 3DGS parameters and the additional normal per Gaussian. Following this stage, a BVH tree is built, and visibility is precomputed. In the final step, they freeze the geometric information and only optimize PBR-dependent parameters, such as roughness and albedo. For our thesis, only the first stage is relevant; we do not perform the subsequent stages because we aim to improve the convergence of Gaussians in a general Gaussian reconstruction pipeline.

For actually rendering images, R3DGS uses physically based rendering to calculate a color $c'(\omega_o)$ for the outgoing direction ω_o for each Gaussian as shown in Equation 2.5, which is then used in the α-blending of the original 3DGS pipeline:

$$c'(\omega_o) = \sum_{i=1}^{N_s} \underbrace{f(\omega_o, \omega_i)}_{\text{BRDF}} \underbrace{L_i(\omega_i)}_{\text{lighting}} \underbrace{(\omega_i \cdot n)}_{\text{surface orientation}} \Delta\omega_i \tag{2.5}$$

Here, N_s denotes the number of sampled incident light directions; ω_i denotes the incident light direction; $L_i(\omega_i)$ denotes the radiance from the incident light direction ω_i; n is the surface normal; f is the BRDF; and $\Delta\omega_i$ is the solid-angle weight.

2.2 Monocular Depth Estimation

Monocular Depth Estimation aims to infer depth from a single image alone, rather than from geometric properties. It uses visual clues learned from training data to do so.
DAV2 by Yang et al. [YKH+24] is one of these monocular depth estimators. It builds upon the DINOv2 encoder and DPT decoder to produce a relative inverse depth map. An example of such inferred depth is shown in Figure 2.3.

The base model of DAV2 produces an affine-invariant inverse depth map, where we do not have absolute depth values, only relative estimates, that we have to adjust by an offset and scale to bring them into one consistent scene space. The authors of DAV2 provide an additional model that can produce metric depth estimates, thereby rendering the adjustment stage redundant. We still preferred the relative estimates to use the already available sparse COLMAP point clouds that typically accompany scenes for reconstruction to estimate Gaussian depths from the sparse depth values of the COLMAP scene points.

2.3 Structure from Motion

Motivation. To increase the accuracy of our depth estimates, used to unproject sampled Gaussians in image space into our three-dimensional scene, we use prior known points placed in the target reconstruction scene. To obtain these points, we use COLMAP



(a) Synthetic input image (b) Estimated depth

Figure 2.3: An example of monocular depth estimation from a single input image. Adapted from [YKH+24]

[SF16], a SfM pipeline that reconstructs the three-dimensional structure of a scene from multiple overlapping image sets and the camera parameters.

SfM pipeline. COLMAP starts by detecting distinct features in the input images. These features are accompanied by so-called descriptors, which visually describe their corresponding features. This allows COLMAP to match the corresponding features based on the similarity of their descriptors between multiple input images. Since some of the correspondences might be erroneous, Schönberger and Frahm [SF16] use geometric verification, where, for each image pair obtained from the previous matching stage, COLMAP attempts to find a valid transformation that maps the matched key features from one image to the other using projective geometry. This set of matched and subsequently mapped key features is prone to outliers due to its initial reliance solely on appearance. Therefore, a robust selection process is needed; in COLMAP's case, it is RANSAC [FB81]. This process creates a set of geometrically verified image pairs C with their corresponding inliers M_{ab}, where a and b denote image pairs, and outliers are excluded from further processing in the pipeline.
Using these remaining correspondences, the relative positions and orientations of the camera(s) that captured the input images can be estimated. Once a scene point has been observed from multiple viewpoints, its three-dimensional position in the scene can be recovered via triangulation. A feature observed in multiple images therefore forms a so-called track that connects the reconstructed three-dimensional point to its corresponding two-dimensional coordinates on each image along the track.
To avoid the SfM pipeline of COLMAP from drifting into an unrecoverable state, the estimated camera parameters and reconstructed three-dimensional scene points are subsequently refined by bundle adjustment (BA) [TMHF00]. BA aims to minimize the reprojection error E by the joint non-linear refinement of camera parameters P_c and point parameters X_k:

$$E = \sum_j \rho_j(||\pi(P_c, X_k) - x_j||_2^2) \tag{2.6}$$

where ρ is a loss function to potentially down-weight outliers and π is a function transforming scene points into image space. x_j is the coordinate of a local 2D feature corresponding to the observation j.

Output. The relevant result of COLMAP for this thesis is a sparse point cloud used to adjust relative monocular depth estimations to a metric depth map we use for placing sampled Gaussian primitives into our three-dimensional scene. Furthermore, we assume that the image-space coordinates of the feature observations used to triangulate scene points indicate how well certain locations are supported by SfM geometry for depth estimation. We reason that the closer a key feature is to a sampled Gaussian splat in image space, the more likely it is that the corresponding scene point can provide an accurate depth correction for that Gaussian primitive.

Not all points reconstructed are suitable for depth adjustments. We usually take only COLMAP-reconstructed points with a reprojection error of about 0.4-0.6 pixels (see parameter Table B.1).

2.4 On-the-fly Reconstruction for Large-Scale Novel View Synthesis from Unposed Images

Even though 3DGS is already a fast solution among radiance field methods with high visual quality, a full reconstruction of a scene, typically consisting of upfront camera pose estimation followed by subsequent optimization with 3DGS, can take from multiple minutes to multiple hours for larger scenes. Meuleman et al. [MSL+25] presents their own on-the-fly radiance field reconstruction, intended to run on consumer hardware and to reconstruct the scene by the time the capture has finished.

Lightweight Initial Pose Estimation / Joint Pose and Gaussian Optimization and Scheduling. Since each view image is initially unposed, Meuleman et al. [MSL+25] starts by approximating their poses using deep feature correspondences. These estimates provide the initialization for the subsequent joint optimization of camera poses, which is performed alongside scene reconstruction. This avoids the overhead of running a complete offline SfM pipeline like COLMAP before reconstruction.

Sampling Gaussian Primitives. To mitigate the cost and shortcomings of the densification process, Meuleman et al. [MSL+25] introduce their own sampling approach to generate a dense set of Gaussian primitives per frame that aims to cover previously unseen regions or complement under-reconstructed regions. The sampling method aims to place Gaussians in regions that most likely need a high number of Gaussians. The authors

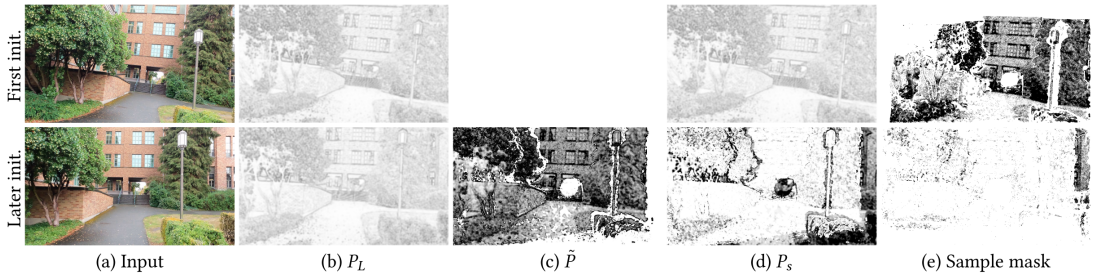


Figure 2.4: Probability-based sampling of Gaussian primitives using the input-image response and the penalty map of the current reconstruction; source: [MSL+25].

propose the sides of discontinuities in high-frequency regions as preferred locations for placing Gaussian primitives.

For that purpose, they introduce a probability map P_L that is based on the LoG response:

$$P_L(x, y) = min(||\nabla^2(n_\sigma) * I(x, y)||, 1) \tag{2.7}$$

Here, ∇^2 is the Laplacian Operator, n_σ is a Gaussian kernel with standard deviation σ, and I is the input image.

To avoid the placement of excessive numbers of Gaussians, meaning the placement of more Gaussian primitives in a given region, which is necessary to represent a named region, Meuleman et al. [MSL+25] introduces a penalty map $\tilde{P}$, which is the LoG response of the rendered image from the current view $\tilde{I}$, taking all previously spawned Gaussians into account:

$$\tilde{P}(x, y) = min(||\nabla^2(n_\sigma * \tilde{I}(x, y)||, 1) \tag{2.8}$$

Combining these two maps yields the final probability map P_s for spawning Gaussian primitives in the current view:

$$P_s(x, y) = max(P_L(x, y) - \tilde{P}(x, y), 0) \tag{2.9}$$

Figure 2.4 illustrates this approach.

This only yields a distribution of Gaussian means over a 2D picture. To meaningfully place them in the scene, Meuleman et al. [MSL+25] used the monocular depth estimator DAV2 [YKH+24] to estimate a depth and unproject the Gaussian primitives into the 3d scene. They align the depth based on triangulated matches, using MADM described by Kerbl et al. [KMK+24] in equation 3.5. Since monocular depth can introduce noticeable errors,

Meuleman et al. [MSL+25] applies guided matching using a standard correlation-volume approach centered on monocular depth.

Finally, it is necessary to determine the size parameters for every newly spawned Gaussian primitive. A conventional approach is to base the isotropic scale of each Gaussian on the mean distance to its neighbors, as in Kerbl et al. [KKLD23]. However, according to Meuleman et al. [MSL+25], this tends to lead to overly large Gaussians and is sensitive to outliers. Meuleman et al. [MSL+25] instead propose an appropriate scaling factor based on the probability distribution P_L before applying the penalty term. Assuming that the sampled pixels follow a two-dimensional Poisson process, they derive the scale s' for a pixel (x, y) based on the formulation of Clark and Evans [CE54]:

$$s' = \frac{1}{2\sqrt{P_L(x, y)}} \tag{2.10}$$

Utilizing the cameras focal length f and the estimated depth z Meuleman et al. [MSL+25] calculate the final scale s by:

$$s = \frac{zs'}{f} \tag{2.11}$$

This scale s is applied to each axis of its respective Gaussian primitive.

Scalable Incremental Gaussian Construction To achieve fast reconstruction of large-scale scenes, Meuleman et al. [MSL+25] aims to avoid the overhead of building, optimizing, and maintaining hierarchical data structures for managing large numbers of Gaussian primitives. Instead, they introduce their Anchor approach, in which they maintain an active set of Gaussians that are currently in use. Gaussians deemed not to be useful, i.e., Gaussians whose projected size is too small to contribute to rendering, are offloaded into so-called anchors, which themselves are offloaded onto the CPU. If necessary, these anchors are loaded back into GPU RAM so they can be rendered and contribute to training or for visualization.

CHAPTER 3

Methods

3.1 System overview

The reconstruction pipeline is divided into two parts: the initial Gaussian spawning and the optimization stage. The latter part corresponds to the optimization stage in Gao et al. [GGL+24], whereas the former implements our initial Gaussian spawning scheme based on the direct sampling method in Meuleman et al. [MSL+25].

For the initial Gaussian spawning, we require a prior reconstructed scene from COLMAP with a sparse point cloud and key features associated with the training images provided to the reconstruction pipeline.

The overall reconstruction pipeline is illustrated in Figure 3.1, while Algorithm 3.1 details the initial Gaussian spawning procedure.

3.2 Placing of Gaussian Primitives

It is common practice to sample initial Gaussian primitives either by uniformly distributing them across the scene or by using an existing prior point cloud, which, for example, was obtained with COLMAP.

Meuleman et al. [MSL+25] introduces a new approach in which Gaussian primitives are spawned per training view, with a heuristic that favors high-frequency over low-frequency regions, and a monocular depth estimate, aided by a depth adjustment, that places these Gaussians into our 3D scene. The intuition is that low-frequency regions of our reconstructed objects can be covered by a few big and flat Gaussians, while high-frequency regions tend to need more Gaussians in number. Following this idea, Meuleman et al. [MSL+25] propose using a probability map over input images, where locations near discontinuities in high-frequency regions have high probability. They utilize the LoG to

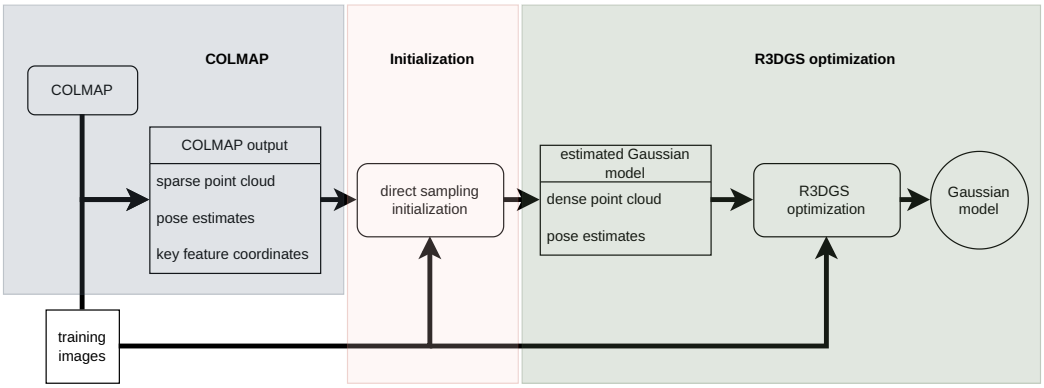


Figure 3.1

Algorithm 3.1: Direct Sampling Initialization

Input: training views $\mathcal{V}$, COLMAP reconstruction R, feature threshold τ_F, neighborhood criterion α

Output: initial Gaussian set $\mathcal{G}$

$\mathcal{G} \leftarrow \emptyset$;
foreach *view* $V_i \in \mathcal{V}$ **do**
  normed view vector $v_i \in V_i$;
  $N_i \leftarrow \{v_k \in \{V_k \in \mathcal{V}\} \mid k \neq i \wedge \langle v_i, v_k \rangle > \cos(\alpha)\}$;
end
foreach *view* $V_i \in \mathcal{V}$ **do**
  $K_i \leftarrow$ `VisibleKeyFeatures`$(\mathcal{V}, R)$;
  if $|K_i| \geq \tau_F$ **then**
    $D_i \leftarrow$ `EstimateDepth`(V_i, R) ;
    $\tilde{I}_i \leftarrow$ `RenderPenalty`$(V_i, N_i, \mathcal{G})$;
    $(P_{L,i}, S_i) \leftarrow$ `SampleMask`$(V_i, \tilde{I}_i)$;
    $\mathcal{G}_i \leftarrow$ `GenerateGaussians`$(V_i, D_i, P_{L,i}, S_i)$;
    $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_i$;
  end
end
return $\mathcal{G}$;

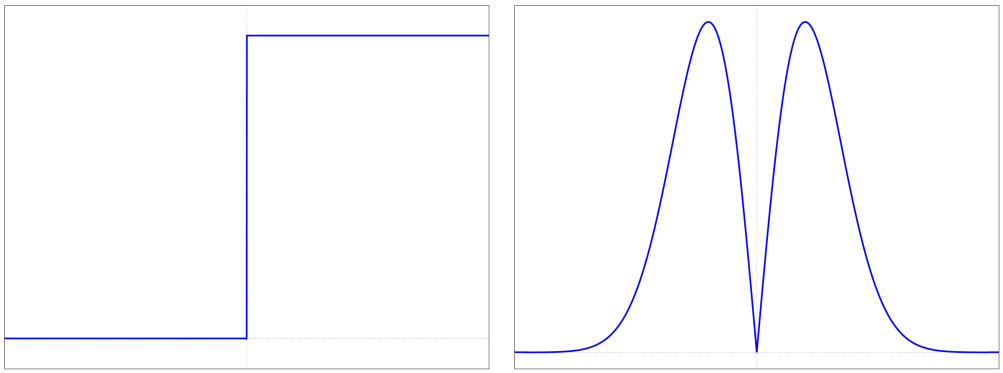
create a corresponding function, of which we use a slightly altered version without any clipping:

$$P_L(x, y) = ||\nabla^2(n_\sigma) * I(x, y)|| \tag{3.1}$$

Here, $I(x, y)$ is the input pixel at the image coordinates (x, y), n is a Gaussian kernel

with standard deviation σ, and ∇^2 is the Laplacian operator. Taking the absolute value of this convolution yields the initial probability map $P_L(x, y)$.

