

Hierarchical Dense Edge-Guided Initialization for Real-Time 3D Gaussian Splatting

Timon Coucke

Student number: 02107660

Supervisors: Prof. dr. ir. Glenn Van Wallendael, Prof. dr. Peter Lambert

Master's dissertation submitted in order to obtain the degree of Master of Science in Computer Science

Academic year 2025-2026

Hierarchical Dense Edge-Guided Initialization for Real-Time 3D Gaussian Splatting

Timon Coucke

Student number: 02107660

Supervisors: Prof. dr. ir. Glenn Van Wallendael, Prof. dr. Peter Lambert

Master's dissertation submitted in order to obtain the degree of Master of Science in Computer Science

Academic year 2025-2026

Acknowledgements

I would first like to thank my co-supervisor, ir. Bert Ramlot, for our weekly meetings and for his clear guidance throughout the thesis. His feedback and advice helped me make progress at every stage.

I am also grateful to my supervisors, prof. dr. ir. Glenn Van Wallendael and prof. dr. Peter Lambert, for giving me the opportunity to work on this topic.

Finally, I want to thank my parents, sister, girlfriend and friends for their constant support, motivation and patience during this entire period.

Admission to loan

“The author(s) gives (give) permission to make this master’s dissertation available for consultation and to copy parts of this master’s dissertation for personal use. In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master’s dissertation.”

Toegang tot bruikleen

“De auteur(s) geeft (geven) de toelating dit afstudeerwerk voor consultatie beschikbaar te stellen en delen van het afstudeerwerk te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit dit afstudeerwerk.”

Timon Coucke, May 2026

Thesis as part of an exam

“This master’s dissertation is part of an exam. Any comments formulated by the assessment committee during the oral presentation of the master’s dissertation are not included in this text.”

Timon Coucke, May 2026

Abstract

English

This thesis investigates the initialization of 3D Gaussian Splatting (3DGS), a recent rendering technique that models a 3D scene as a collection of colored 3D Gaussians optimized to match input images and enable real-time, photorealistic view synthesis. Incorporating Light Detection and Ranging (LiDAR) data addresses this by supplying precise, metric depth measurements, resulting in more consistent and geometrically accurate priors compared to learned depth estimation alone.

We exploit these advantages by constructing a hierarchical, edge-guided tree partition of RGBD images that leverages both photometric and geometric edges to decide where Gaussians are placed in each frame. Every leaf region of this tree spawns a single Gaussian whose position and scale are determined from the leaf's extent, the completed depth, and the camera intrinsics and extrinsics, producing a dense yet structured initialization that concentrates Gaussians in visually and geometrically complex areas while using fewer primitives in homogeneous regions.

Experiments demonstrate faster early convergence, improved or competitive reconstruction quality at lower Gaussian budgets, and a more favorable quality-complexity trade-off compared to a strong LiDAR-based baseline. These results show that a more intelligent, edge-guided initialization by itself can improve both the efficiency and the performance of 3DGS pipelines.

The implementation of the proposed method is available as open-source code on Github¹.

Nederlands

Deze thesis onderzoekt de initialisatie van 3DGS, een recente renderingstechniek die een 3D-scène modelleert als een verzameling gekleurde 3D Gaussians, geoptimaliseerd om inputbeelden te benaderen en real-time, fotorealistische view synthesis mogelijk te maken. Het integreren van LiDAR-data pakt hierbij een belangrijk probleem aan door nauwkeurige, metrische dieptemetingen te leveren, wat resulteert in meer consistente en geometrisch correcte priors in vergelijking met louter geleerd depth estimation.

Wij benutten deze voordelen door een hiërarchische, edge-guided boomstructuur te construeren over RGBD-beelden, die zowel fotometrische als geometrische randen gebruikt om te bepalen waar Gaussians per frame worden geplaatst. Elke bladregio van deze boom genereert één Gaussian, waarvan de positie en schaal worden afgeleid uit de omvang van de regio, de aangevulde diepte en de camera-intrinsics en -extrinsics. Dit leidt tot een dichte maar gestructureerde initialisatie die Gaussians concentreert in visueel en geometrisch complexe gebieden, terwijl in homogene regio's minder primitieve elementen worden gebruikt.