This results in a step function that not only yields our favored high values next to discontinuities, but also produces low values directly along named discontinuities. Figure 3.2 shows this conceptually, using an ideal step function as input, and Figure 3.3 demonstrates this approach for a clear, simple discontinuity.



(a) Ideal one-dimensional step edge.

(b) Absolute LoG response to the ideal step edge, recreated from Meuleman et al. [MSL+25].

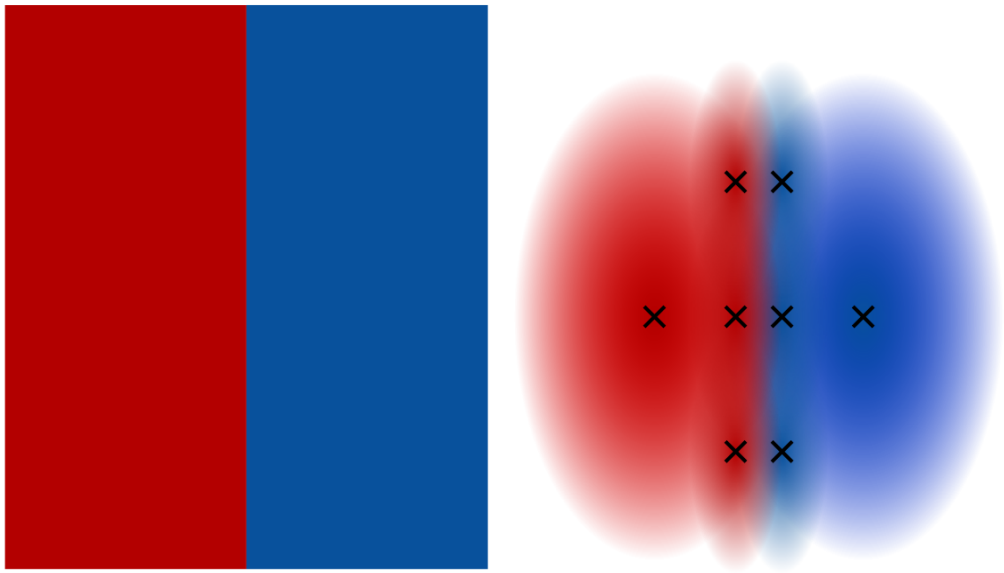
Figure 3.2: Conceptual illustration of the LoG response to an ideal step edge. The response produces high values on both sides of the discontinuity, while the value at the discontinuity itself is low.

An issue arises when the same high-frequency region is captured by multiple camera angles, which is a near-given in training images used for Gaussian reconstruction. Since new Gaussians are spawned for each training image that captures the same region, these regions suffer from an excessive number of Gaussians that are not fully needed to represent them correctly. Meuleman et al. [MSL+25] utilizes a penalty map $\tilde{P}$ to combat this issue, where we use the rendered image $\tilde{I}$ that we obtained by rendering from the viewpoint of the current training image that we want to use to initialize new Gaussians according to the LoG response 3.1:

$$\tilde{P}(x, y) = ||\nabla^2(n_\sigma) * \tilde{I}(x, y)|| \tag{3.2}$$

Intuitively, this penalty map indicates how well the scene is already reconstructed from the current view, as measured by the LoG. Since training images and rendered images of the reconstructed scene from the same view converge, P_L and $\tilde{P}$ must converge too. Their difference shows the remaining error.

In contrast to the original paper of Meuleman et al. [MSL+25], we cannot rely on a sequentially ordered image set. This leads to issues where Gaussians contribute to a



(a) A discontinuity in an image. (b) Placement of the Gaussians.

Figure 3.3: A showcase of how Gaussians could be placed according to the LoG, reproduced from Meuleman et al. [MSL+25]. Each cross represents the mean of a Gaussian splat.

rendered image of a penalty map that should be covered by newly to-be-initialized Gaussians. Even with a volumetric rendering approach, such as the fast rasterizer in Kerbl et al. [KKLD23], elements can be covered by other Gaussians when their opacity is low. Thereby, we introduce our own approach to this problem by selectively choosing which Gaussians to render for our penalty maps.

For that, we assign each newly spawned Gaussian batch, i.e., all Gaussians sampled from one training view, to the respective view that initialized it and create a neighborhood map, where each view i has a list of neighbors defined by a criterion using their view vector v:

$$\cos(\alpha) < \langle v_{\text{penalty}}, v_{\text{current}} \rangle \tag{3.3}$$

The map is created once, before starting the initial Gaussian spawning, and empty views without any associated Gaussian splats are simply ignored.

Since the norm of the LoG function peaks next to visual discontinuities, regions with low frequencies could suffer from missing Gaussian representation. We introduced a base probability p (see parameter Table B.1) to combat this issue.

As we will see in the later section on depth estimation 3.3, we use spatially close scene points from COLMAP to refine our depth estimates. Since we assume that the quality of the depth adjustment declines the farther away a Gaussian is placed from a prior known scene point, we introduce a coverage map that punishes the sampling of primitives far

away from mentioned points. To create the coverage map, we produce a two-dimensional raster of the same size as the input image and, by extension, the depth map. We initialize the raster to 0.0 everywhere, and for each two-dimensional coordinate of a COLMAP feature point, we set it to 1.0. Finally, we apply a Gaussian blur to this map with a standard deviation σ_C (see parameter Table B.1) to obtain a coverage map P_C and multiply it by our LoG P_L. Together with our penalty $\tilde{P}$ and base probability p, we determine the probability to sample a Gaussian primitive at the given pixel (x, y) on our input image by:

$$P = \max(P_L(x, y), p) * P_C(x, y) - \tilde{P}(x, y) \tag{3.4}$$

The construction of the final probability map is summarized visually in Figure 3.4 and algorithmically in Algorithm 3.2.

Algorithm 3.2: Sampling mask construction

Input: training views $\mathcal{V}$, COLMAP reconstruction R, key Input image $I \in \mathbb{R}^{H \times W}$, rendered penalty image $\tilde{I}_i \in \mathbb{R}^{H \times W}$

base probability p and image mask M_i, uniform distribution $\mathcal{U}(0, 1)$

Output: probability map P_L, sampling mask S_i

$K_i \leftarrow$ `VisibleKeyFeatures(`$\mathcal{V}, R$`)`;

$P_L \leftarrow$ `LoG(`I_i`)`;

$P_L \leftarrow \max(P_L, p)$;

$P_C \leftarrow$ `CoverageMap(`K_i`)`;

$P_F \leftarrow P_L \odot P_C$;

$\tilde{P} \leftarrow$ `LoG(`$\tilde{I}_i$`)`;

$P \leftarrow P_F - \tilde{P}$;

$U \leftarrow \mathcal{U}(0, 1)^{H \times W}$;

$S_i \leftarrow (U < P) \wedge M_i$;

return P_L, S_i;

3.3 Depth estimation

Since our process in Chapter 3.2 only produces Gaussians distributed over a two-dimensional image, we need to estimate their respective depths to place them in our three-dimensional scene. Just as in the original paper by Meuleman et al. [MSL+25], we use DAV2 [YKH+24]. This yields a relative inverse depth map, which we must restore to its correct orientation and adjust so that we obtain metric depth values consistent across all depth estimates, regardless of view. As a first step, we apply the method used by Kerbl et al. [KMK+24], which we call MADM, to obtain an adjusted depth map D^*:



(a) Input



(b) probability map P_L



(c) rendered image

(d) penalty map $\tilde{P}$



(e) coverage map P_C

(f) final probablity map

Figure 3.4: Presentation of our pipeline for creating the probability map for the Gaussian sampling. (a) is the current image we base our sampling on. (b) is the raw LoG of our current input image, darker colors representing higher values. (c) is a rendered image of our scene that we have reconstructed so far using only Gaussian primitives; we use this to (d) create a penalty map that penalizes regions already covered by Gaussians. Before we subtract the penalty map from the probability map, we apply the coverage map (e), which reduces the probability of a Gaussian primitive being spawned in regions that are not well covered by prior known scene points reconstructed by COLMAP, thereby adjusting the depth estimates.

$$
\begin{aligned}
t(D) &= \text{median}(D) \\
s(D) &= \frac{1}{M} \sum_{i \in \text{SfM}} |D_i - t(D)| \\
D^* &= \frac{s(D_{\text{SfM}})}{s(D)} D + t(D_{\text{SfM}}) - t(D) \frac{s(D_{\text{SfM}})}{s(D)}
\end{aligned}
\tag{3.5}
$$

Here, D is the depth map produced by DAV2, as shown in Figure 3.5a, M is the number of available COLMAP scene points for the particular view that produced the initial depth map, $i \in \text{SfM}$ are coordinates of COLMAP's scene points projected into image space, thus D_i is a depth value at the coordinate associated with i, and D_{SfM} is the sparse set of COLMAP depth values available for this particular view.

This approach is a scale-and-shift operation of the form $D^* = aD + b$, still operating within an inverted orientation, but already defining a metric space. To restore the depth map to its correct orientation, we invert D^*. The inversion of orientation is evident in the transition from Figure 3.5a to Figure 3.5b, where the colors are inverted. However, this still led to substantial depth misalignments. Therefore, we retain this transformation step and add an additional depth alignment. First, we create a difference map between the DAV2 result, which was already aligned once by the method mentioned above, and the depth of the available COLMAP key features. Then, for each depth value D_{final}, we choose the k nearest neighbors (see parameter Table B.1) and weigh them by the weight w defined as:

$$
w = \frac{1}{d^\gamma + \epsilon_w} \tag{3.6}
$$

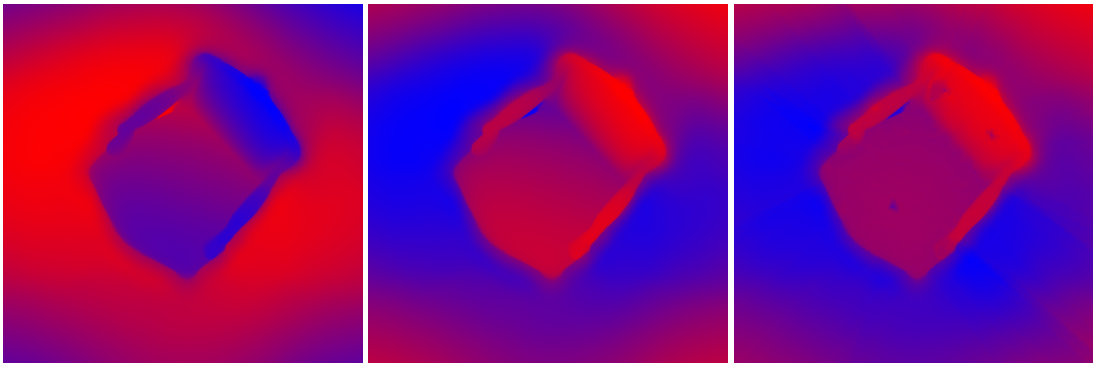
where d is the Euclidean distance between the depth pixel in question and the COLMAP key feature that contributed to the difference map. We use the constant γ as a distance decay exponent to control how strongly close points are favored over distant ones. ϵ_w is added to the denominator for numerical stability. The final correction Δ per pixel q is an interpolation of the k nearest neighbors using the raw difference $\delta_{i,q}$ and again an epsilon ϵ_q as a numerical stabilator (see parameter Table B.1):

$$
\Delta_q = \frac{\Sigma_{i=1}^{k} w_{i,q} \delta_{i,q}}{\Sigma_{i=1}^{k} w_{i,q} + \epsilon_q} \tag{3.7}
$$

The naive approach would be to perform a nearest-neighbor search for each pixel. To limit the computational cost of querying every pixel of the depth map, we spatially downsample the search space and upsample it again for the correction map with a stride s. (see parameter Table B.1) We observed that down- and upsampling did not meaningfully reduce reconstruction quality and made some scenes with high-resolution input images more manageable. The final product of this process is shown in Figure 3.5c.

Note the small blue dots on our object. These are caused by outliers in our sparse point cloud produced by COLMAP, with high depths that are inconsistent with the scene. These outliers do not seem to substantially influence optimization or the initial Gaussian spawning with our direct sampling initialization approach.

Depending on available VRAM, we either utilized the GPU by performing the k-nearest-neighbor search with the CUDA library *pytorch3d.ops* or the CPU-based library *scipy.spatial.KDTree*. The former is used for the NeRF Synthetic and DTU datasets, and the latter is used for the TnT dataset due to its high GPU memory consumption.



(a) DAV2 monocular inverse relative depth estimate. [YKH+24]

(b) Depth adjustment by MADM with subsequent inversion. [MSL+25]

(c) Depth adjustment by k nearest neighbor search. [KMK+24]

Figure 3.5: Depth estimation pipeline, beginning with the DAV2 estimate, then being adjusted by the MADM, which itself is subsequently adjusted with a k-nearest-neighbor search. Red regions represent a low depth and blue regions represent high depth.

Since our depth estimation for a given Gaussian now mainly depends on the availability of nearby COLMAP points, we introduce a coverage map that discourages the spawning of Gaussians in locations that are not well covered by existing COLMAP points. This is done by taking the two-dimensional feature map produced by COLMAP and blurring it with a Gaussian kernel. This blurred image is multiplied by the probability map 3.1, producing Gaussian splats before the penalty map is applied.

3.4 Covariance estimation

The covariance comprises the scale and orientation of the Gaussian splat in question. To simplify the process of spawning Gaussians, we make them isotropic from the start and let their orientation be handled only later, during optimization. Thus, we only need to assign each initial Gaussian an appropriate scale.

A traditional way to set the scale of initially spawned Gaussian splats is to average the distances to the three nearest neighbors, as introduced by Kerbl et al. [KKLD23]. However, Meuleman et al. [MSL+25] pointed out that this approach is prone to producing

overly large Gaussian splats, and instead proposed an alternative that uses the existing probability map P_L obtained from 3.1. They assume that the sampled pixel locations follow a two-dimensional Poisson point process, where $P_L(u, v)$ represents the local intensity around each pixel (u, v). According to Clark and Evans [CE54], we can calculate a scale s' that is in pixel space:

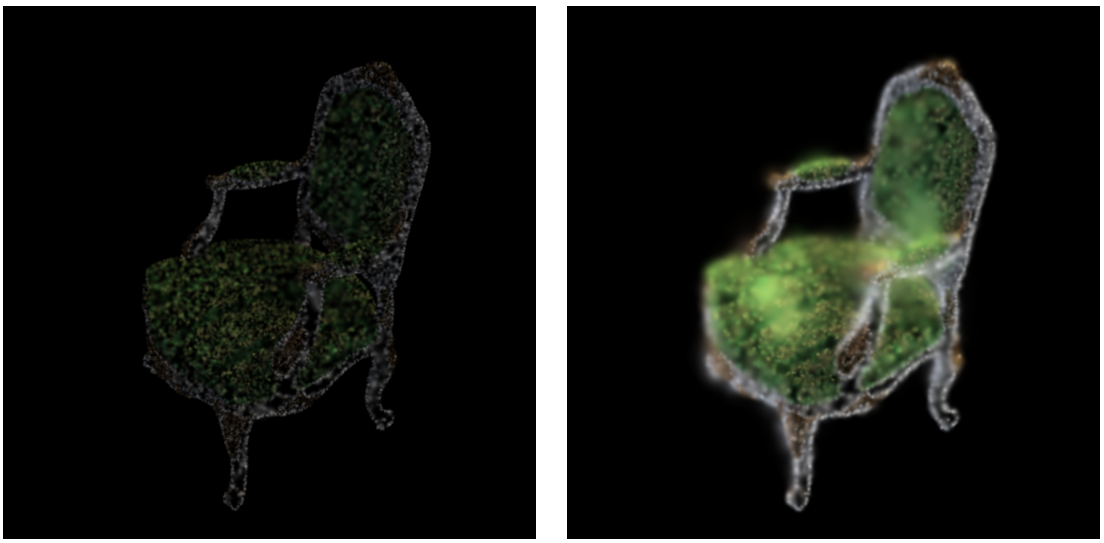
$$s' = \frac{1}{2\sqrt{P_L(u, v)}} \tag{3.8}$$

Following Meuleman et al. [MSL+25], we obtain a meaningful scaling that is bound to our three-dimensional space; we are utilizing the focal length f of the respective camera used for the view in question and our estimated depth z, which we have estimated and adjusted in the previous stage:

$$s = \frac{zs'}{f} \tag{3.9}$$

Just as Meuleman et al. [MSL+25], we apply our final scaling s to all three axes of the newly spawned Gaussian at the pixel in the current view at each position (u, v).

Figure 3.6 visually compares the scale-initialization strategy of Kerbl et al. [KKLD23] with the aforementioned method of Meuleman et al. [MSL+25], which is based on Clark and Evans [CE54].



(a) Initialization using the Clark and Evans [CE54] based scale estimate from Meuleman et al. [MSL+25].

(b) Initialization using the average distance to the three nearest neighboring Gaussians following Kerbl et al. [KKLD23]

Figure 3.6: Comparison of two strategies for initializing the scale of newly spawned Gaussians from a training view. Both images are rendered from the same viewpoint and contain Gaussians spawned using the corresponding training view. The Clark-Evans-based initialization estimates the Gaussian scale from the local 2D probability map, whereas the traditional nearest-neighbor approach sets the scale according to the average distance to the three closest Gaussians.

CHAPTER 4

Evaluation

4.1 Setup

The evaluation was conducted on Ubuntu 24.04.4 LTS with an NVIDIA GeForce RTX 5090, providing up to 32 GiB of VRAM using the CUDA Toolkit 12.8. For further details on the experimental environment, consult the appendix C.

For the evaluation, we used three separate datasets: NeRF Synthetic [MST+21], selected scenes of the DTU dataset [JDV+14], and selected scenes of the TnT dataset [KPZK17]. For the DTU dataset, we reused the scenes provided by Gao et al. [GGL+24]. They already narrowed down the pool of scenes for training to 15 of the 124 available DTU scenes and additionally provided object masks.
NeRF Synthetic already contains an out-of-the-box evaluation image set, which we used for our evaluation. For the DTU dataset, we used the hard-coded approach of the R3DGS codebase by Gao et al. [GGL+24], where they removed the training images [2, 12, 17, 30, 34] from the training set and used them solely for evaluation. As for the TnT dataset, we also used the hard-coded approach of the codebase and subsequently used every eighth input image solely for evaluation, including the very first input image. Further on, we reused the same parameters for the Gaussian reconstruction as in Gao et al. [GGL+24].

We ran the optimization for 30,000 iterations per scene, the same number of iterations as in the first stage of the original R3DGS pipeline [GGL+24]. The parameters for our direct sampling initialization of the Gaussians were selected empirically and differ slightly from dataset to dataset. Our empirical methods to determine the parameters used in our tests include a one-dimensional hill-climb search with an adaptive step size for the spatial learning scale and a coordinate descent with adaptive step sizes for the error threshold, standard deviation σ_c of the coverage map, base probability p, and the neighborhood criterion α. For the baseline, we chose the R3DGS pipeline, using the same sparse COLMAP point cloud that our approach uses for depth adjustment. To reconstruct

more accurate scene points via COLMAP, we reused the given camera parameters of each dataset. A complete list of the parameters used in our evaluation is provided in the appendix B.

Furthermore, we reused the downsampling scheme from Gao et al. [GGL+24], in which images with width $w > 1600$ are downsampled by a factor of $s = \frac{1600}{w}$. For the original dimensions (w, h), we thus receive $w' = w * s$ and $h' = h * s$. Images with a width $w \leq 1600$ are used in their full resolution for training.

The last major point of our setup was the spatial learning scale. Our codebase by Gao et al. [GGL+24] uses the so-called camera extent e_{cam} as the basis for determining the final spatial learning rate. They obtain it by computing the mean center c_{mean} of all camera positions c_i in world space, then selecting the largest distance from c_{mean} to each camera and multiplying that distance by 1.1. This camera extent is multiplied by l_{init} (see parameter Table B.1), which is 0.00016 and decays to 0.0000016 during optimization:

$$l = \underbrace{1.1 \cdot \max_i \|\mathbf{c}_i - \mathbf{c}_{\text{mean}}\|_2}_{e_{cam}} \cdot l_{\text{init}}, \tag{4.1}$$

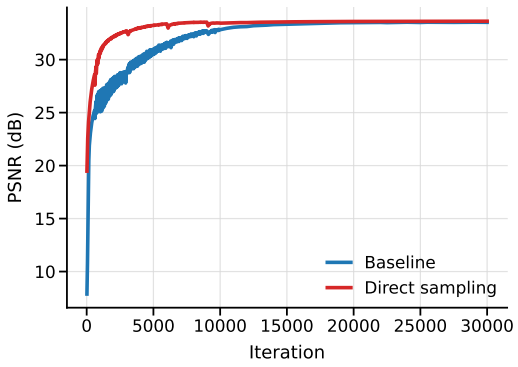
This learning scale was still too large for our optimization, so we continually scaled it down using a position learning rate scale factor, typically $0.07 - 0.09$ depending on the dataset.

4.2 Results.

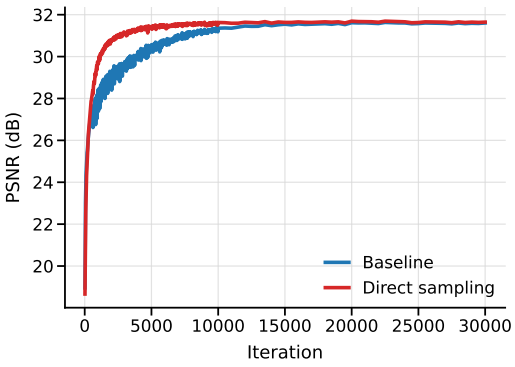
PSNR over iterations. Our direct sampling initialization approach mainly outperforms the baseline up to 10k iterations, before both the baseline and direct sampling initialization approach converge. This holds for all scenes for all three of our datasets, as shown in Figure 4.1 and summarized in Table 4.1.

Upon inspecting our approach, the PSNR appears to be more stable than the baseline at the beginning of optimization, before the PSNR of both converge and no longer oscillate. Intuitively, this indicates that our approach already has a good approximation of the model and only needs refining, compared to the baseline. This exposes our approach to the danger of falling into a local minimum and failing to escape it. This error state did not occur even once during evaluation. The variance in PSNR is also influenced by parameters such as the spatial learning rate, which we altered. Read section 4.1 for further information on the spatial learning scale.

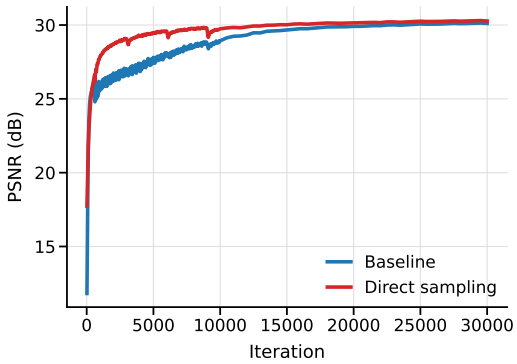
PSNR over time. In practice, one might rather prefer to measure reconstruction time instead of plain iteration count. The graph 4.2 shows PSNR over time. Note that the direct sampling initialization approach starts its reconstruction later. This is because the initial Gaussian spawning is non-trivial and adds substantial time to the reconstruction pipeline. In contrast, we dismissed the initialization of the baseline,



(a) NeRF Synthetic



(b) DTU



(c) TnT

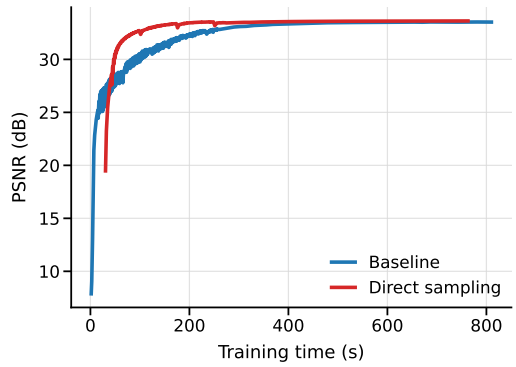
Figure 4.1: Average PSNR of evaluation images over iteration. The values are averaged over every scene of the specific dataset.

dataset	ΔPSNR @ 2k ↑	ΔPSNR @ 5k ↑	ΔPSNR @ 10k ↑	Final ΔPSNR ↑
NeRF Synthetic	+3.87	+2.24	+0.62	+0.11
DTU	+1.47	+0.87	+0.27	+0.04
TnT	+1.96	+1.68	+0.81	+0.16

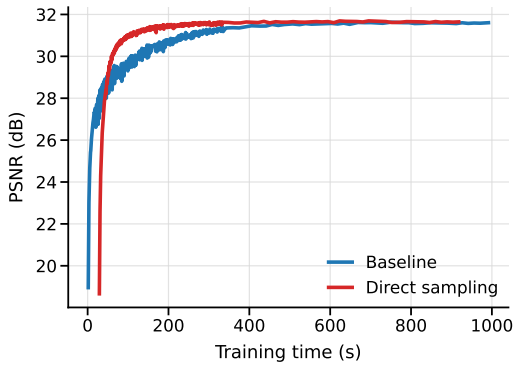
Table 4.1: The mean difference of PSNR over iterations between our direct sampling initialization approach and the baseline. The final ΔPSNR is at 30k iterations.

consisting of loading scene points and initializing Gaussians from them, as trivial. A complete runtime breakdown, including COLMAP reconstruction, initialization, training, and total runtime, is provided in Tables A.5, A.6, and A.7.

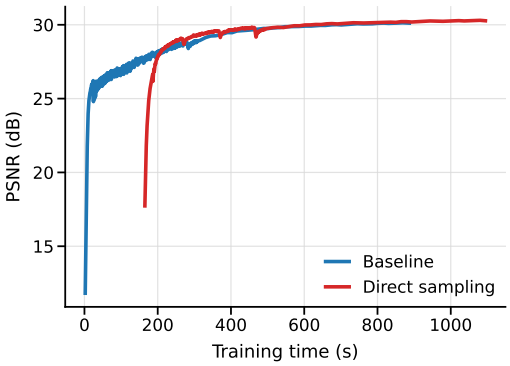
We can see for NeRF Synthetic and DTU that, even though our direct sampling initialization approach starts with a delay due to initialization, it quickly catches up and holds a meaningful lead over the baseline until its PSNR inevitably converges. The total reconstruction time with our fixed iteration count was even reduced by approximately one minute in both instances. This is in stark contrast to the TnT dataset, where the long direct sampling initialization removes any meaningful lead time for our approach before convergence and even increases the total reconstruction time. Read paragraph 5.1 in our discussion section 5.1 on the issue of the slowdown of our approach with regard to the TnT dataset.



(a) NeRF Synthetic



(b) DTU



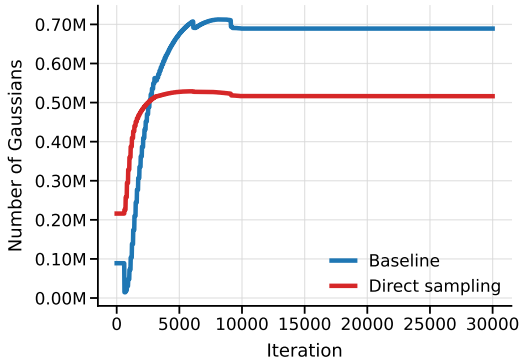
(c) TnT

Figure 4.2: Average PSNR of evaluation images over time. The values are averaged across all scenes in the specific dataset.

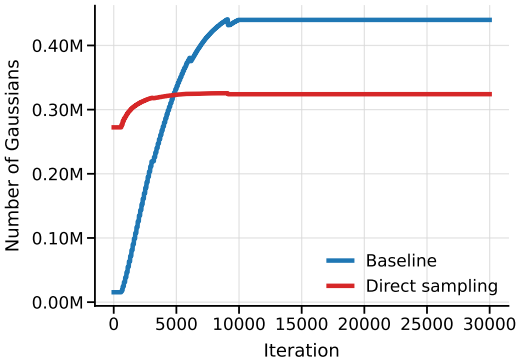
Gaussian over iterations. A secondary finding was a substantial reduction in Gaussians needed to represent certain scenes of most of our datasets. We found an average

reduction in the number of Gaussians representing a scene in the NeRF Synthetic dataset, with approximately 25%, taking the average baseline itself as the base for the relative comparison, fewer primitives than the baseline, and the same for the DTU dataset, with 26% fewer primitives at the end of reconstruction. In contrast, the TnT dataset yields a much denser reconstruction with our approach, with about 84% more Gaussians representing the scene at the end of the reconstruction than the baseline. (see table 4.2) This split of a sparse reconstruction for the former two datasets and a much denser reconstruction for the latter is also consistent for the initial density of Gaussians spawned. For NeRF Synthetic and DTU, the initial reconstruction is, on average, too sparse and is densified to satisfy the optimization. For the TnT dataset, the initial reconstruction is too dense, and Gaussians are pruned to make the scene sparser, yet the overall scene remains denser than the baseline at the end of training. Figure 4.3 shows the development of Gaussian count over the course of optimization.

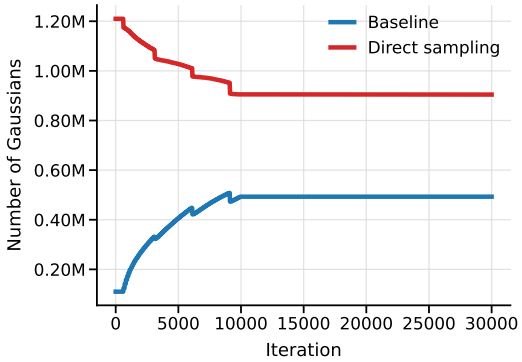
The above-mentioned behavior of the number of Gaussian primitives in a scene after optimization is consistent across all TnT scenes. In contrast, the NeRF Synthetic dataset, like the DTU dataset, exhibits large variations across scenes. For NeRF Synthetic, our approach produces fewer Gaussians in five of eight scenes, whereas the number of Gaussians at the end of optimization for the remaining three scenes does not meaningfully differ between the baseline and ours (no more than 10% difference in the number of Gaussians, taking the scene with the higher number of Gaussians as base). For the DTU set, eight out of fifteen scenes are wins for our approach, three are ties, and four are losses. Following this categorization, we split our scenes into winners, losers, and ties; we obtain an approximately 35% reduction in Gaussian primitive count for the NeRF Synthetic winner dataset with no counterpart in the loser set and an approximately 44% reduction for the DTU winner dataset, compared to a 52% increase in Gaussians for the DTU loser set. The TnT dataset consists only of losers, yielding the aforementioned 84% gain in Gaussian count. See Table 4.2 for a complete overview of the Gaussian count over certain iteration milestones and look into Appendix A.3 for individual plots depicting the development of Gaussian number size over iteration for each scene.



(a) NeRF Synthetic



(b) DTU



(c) TnT

Figure 4.3: Average number of Gaussian primitives over iterations. The values are averaged across all scenes in the specific dataset.

dataset	category	ΔN_G^{rel} [%] ↓				scenes
		2k iter.	5k iter.	10k iter.	Final iter.	
NeRF Synthetic	winner	+21.0	-29.3	-34.7	-34.7	5
DTU		+94.7	-25.6	-43.7	-43.7	8
TnT		none				0
NeRF Synthetic	tie	+54.9	-6.0	-2.6	-2.6	3
DTU		+182.3	+28.0	-1.9	-1.9	3
TnT		none				0
NeRF Synthetic	loser	none				0
DTU		+329.7	+93.4	+52.0	+52.0	4
TnT		+321.0	+154.1	+83.5	+83.5	4
NeRF Synthetic	total	+32.2	-21.8	-25.1	-25.1	8
DTU		+145.9	-2.6	-26.3	-26.3	15
TnT		+321.0	+154.1	+83.5	+83.5	4

Table 4.2: Mean relative difference in the number of Gaussian primitives grouped by the final Gaussian-count category of each scene. Winner and loser scenes lie outside the 10% tie threshold and contain fewer or more final Gaussians than the baseline, respectively. See section in appendix A.3 for tables with absolute values.

CHAPTER 5

Conclusion

This thesis integrated the direct sampling approach of Meuleman et al. [MSL+25], which is based on the LoG, into the R3DGS pipeline by Gao et al. [GGL+24] with the goal of achieving an improved convergence of PSNR over iterations of optimization and a speedup of convergence of PSNR over physical time. To integrate the direct sampling approach, we additionally had to find solutions for more precise depth estimates to map our Gaussian samples from image space to scene space. We used COLMAP scene points with varying confidence levels indicating their accuracy in representing the scene to adjust our initial depth estimates from the Monocular Depth Estimator DAV2 by Yang et al. [YKH+24]. Additional refinements, such as coverage maps, were introduced to improve the quality of the initial reconstruction.

Our approach improved the convergence of PSNR over optimization iterations across all three datasets: NeRF Synthetic, DTU, and TnT. In contrast, convergence of PSNR over time has only improved in the first two datasets, whereas the latter dataset, TnT, incurred high costs for initial Gaussian spawning, diminishing the resulting speed-up in the convergence of those initial spawns. Our direct sampling initialization approach not only improved the convergence of PSNR over time for NeRF Synthetic and DTU, but also, on average, shortened the total reconstruction time for scenes from those datasets with a fixed iteration count compared to the baseline. This led us to investigate possible explanations and led us to our secondary findings.

For selected datasets, our direct sampling initialization approach not only improves PSNR convergence but also reduces the number of Gaussians required to represent the scene. This is the case for the NeRF Synthetic dataset and the DTU datasets. In contrast, the TnT dataset exhibits the opposite phenomenon, with more Gaussian primitives representing the scene after optimization. Read the paragraph 5.1 about TnT-specific behavior for our hypothesis on this in our discussion section 5.1.

5.1 Discussion

In this section, we discuss our findings on the convergence of PSNR with respect to the computational cost of our approach, as well as our observations regarding the Gaussian primitive count.

PSNR convergence over time and computational cost. In general, we have a tradeoff. The initial direct sampling initialization Gaussian spawning incurs a nontrivial computational cost that can prolong the overall reconstruction time, depending on the scene and, more broadly, the dataset. This is the main computational bottleneck of our approach and needs special consideration. On the other hand, thanks to this initial Gaussian spawning, the PSNR improves faster over time, since optimization can already start with an early estimate of the model, and as we will see in the next paragraph 5.1, a possible reduction in the number of Gaussians leads to faster optimization, since fewer primitives need to be processed per iteration. The trade-off paid off for our first two datasets, NeRF Synthetic and DTU, resulting in faster convergence over time and a shorter total reconstruction time. In contrast, our trade-off did not pay off for TnT, resulting in no meaningful improvement in PSNR convergence over time and a longer total reconstruction time. Read the paragraph 5.1 for further discussions on TnT specific behavior.

Gaussian primitives count. A positive aspect of our secondary finding is that most scenes in NeRF and DTU end up with a more efficient representation of the reconstructed scene using fewer Gaussians than the baseline. On the other hand, this is not the case for all of them. Some of the scenes are ties, where there are differences within our 10% tie threshold between ours and the baseline, which we consider not substantial enough, and some scenes have a more dense reconstruction of the scene, without any meaningful advantage in representation, which is the case for some scenes in DTU and all scenes in TnT. We discuss the extraordinary case for TnT in the next paragraph 5.1.

TnT-specific behavior. The TnT dataset is the only set of scenes that has an overall longer reconstruction time using our direct sampling initialization approach (1), and in contrast to all the other datasets, TnT starts with a denser set of Gaussian primitives that need to be pruned over the course of optimization and end up with more Gaussian splats at the end of training compared to the baseline. (2)

For (1), we hypothesize that our direct sampling initialization method does not scale as well with the number of high-resolution input images. Our approach strongly depends on image width, height, and number, whereas the baseline only needs to load these images into memory at the start of training, along with the COLMAP scene points, which is a true subset of our process's workload. This leads to a lengthy direct sampling initialization , offsetting the advantages of improved convergence we gain later in the optimization step and rendering our approach useless from a time-saving perspective. To put the differences in the number of input images and resolutions in perspective:

Scenes of TnT have an average number of images of 274, where, according to the image scheme described in our Evaluation chapter 4, we approximately used 239 for training on average, with a resolution of 1920×1080 downsampled to 1600×900 for training. This is in contrast to NeRF Synthetic with a standard set of 96 images per scene and a corresponding resolution of 800×800, and DTU with 48 images per scene, of which 43 are used (see our Setup section for further information 4.1) with a resolution of 1600×1200. Ignoring image masks, the number of pixels that need to be processed for NeRF Synthetic is thus only about 18% of TnT and 24% for DTU.

$$\frac{96 \cdot 800 \cdot 800}{239 \cdot 1600 \cdot 900} \approx 0.179$$

$$\frac{43 \cdot 1600 \cdot 1200}{239 \cdot 1600 \cdot 900} \approx 0.240$$

Another factor contributing to the slowdown (1) of the convergence over time could be the increased number of Gaussians that need to be processed per optimization iteration compared to the baseline, as seen in Table 4.2.

This leads to our second hypothesis about the high Gaussian count generated by our approach compared to the baseline (2). We speculate that the initial Gaussian does not yet fully represent the scene, so the generated penalty maps from different views do not penalize newly created sampling masks strongly enough. Due to the large number of training images for TnT compared to its counterparts, regions are more often captured from multiple views, amplifying the shortcomings of our penalty map and generating a scene more densely populated with Gaussians as needed. We discuss a possible solution for that in paragraph 5.3 of our Future Work section 5.3.

5.2 Limitations

A limitation of our method is its dependence on the availability of COLMAP scene points, which are needed to adjust the depth estimates of DAV2 and, together with calibrated and posed images, to unproject the sampled Gaussians and subsequently optimize them. Further on, there is a strong dependence of our direct sampling initialization approach on prior COLMAP scene points in a quantitative way, since sampled Gaussians need spatially close COLMAP scene points to adjust their depth, and in qualitative ways, where misplaced COLMAP scene points lead to wrong depth corrections for sample Gaussians, causing them to miss their target.

5.3 Future Work

There are several ways to improve or extend our approach; here is a selected list of them.

Metric DepthAnythingV2. Our current solution relies on sparse COLMAP scene points to align and refine the relative DAV2 estimates, putting all depth maps into one consistent and metric coordinate system. Using metric depth estimates from DAV2 could eliminate the need for this alignment stage. This alternative approach would still require posed and calibrated input images, which are currently handled by COLMAP in our pipeline, since first we would have to bring training views and depth estimates into one consistent coordinate system, and second we would use these views for the optimization stage, which also requires posed and calibrated training images. This would leave us with the option of using a pose-estimation approach, other than COLMAP, that does not require a prior sparse reconstruction, or of revisiting the original on-the-fly approach by Meuleman et al. [MSL+25].

DepthAnythingV3. Since our work on this thesis has begun, many new tools relevant to the subject have been released. One of them is DepthAnythingV3 from Lin et al. [LCL+25], which promises more accurate monocular metric depth estimates and is especially well-suited to Gaussian Spawning. With a sufficiently short estimation time and a high-quality depth estimate, this could replace the depth adjustment step, substantially reducing the overhead associated with initialization time. This could also address the aforementioned low performance of the TnT dataset.

Anisotropic Gaussian primitives. Gaussian primitives in our approach are spawned isotropic. Having depth maps allows us to easily compute surface normals, which we could use to spawn flat Gaussians that cover large low-frequency regions. This could lead to a reduced Gaussian count, further supporting our secondary finding of an efficient Gaussian representation of the scene and perhaps improving PSNR convergence over iterations/time.

Constraint optimization. We observed that Gaussians spawned by our approach initially do not represent the scene very well, even from the view in which they were generated. This might not be an issue, considering these Gaussians will be optimized in the next stage of the pipeline, but this suboptimal initial representation has strong negative effects on our penalty map, which we assume does not identify the scene as well-constructed, even though the density of Gaussians might be sufficient to represent it. This results in a weak penalty map that does not penalize new sampling masks strongly enough, potentially leading to excessive Gaussian counts. We propose, similar to Meuleman et al. [MSL+25], to interleave the initial Gaussian spawning with the optimization stage. More precisely, we propose introducing a small number of optimization steps after each newly spawned batch of Gaussians from a view. Since these Gaussians have only been initialized from that particular view, we would optimize them using only this view while keeping their depth fixed. This should allow the newly spawned Gaussians to better approximate the observed image before they are used to render subsequent penalty maps, without introducing inconsistencies from views for which no corresponding Gaussians may have been spawned yet. We hypothesize that this would yield more accurate penalty maps

and therefore reduce unnecessary Gaussian spawning in already sufficiently represented regions.

APPENDIX A

Evaluation Results

The following plots visualize the development of the PSNR with respect to the number of optimization iterations for our initial direct sampling initialization Gaussian spawning approach and the baseline, which is the R3DGS codebase [GGL+24] using the same sparse COLMAP point cloud as we do. Evaluations are performed every 20 steps up to and including 10,000 iterations and every 500 steps thereafter.

A.1 PSNR Results

A.1.1 NeRF Synthetic

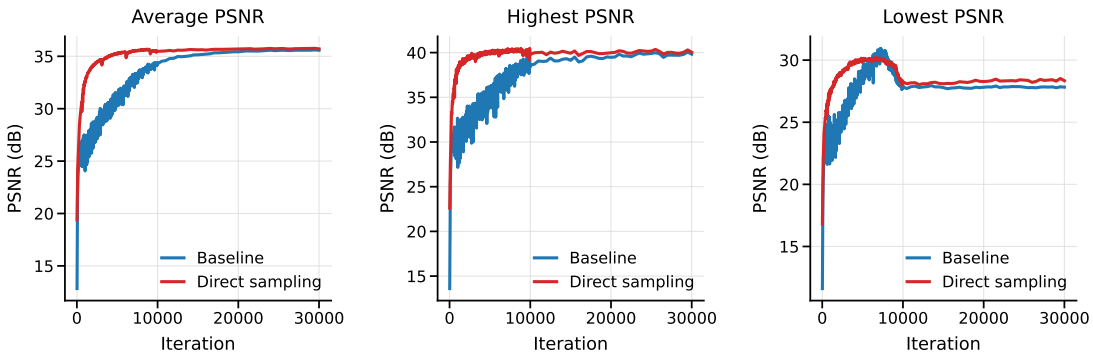


Figure A.1: PSNR curves for Chair in the NeRF Synthetic dataset.

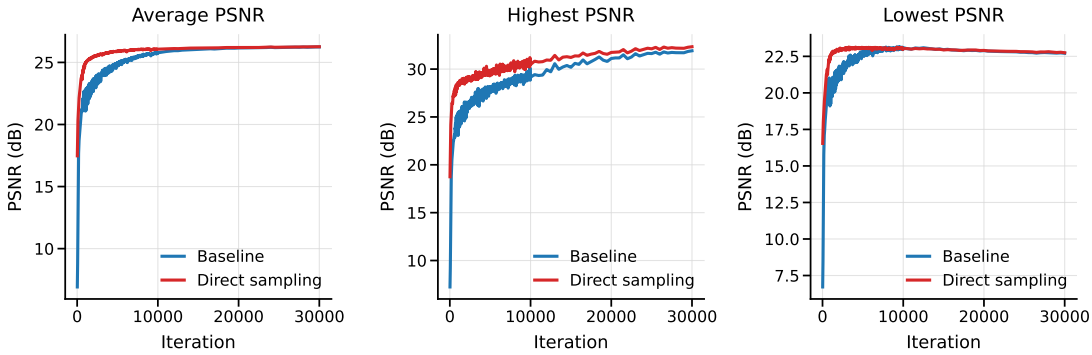


Figure A.2: PSNR curves for Drums in the NeRF Synthetic dataset.

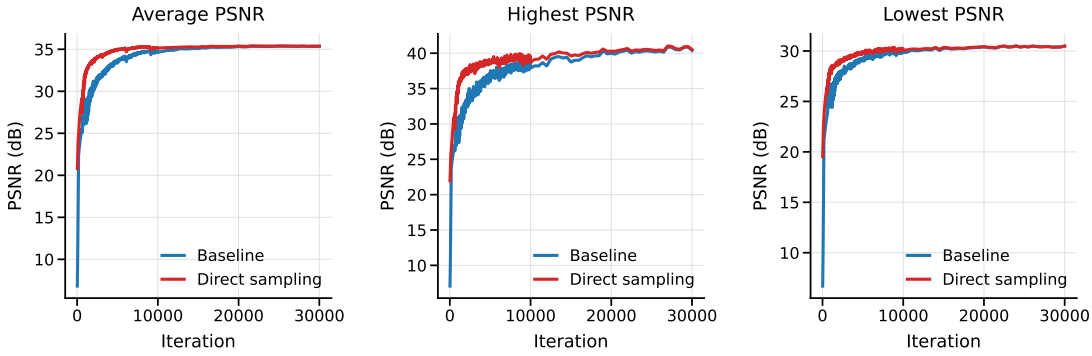


Figure A.3: PSNR curves for Ficus in the NeRF Synthetic dataset.

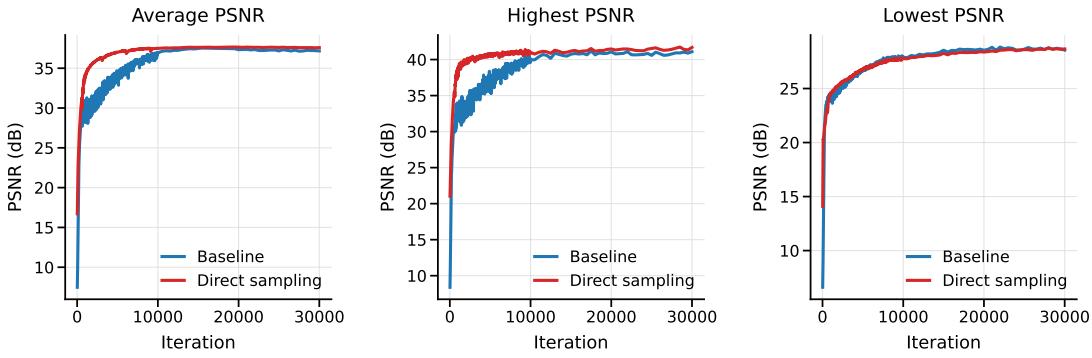


Figure A.4: PSNR curves for Hotdog in the NeRF Synthetic dataset.

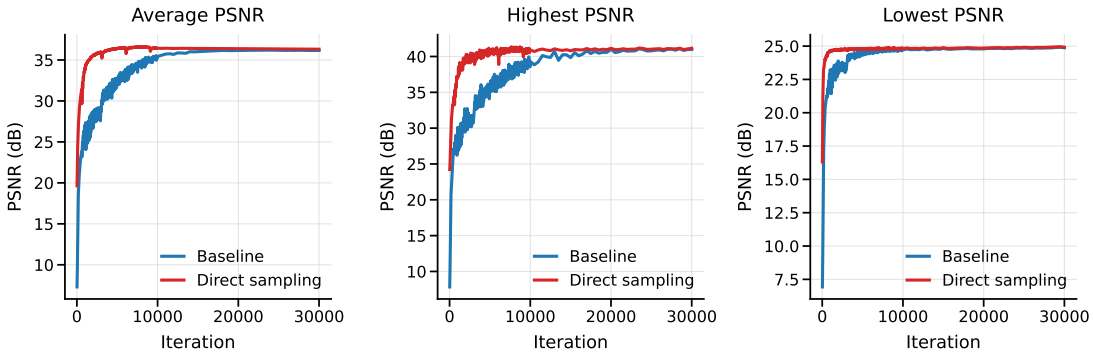


Figure A.5: PSNR curves for Lego in the NeRF Synthetic dataset.

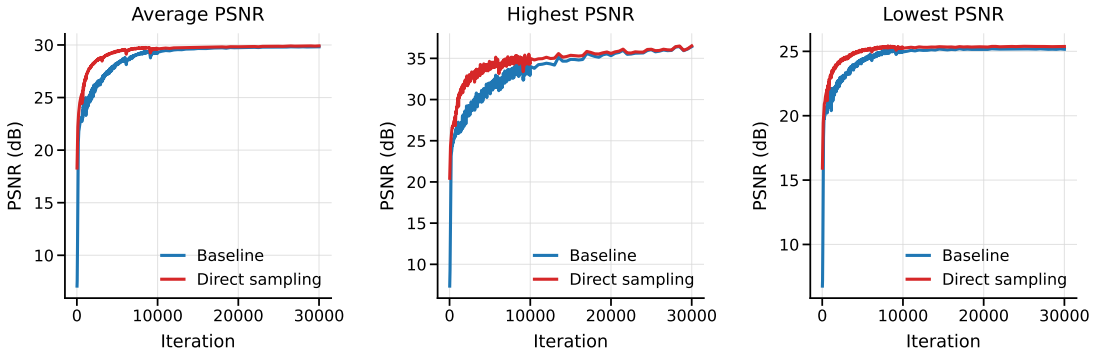


Figure A.6: PSNR curves for Materials in the NeRF Synthetic dataset.

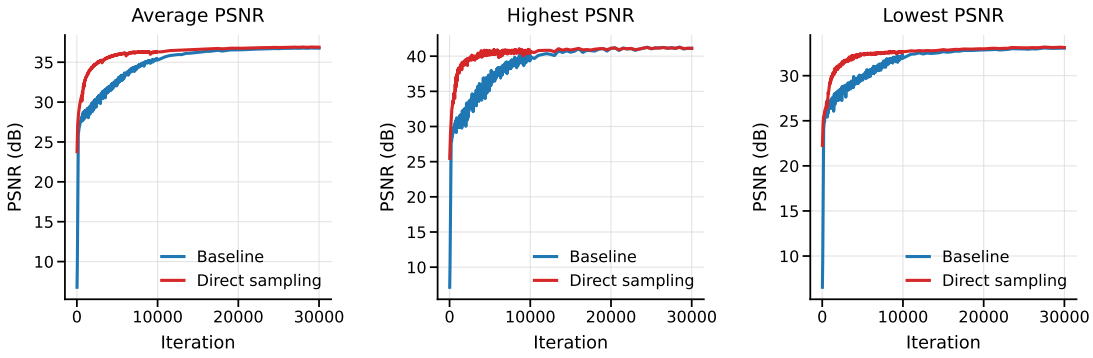


Figure A.7: PSNR curves for Mic in the NeRF Synthetic dataset.

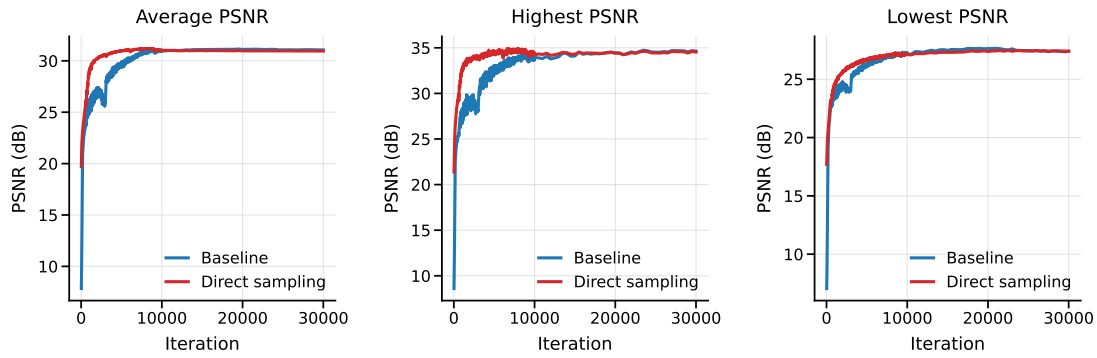


Figure A.8: PSNR curves for Ship in the NeRF Synthetic dataset.

A.1.2 DTU

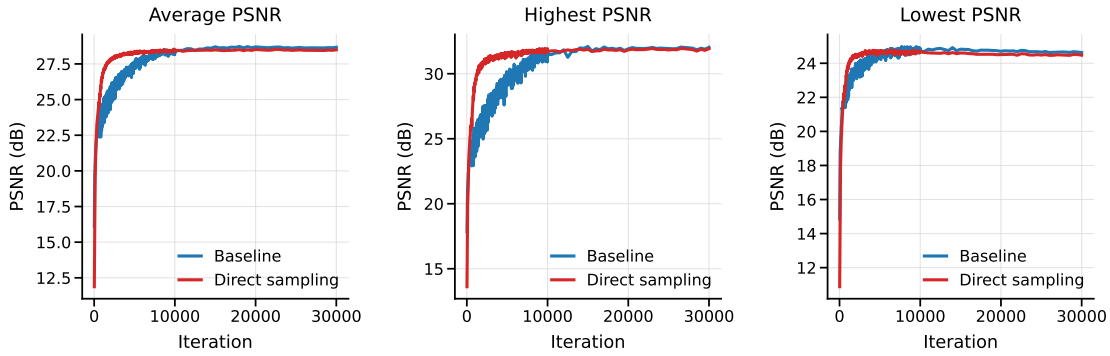


Figure A.9: PSNR curves for Scan 24 in the DTU dataset.

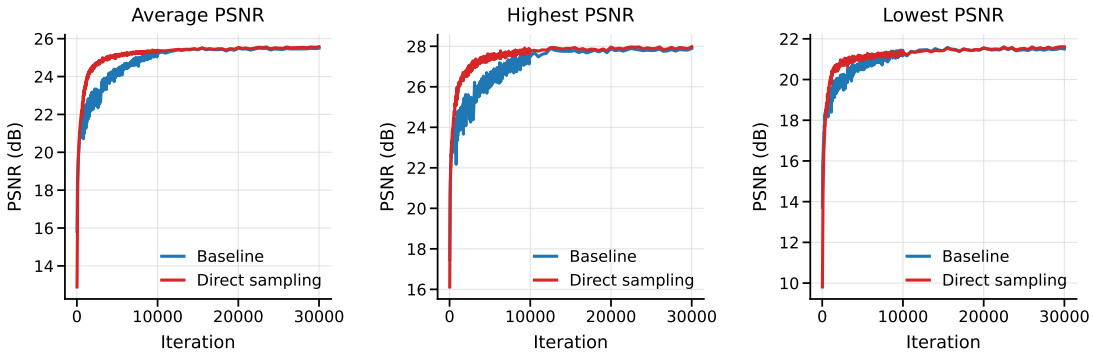


Figure A.10: PSNR curves for Scan 37 in the DTU dataset.

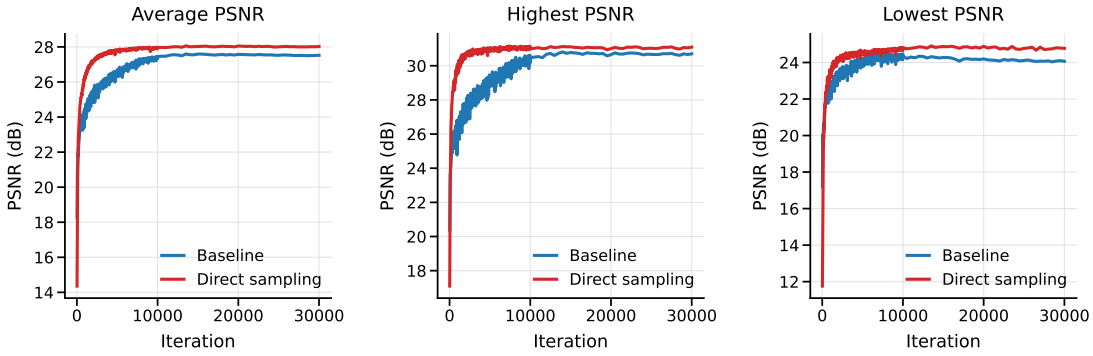


Figure A.11: PSNR curves for Scan 40 in the DTU dataset.

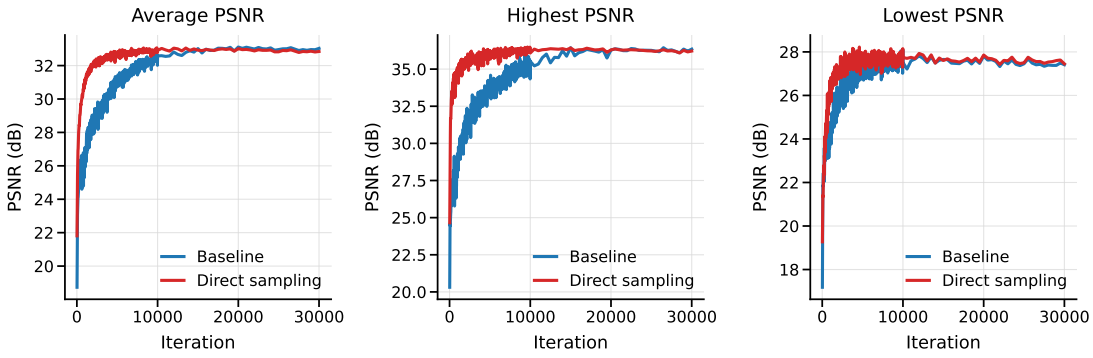


Figure A.12: PSNR curves for Scan 55 in the DTU dataset.

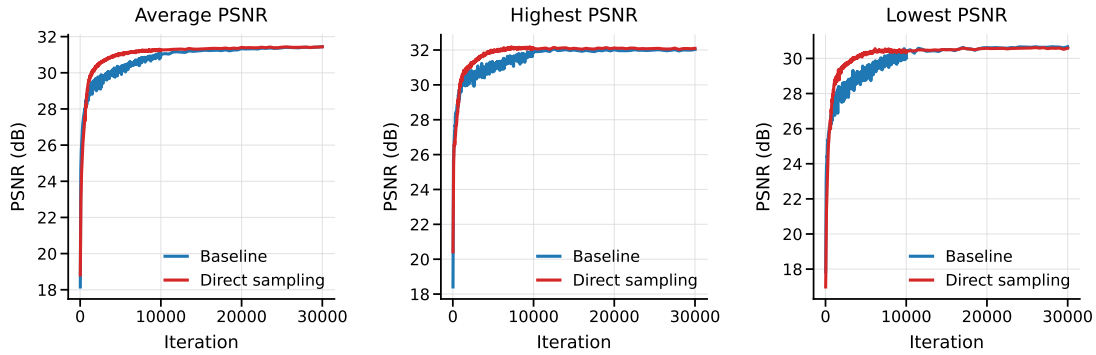


Figure A.13: PSNR curves for Scan 63 in the DTU dataset.

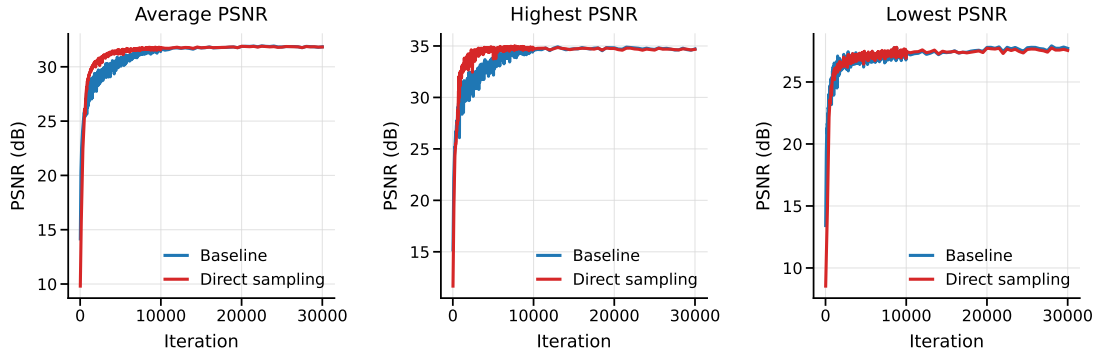


Figure A.14: PSNR curves for Scan 65 in the DTU dataset.

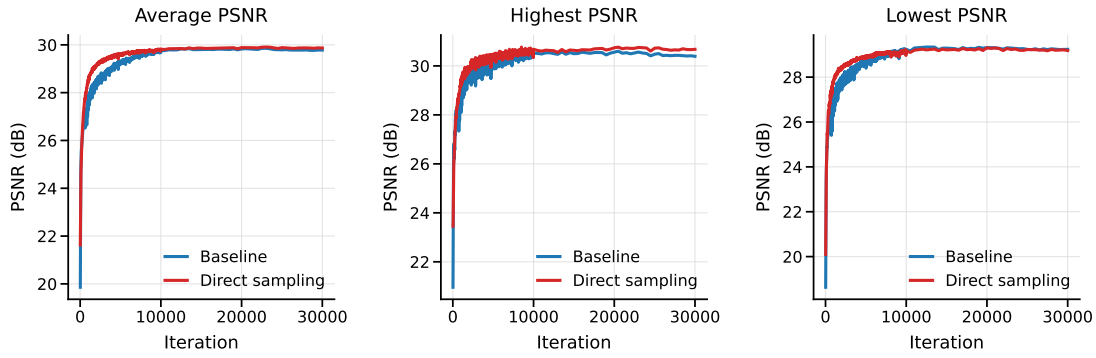


Figure A.15: PSNR curves for Scan 69 in the DTU dataset.

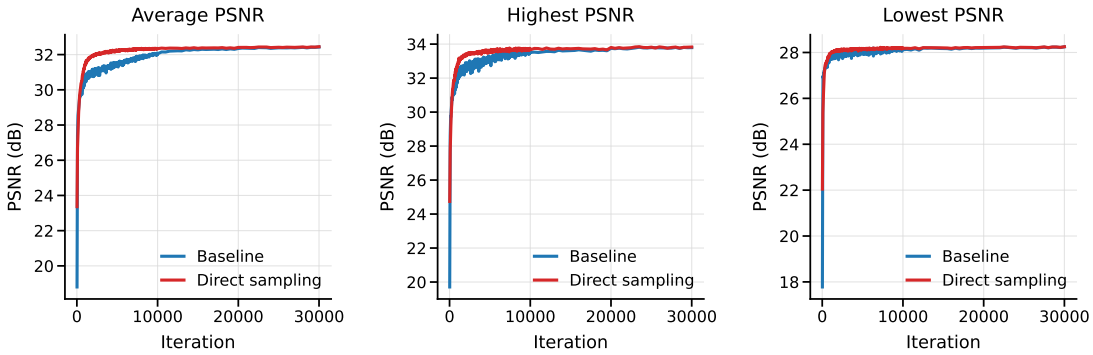


Figure A.16: PSNR curves for Scan 83 in the DTU dataset.

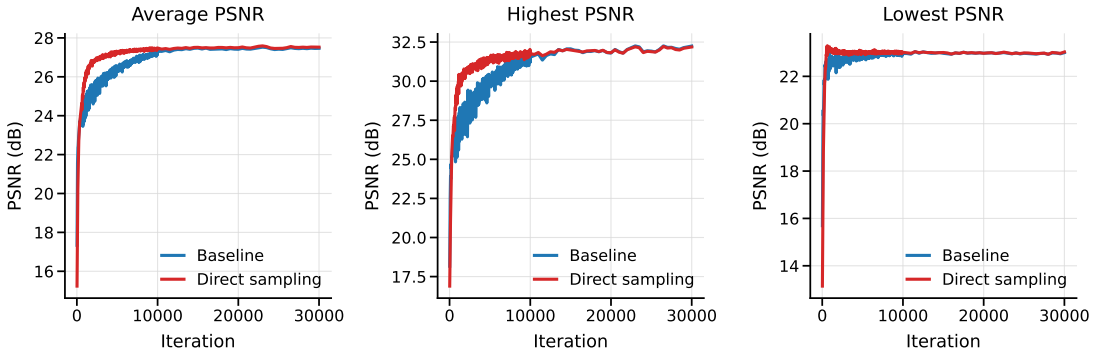


Figure A.17: PSNR curves for Scan 97 in the DTU dataset.

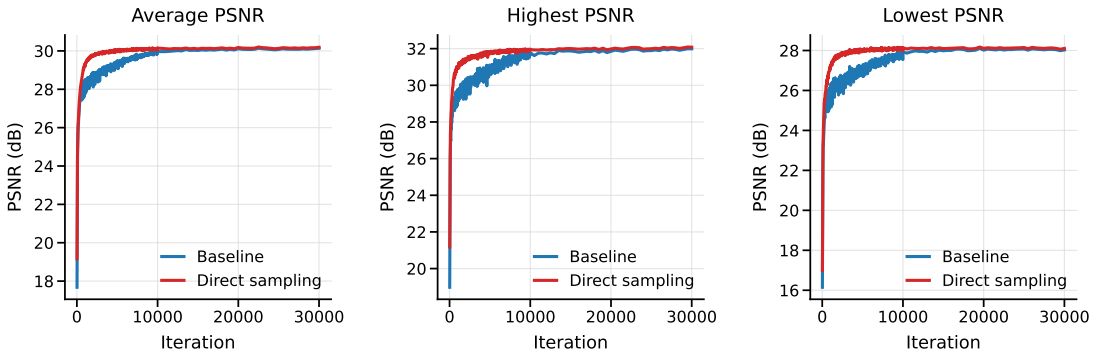


Figure A.18: PSNR curves for Scan 105 in the DTU dataset.

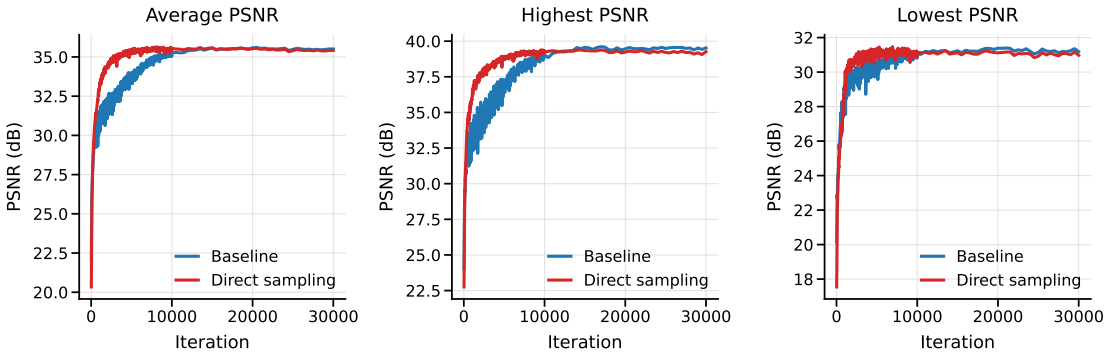


Figure A.19: PSNR curves for Scan 106 in the DTU dataset.

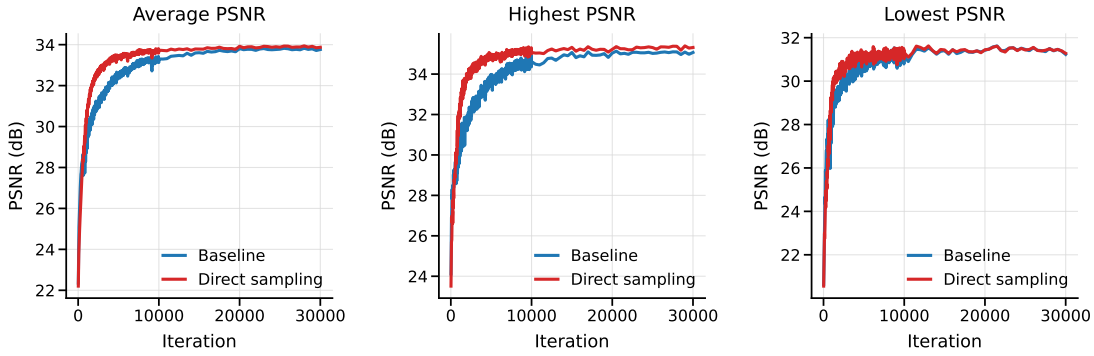


Figure A.20: PSNR curves for Scan 110 in the DTU dataset.

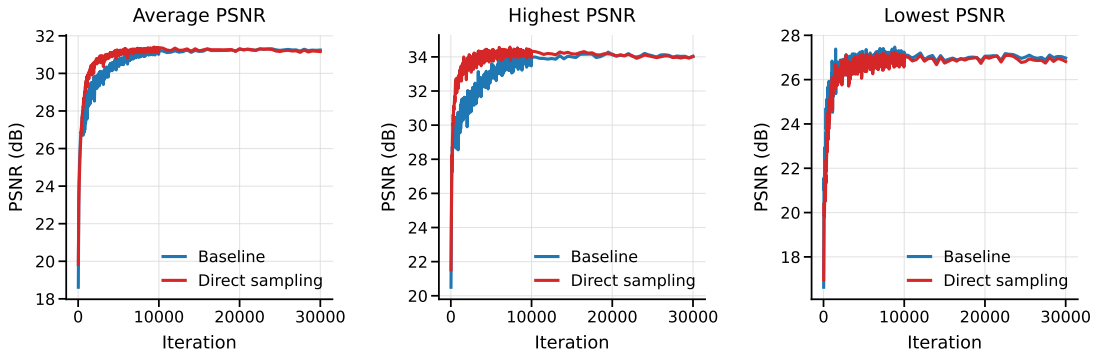


Figure A.21: PSNR curves for Scan 114 in the DTU dataset.

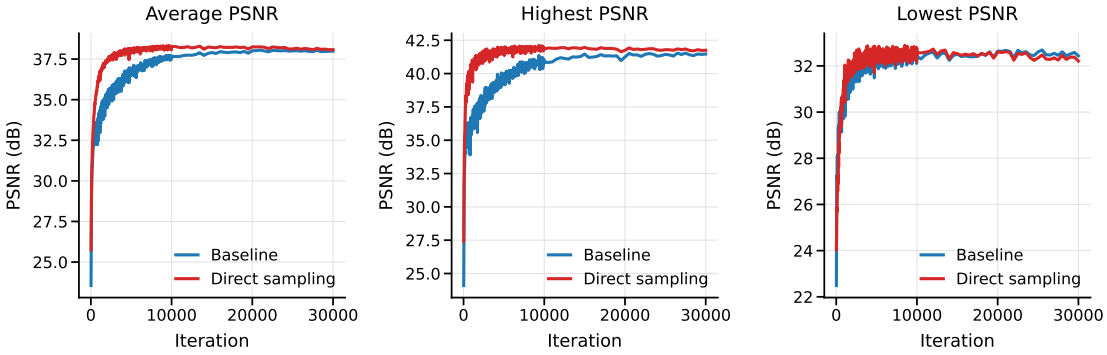


Figure A.22: PSNR curves for Scan 118 in the DTU dataset.

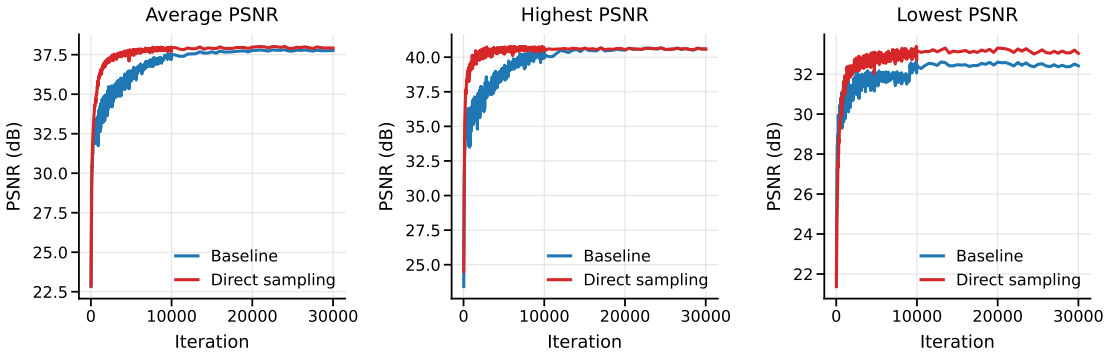


Figure A.23: PSNR curves for Scan 122 in the DTU dataset.

A.1.3 TNT

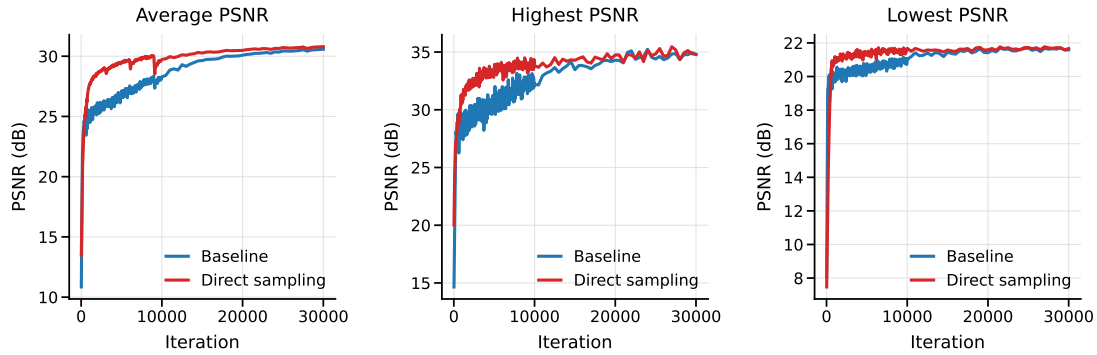


Figure A.24: PSNR curves for Barn in the Tnt dataset.

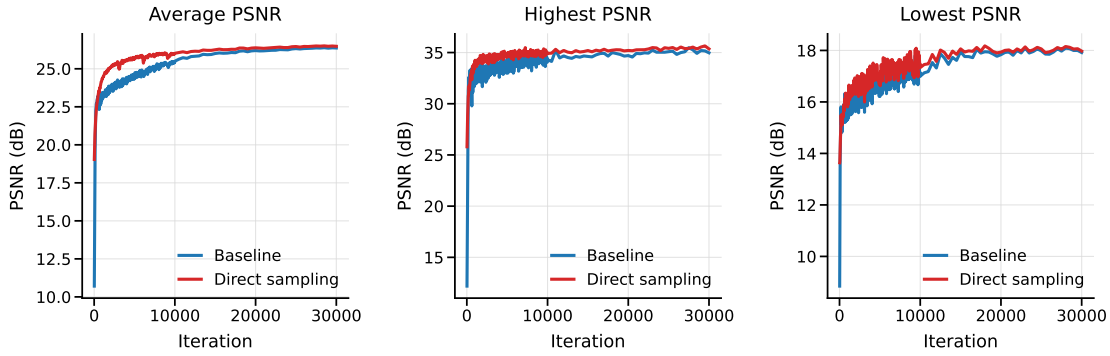


Figure A.25: PSNR curves for Caterpillar in the Tnt dataset.

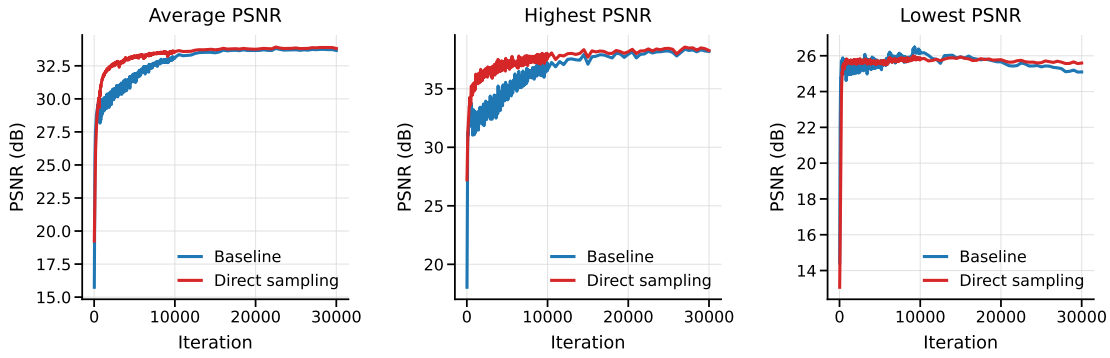


Figure A.26: PSNR curves for Family in the Tnt dataset.

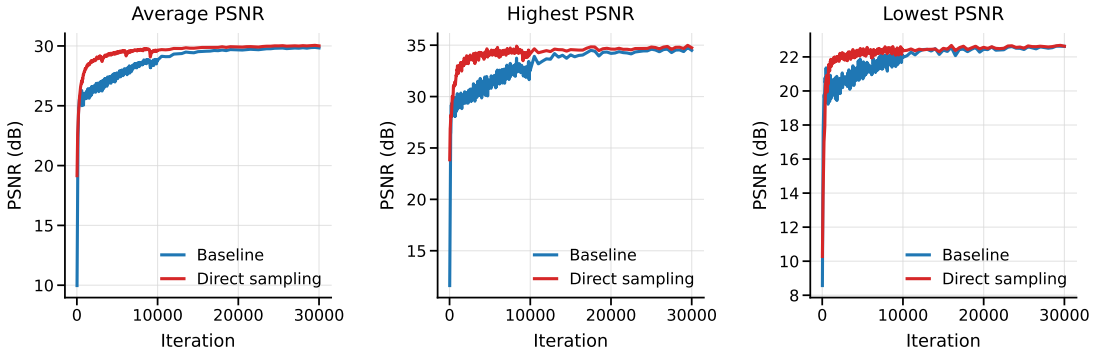


Figure A.27: PSNR curves for Truck in the Tnt dataset.

A.2 SSIM and LPIPS Results

We provide additional evaluation based on different metrics in a limited quantity. The following table A.1 contains values for structural similarity index measure (SSIM) [WBSS04] and Learned Perceptual Image Patch Similarity (LPIPS) [ZIE+18].

dataset	metric	Δ metric, direct sampling initialization minus baseline					
		0.5k	1k	2k	5k	10k	30k
NeRF Synthetic	SSIM ↑	+0.059	+0.044	+0.033	+0.010	+0.001	-0.000
	LPIPS ↓	-0.072	-0.057	-0.053	-0.018	-0.004	-0.000
DTU	SSIM ↑	+0.042	+0.042	+0.031	+0.013	+0.004	+0.002
	LPIPS ↓	-0.070	-0.067	-0.053	-0.028	-0.012	-0.006
TnT	SSIM ↑	+0.025	+0.048	+0.048	+0.037	+0.018	+0.003
	LPIPS ↓	-0.044	-0.074	-0.075	-0.057	-0.026	-0.005

Table A.1: Mean SSIM and LPIPS differences between our direct sampling approach and the baseline over selected iterations. Values are computed as direct sampling initialization minus baseline. Higher SSIM is better, while lower LPIPS is better.

A.3 Gaussian Count Results

dataset	category	ΔN_G [k] ↓				scenes
		2k iter.	5k iter.	10k iter.	Final iter.	
NeRF Synthetic	winner	+81.5	-213.8	-268.4	-268.4	5
DTU		+157.6	-119.3	-274.3	-274.3	8
TnT		none				0
NeRF Synthetic	tie	+175.0	-35.1	-14.0	-14.0	3
DTU		+138.8	+48.0	-4.4	-4.4	3
TnT		none				0
NeRF Synthetic	loser	none				0
DTU		+268.7	+169.9	+117.9	+117.9	4
TnT		+853.9	+623.5	+411.5	+411.5	4
NeRF Synthetic	total	+116.5	-146.8	-173.0	-173.0	8
DTU		+183.5	-8.7	-115.7	-115.7	15
TnT		+853.9	+623.5	+411.5	+411.5	4

Table A.2: Mean absolute difference in the number of Gaussian primitives grouped by the final Gaussian-count category of each scene. Lower-count and higher-count scenes lie outside the 10% tie threshold and contain fewer or more final Gaussians than the baseline, respectively.

dataset	category	N_G [k] ↓			
		2k iter.	5k iter.	10k iter.	Final iter.
NeRF Synthetic	winner	388.1	730.1	774.2	774.2
DTU		166.4	466.9	627.0	627.0
TnT		none			
NeRF Synthetic	tie	318.4	581.1	547.9	547.9
DTU		76.2	171.3	224.7	224.7
TnT		none			
NeRF Synthetic	loser	none			
DTU		81.5	182.0	226.7	226.7
TnT		266.0	404.6	493.0	493.0
NeRF Synthetic	total	362.0	674.2	689.3	689.3
DTU		125.7	331.8	439.8	439.8
TnT		266.0	404.6	493.0	493.0

Table A.3: Mean number of Gaussian primitives used by the baseline grouped by the final outcome of each scene, reported in thousands of primitives.

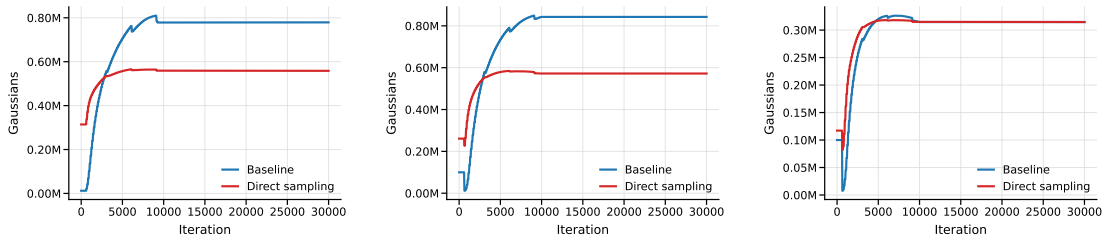
The categories follow the grouping from Table A.2.

dataset	category	N_G [k] ↓			
		2k iter.	5k iter.	10k iter.	Final iter.
NeRF Synthetic	winner	469.6	516.3	505.8	505.8
DTU		324.0	347.6	352.7	352.7
TnT		none			
NeRF Synthetic	tie	493.4	545.9	533.9	533.9
DTU		214.9	219.3	220.3	220.3
TnT		none			
NeRF Synthetic	loser	none			
DTU		350.2	352.0	344.6	344.6
TnT		1120.0	1028.1	904.5	904.5
NeRF Synthetic	total	478.5	527.4	516.3	516.3
DTU		309.2	323.1	324.1	324.1
TnT		1120.0	1028.1	904.5	904.5

Table A.4: Mean number of Gaussian primitives used by our direct sampling approach grouped by the final outcome of each scene, reported in thousands of primitives.

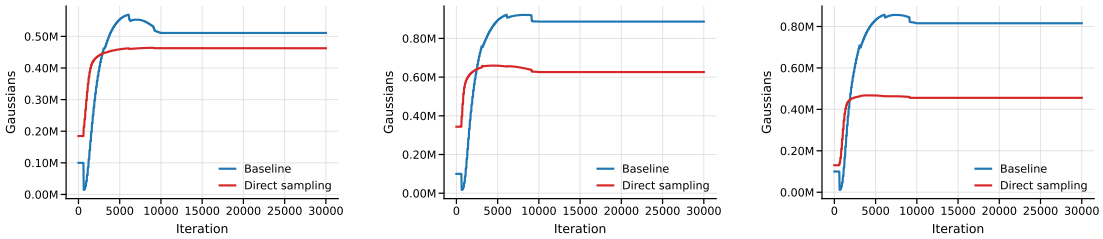
The categories follow the grouping from Table A.2.

A.3.1 NeRF Synthetic



(a) Chair (b) Drums (c) Ficus

Figure A.28: Number of Gaussian primitives over optimization iterations for the NeRF Synthetic scenes Chair, Drums, and Ficus.



(a) Hotdog (b) Lego (c) Materials

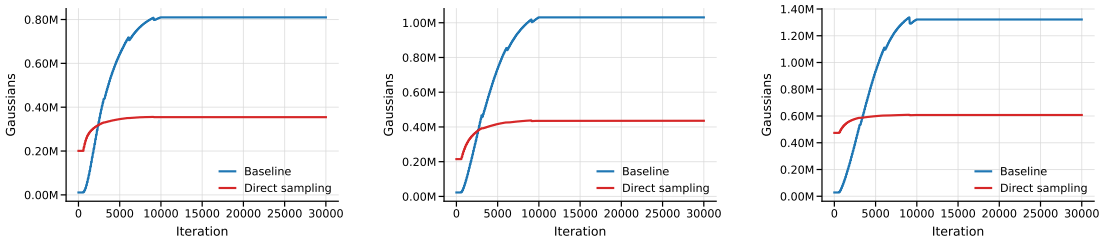
Figure A.29: Number of Gaussian primitives over optimization iterations for the NeRF Synthetic scenes Hotdog, Lego, and Materials.



(a) Mic (b) Ship

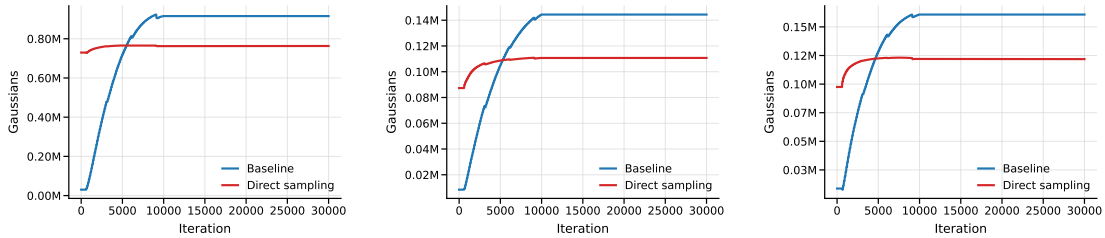
Figure A.30: Number of Gaussian primitives over optimization iterations for the NeRF Synthetic scenes Mic and Ship.

A.3.2 DTU



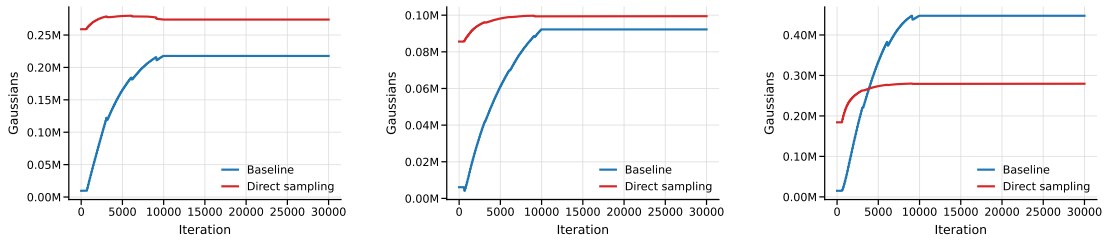
(a) Scan 24 (b) Scan 37 (c) Scan 40

Figure A.31: Number of Gaussian primitives over optimization iterations for the DTU scenes Scan 24, Scan 37, and Scan 40.



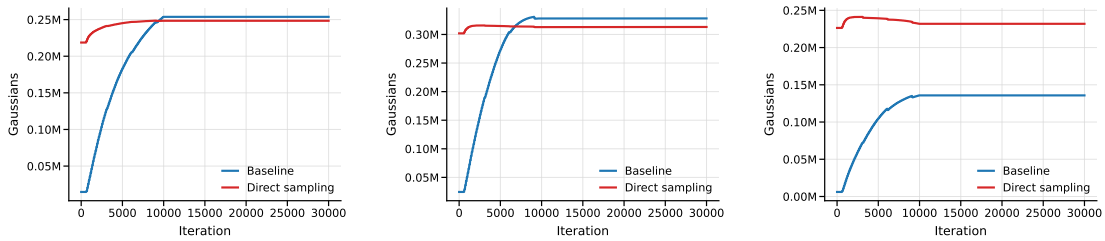
(a) Scan 55 (b) Scan 63 (c) Scan 65

Figure A.32: Number of Gaussian primitives over optimization iterations for the DTU scenes Scan 55, Scan 63, and Scan 65.



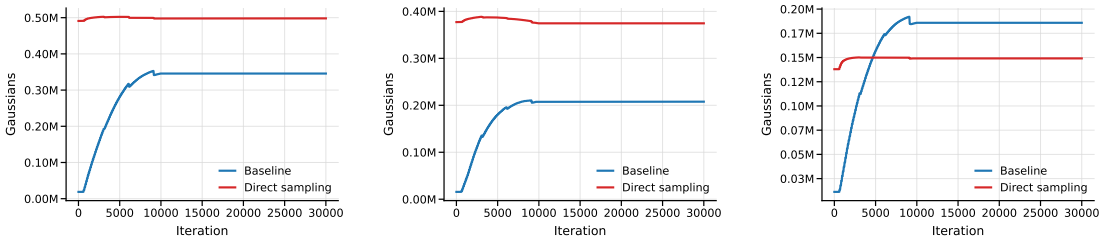
(a) Scan 69 (b) Scan 83 (c) Scan 97

Figure A.33: Number of Gaussian primitives over optimization iterations for the DTU scenes Scan 69, Scan 83, and Scan 97.



(a) Scan 105 (b) Scan 106 (c) Scan 110

Figure A.34: Number of Gaussian primitives over optimization iterations for the DTU scenes Scan 105, Scan 106, and Scan 110.



(a) Scan 114 (b) Scan 118 (c) Scan 122

Figure A.35: Number of Gaussian primitives over optimization iterations for the DTU scenes Scan 114, Scan 118, and Scan 122.

A.3.3 TnT



(a) Barn (b) Caterpillar

Figure A.36: Number of Gaussian primitives over optimization iterations for the TnT scenes Barn and Caterpillar.



(a) Family (b) Truck

Figure A.37: Number of Gaussian primitives over optimization iterations for the TnT scenes Family and Truck.

A.4 Runtime Breakdown

A.4.1 NeRF Synthetic

scene	approach	COLMAP [s] ↓	init [s] ↓	training [s] ↓	total [s] ↓
chair	baseline	52	0	822	873
	direct sampling		31	777	860
drums	baseline	50	0	861	911
	direct sampling		31	751	832
ficus	baseline	41	0	654	695
	direct sampling		23	653	717
hotdog	baseline	34	0	853	887
	direct sampling		30	753	818
lego	baseline	87	0	841	927
	direct sampling		31	733	850
materials	baseline	30	0	803	833
	direct sampling		30	668	729
mic	baseline	23	0	768	791
	direct sampling		31	691	745
ship	baseline	98	0	879	978
	direct sampling		31	837	967
Average	baseline	52	0	810	862
	direct sampling		30	733	815

Table A.5: Runtime breakdown for the NeRF Synthetic dataset. COLMAP reconstruction time on CPU, initialization time, reconstruction time, and total time are reported in seconds.

A.4.2 DTU

scene	approach	COLMAP [s] ↓	init [s] ↓	training [s] ↓	total [s] ↓
Scan 24	baseline	96	0	1285	1382
	direct sampling		28	1015	1140
Scan 37	baseline	80	0	1130	1210
	direct sampling		28	864	972
Scan 40	baseline	123	0	1365	1488
	direct sampling		28	1045	1196
Scan 55	baseline	94	0	1083	1178
	direct sampling		28	970	1093
Scan 63	baseline	29	0	839	868
	direct sampling		28	774	832
Scan 65	baseline	23	0	900	923
	direct sampling		28	825	877
Scan 69	baseline	31	0	972	1004
	direct sampling		28	932	992
Scan 83	baseline	18	0	773	791
	direct sampling		28	734	780
Scan 97	baseline	44	0	1004	1048
	direct sampling		28	882	955
Scan 105	baseline	48	0	924	972
	direct sampling		28	873	949
Scan 106	baseline	65	0	962	1027
	direct sampling		28	892	985
Scan 110	baseline	33	0	898	931
	direct sampling		28	877	938
Scan 114	baseline	63	0	1020	1083
	direct sampling		28	995	1086
Scan 118	baseline	42	0	872	913
	direct sampling		28	886	956
Scan 122	baseline	31	0	844	875
	direct sampling		28	781	841
Average	baseline	55	0	991	1046
	direct sampling		28	890	973

Table A.6: Runtime breakdown for the DTU dataset. COLMAP reconstruction time on CPU, initialization time, reconstruction time, and total time are reported in seconds.

A.4.3 TnT

scene	approach	COLMAP [s] ↓	init [s] ↓	training [s] ↓	total [s] ↓
Barn	baseline	2324	0	1002	3326
	direct sampling		219	1076	3619
Caterpillar	baseline	1547	0	897	2444
	direct sampling		210	969	2726
Family	baseline	259	0	751	1010
	direct sampling		86	734	1079
Truck	baseline	964	0	897	1860
	direct sampling		140	944	2048
Average	baseline	1274	0	887	2160
	direct sampling		164	931	2368

Table A.7: Runtime breakdown for the TnT dataset. COLMAP reconstruction time on CPU, initialization time, reconstruction time, and total time are reported in seconds.

APPENDIX B

Evaluation Parameters

Parameter	NeRF Synthetic	DTU	TnT
`error_threshold`[a]	0.475	0.675	0.45
`base_prob` - p[b]	0.05	0.06	0.05
`knn_n` - k[c]	4	4	4
`knn_p` - γ[d]	2	2	2
`knn_epsilon` - ϵ_q	10^{-6}	10^{-6}	10^{-6}
`knn_stride` - s[e]	4	4	4
`neighbourhood_angle_criteria` - α[f]	1.0472	1.0472	1.5970
`dav2_target_width`	770	770	384
`feature_sigma` - σ_C[g]	50	50	49
KNN implementation	PyTorch3D	PyTorch3D	SciPy cKDTree
`position_lr_scale_factor` - l_{init}[h]	0.07	0.08	0.09

Table B.1: Parameters used for the proposed Gaussian initialization and subsequent optimization across the evaluated datasets.

a Each scene point reconstructed by COLMAP has a reprojection error. The higher the error, the less accurately the scene point represents the actual true position it aims to capture. To filter out highly erroneous scene points, we set these thresholds; see section 3.3

b lowest possible probability p for sampling; see equation 3.4

c number of nearest neighbors k, used for interpolating depth corrections; see section 3.3

d exponent γ for weighted interpolation of nearest neighbors; see equation 3.7

e Stride s; see section 3.3

f Relative angle α between view vectors of the cameras that generated Gaussians, which decides whether a specific set of Gaussians ought to be used for the penalty map $\tilde{P}$; see section 3.2

g Standard deviation σ of Gaussian blur for coverage map; see section 3.2

h positional learning scale for the actual optimization pipeline; see section 2.1

Parameter	NeRF Synthetic	DTU	TnT
`lambda_normal_render_depth`	0.01	0.01	0.01
`lambda_normal_smooth`	0.01	0.01	0.01
`lambda_mask_entropy`	0.1	0.1	0.1
`lambda_depth_var`	10^{-2}	10^{-2}	10^{-2}
`densification_interval`	default	default	100
`densify_grad_normal_threshold`	default	999	10^{-8}
Optimization iterations	30,000	30,000	30,000

Table B.2: R3DGS optimization parameters used by the baseline [GGL+24] and our proposed direct sampling initialization approach.

APPENDIX C

Experimental Environment

Component	Version / Specification
Operating system	Ubuntu 24.04.4 LTS
CPU	AMD Ryzen 9 5950X 16-Core Processor (32 threads)
GPU	NVIDIA GeForce RTX 5090 (32 GiB)
NVIDIA driver	595.71.05
CUDA Toolkit	12.8
NVCC	12.8.61
COLMAP	COLMAP 3.9.1
Python	3.10.19
PyTorch	2.8.0+cu128
torchvision	0.23.0+cu128
NumPy	2.2.6
SciPy	1.15.0
OpenCV	4.12.0.88
PyTorch3D	0.7.8
Kaolin	0.18.0
Open3D	0.19.0
nvdiffrast	0.3.3

Table C.1: Hardware and software environment used for the evaluation.

The corresponding source code used is available on GitLab:
https://gitlab.cg.tuwien.ac.at/stef/gaussian-splatting

The tests were conducted under the commit hash:
a4bbff5c92dc998813ce39590f2fd371656c2a85

Overview of Generative AI Tools Used

Grammarly was used throughout the entire writing process of this thesis to improve the language and correct the grammar of all written text.

Codex (GPT-5.5 medium) was used to assist in creating all tables of this thesis.

ChatGPT (GPT-5.6 Sol) was used to correct the already existing German version of the abstract.

List of Figures

2.1 3DGS pipeline 7
2.2 3DGS densification illustration 8
2.3 DAV2 depth estimate 10
2.4 On-the-fly sampling pipeline 12

3.1 System overview 16
3.2 Conceptual illustration of LoG 17
3.3 LoG spawning showcase 18
3.4 direct sampling initialization sampling pipeline 20
3.5 direct sampling initialization depth estimation pipeline 22
3.6 Scale initialization comparison 24

4.1 Average PSNR over iteration result 27
4.2 Average PSNR over time result 28
4.3 Average Gaussian count over iterations result 30

A.1 PSNR curves for Chair in the NeRF Synthetic dataset. 39
A.2 PSNR curves for Drums in the NeRF Synthetic dataset. 40
A.3 PSNR curves for Ficus in the NeRF Synthetic dataset. 40
A.4 PSNR curves for Hotdog in the NeRF Synthetic dataset. 40
A.5 PSNR curves for Lego in the NeRF Synthetic dataset. 41
A.6 PSNR curves for Materials in the NeRF Synthetic dataset. 41
A.7 PSNR curves for Mic in the NeRF Synthetic dataset. 41
A.8 PSNR curves for Ship in the NeRF Synthetic dataset. 42
A.9 PSNR curves for Scan 24 in the DTU dataset. 42
A.10 PSNR curves for Scan 37 in the DTU dataset. 43
A.11 PSNR curves for Scan 40 in the DTU dataset. 43
A.12 PSNR curves for Scan 55 in the DTU dataset. 43
A.13 PSNR curves for Scan 63 in the DTU dataset. 44
A.14 PSNR curves for Scan 65 in the DTU dataset. 44
A.15 PSNR curves for Scan 69 in the DTU dataset. 44
A.16 PSNR curves for Scan 83 in the DTU dataset. 45
A.17 PSNR curves for Scan 97 in the DTU dataset. 45
A.18 PSNR curves for Scan 105 in the DTU dataset. 45

A.19 PSNR curves for Scan 106 in the DTU dataset. 46
A.20 PSNR curves for Scan 110 in the DTU dataset. 46
A.21 PSNR curves for Scan 114 in the DTU dataset. 46
A.22 PSNR curves for Scan 118 in the DTU dataset. 47
A.23 PSNR curves for Scan 122 in the DTU dataset. 47
A.24 PSNR curves for Barn in the Tnt dataset. 48
A.25 PSNR curves for Caterpillar in the Tnt dataset. 48
A.26 PSNR curves for Family in the Tnt dataset. 48
A.27 PSNR curves for Truck in the Tnt dataset. 49
A.28 No. of Gaussians: Chair, Drums, Ficus (NeRF Synthetic) 51
A.29 No. of Gaussians: Hotdog, Lego, Materials (NeRF Synthetic) 52
A.30 No. of Gaussians: Mic, Ship (NeRF Synthetic) 52
A.31 No. of Gaussians: Scan 24, Scan 37, Scan 40 (DTU) 52
A.32 No. of Gaussians: Scan 55, Scan 63, Scan 65 (DTU) 53
A.33 No. of Gaussians: Scan 69, Scan 83, Scan 97 (DTU) 53
A.34 No. of Gaussians: Scan 105, Scan 106, Scan 110 (DTU) 53
A.35 No. of Gaussians: Scan 114, Scan 118, Scan 122 (DTU) 54
A.36 No. of Gaussians: Barn, Caterpillar (TnT) 54
A.37 No. of Gaussians: Family, Truck (TnT) 54

List of Tables

4.1 mean difference of PSNR over iteration 27
4.2 categorized relative difference of number of Gaussians over iteration . . . 31

A.1 SSIM and LPIPS differences over iterations 49
A.2 categorized absolute difference of number of Gaussians over iteration . . . 50
A.3 categorized baseline number of Gaussians over iteration 50
A.4 categorized direct sampling number of Gaussians over iteration 51
A.5 NeRF Synthetic runtime breakdown 55
A.6 DTU runtime breakdown 56
A.7 TnT runtime breakdown 57

B.1 direct sampling initialization specific evaluation parameters 59
B.2 R3DGS specific evaluation parameters 60

C.1 experimental environment 61

List of Algorithms

3.1 Direct Sampling Initialization 16
3.2 Sampling mask construction 19

Acronyms

3DGS 3D Gaussian Splatting. ix, xi, 1–9, 11, 65

BA bundle adjustment. 10

BRDF bidirectional reflectance distribution function. 3

D-SSIM differentiable structural similarity index measure. 7

DAV2 Depth Anything V2. 2–5, 9, 12, 19, 21, 22, 33, 35, 36, 65

LoG Laplacian of Gaussian. 2, 3, 12, 15, 17–20, 33, 65

LPIPS Learned Perceptual Image Patch Similarity. 49

MADM mean absolute deviation from the median. 3, 12, 19, 22

NeRF Neural Radiance Field. ix, xi, 1, 3, 22, 25, 28, 29, 33–35

PBR physically based rendering. 3, 9

PSNR peak signal-to-noise ratio. ix, xi, 3, 26–28, 33, 34, 36, 39, 65, 67

R3DGS Relightable3DGaussian. ix, xi, 2–5, 8, 9, 25, 33, 39, 60, 67

SfM Structure-from-Motion. 1, 4, 5, 7, 8, 10, 11

SH spherical harmonics. 6, 7

SSIM structural similarity index measure. 49

TnT Tanks and Temples. ix, xi, 3, 22, 25, 28, 29, 33–36

Bibliography

[CE54] Philip J. Clark and Francis C. Evans. Distance to nearest neighbor as a measure of spatial relationships in populations. *Ecology*, 35(4):445–453, 1954.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[GGL+24] Jian Gao, Chun Gu, Youtian Lin, Zhihao Li, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3d gaussians: realistic point cloud relighting with brdf decomposition and ray tracing. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XLV*, pages 73–89, Milan, Italy. Springer-Verlag, 2024.

[JDV+14] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014.

[KKLD23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), July 2023.

[KLR+22] Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. Neural point catacaustics for novel-view synthesis of reflections. *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, 41(6):201, 2022.

[KMK+24] 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 Trans. Graph.*, 43(4), July 2024.

[KPLD21] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. *Computer Graphics Forum*, 40(4):29–43, 2021.

[KPZK17] 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.

[LCL+25] Haotong Lin, Sili Chen, Junhao Liew, Donny Y. Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: recovering the visual space from any views, 2025.

[MESK22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), July 2022.

[MSL+25] Andreas Meuleman, Ishaan Shah, Alexandre Lanvin, Bernhard Kerbl, and George Drettakis. On-the-fly reconstruction for large-scale novel view synthesis from unposed images. *ACM Trans. Graph.*, 44(4), July 2025.

[MST+21] 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.

[SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[TMHF00] Bill Triggs, Philip Mclauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle Adjustment – A Modern Synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Lecture Notes in Computer Science*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372, Corfu, Greece. Springer-Verlag, September 2000. This work was supported in part by the European Commission Esprit LTR project CUMULI (B. Triggs), the UK EPSRC project GR/L34099 (P.McLauchlan), and the Royal Society (A. Fitzgibbon).

[WBSS04] 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.

[YFT+21] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: radiance fields without neural networks, 2021.

[YKH+24] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Vancouver, BC, Canada. Curran Associates Inc., 2024.

[ZIE+18] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

[ZPBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen Baar, and Markus Gross. Ewa volume splatting. *IEEE Visualization*, January 2001.