Experimenten tonen snellere initiële convergentie, verbeterde of competitieve reconstructiekwaliteit bij lagere aantallen Gaussians, en een gunstigere kwaliteit-complexiteit trade-off in vergelijking met een sterke LiDAR-gebaseerde baseline. Deze resultaten tonen aan dat een meer intelligente, edge-guided initialisatie op zichzelf al zowel de efficiëntie als de prestaties van 3DGS-pipelines kan verbeteren.

De implementatie van de voorgestelde methode is als open-source code beschikbaar op Github¹.

Keywords: 3D Gaussian Splatting, LiDAR, Multi-modal Edge Detection, Dense Initialisation, Hierarchical Partitioning

¹<https://github.com/LGDTimou/gaussian-lic-research>

Hierarchical Dense Edge-Guided Initialization for Real-Time 3D Gaussian Splatting

Timon Coucke

Supervisors: prof. dr. ir. Glenn van Wallendael, prof. dr. Peter Lambert

Co-supervisor: ir. Bert Ramlot

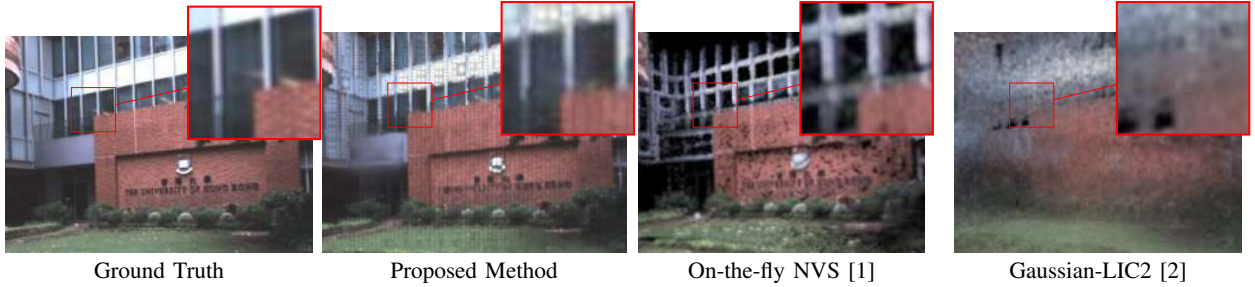


Fig. 1: Visual comparison of initialization quality on a single frame without optimization. **Left:** Ground truth RGB image. **Center-left:** Our proposed dense, edge-guided initialization produces a clear representation with well-preserved detail in complex regions and adequate coverage in homogeneous areas. **Center-right:** On-the-fly NVS [1], a 3DGS reconstruction method that incrementally spawns Gaussian primitives using direct, Laplacian of the Gaussian (LoG)-based weighted sampling of primitive positions and shapes, captures high-detail regions effectively but leaves gaps in uniform areas, resulting in visible black holes. **Right:** Gaussian-LIC2 [2], a LiDAR-Inertial-Camera Gaussian Splatting SLAM system, initializes Gaussians uniformly from LiDAR depth and depth-completed maps, resulting in coarse, blurry coverage despite using a significant amount of primitives.

Abstract—3D Gaussian Splatting (3DGS) is a rendering technique that represents a scene as a collection of 3D Gaussians optimized to match input images. This thesis presents a dense initialization method for 3DGS that allows for real-time initialization and optimization by leveraging LiDAR-enhanced RGB-D data. A hierarchical, edge-guided tree partition is constructed over each image, using both photometric and geometric cues to guide the placement of Gaussians. Each leaf region generates a single Gaussian whose position and scale are derived from the leaf’s spatial extent, the completed depth, and camera intrinsics and extrinsics, resulting in a structured distribution that concentrates primitives in complex regions while reducing redundancy in homogeneous areas. Experiments show faster early convergence, competitive or improved reconstruction quality at lower Gaussian counts, and a more favorable quality–complexity trade-off compared to a state-of-the-art real-time LiDAR-based baseline. These results demonstrate that improving the initialization alone can significantly enhance both the efficiency and performance of 3DGS pipelines. The implementation of the proposed method is available as open-source code at <https://github.com/LGDTimou/gaussian-lic-research>.

I. INTRODUCTION

3D Gaussian Splatting (3DGS) [3] represents a scene as a collection of optimizable 3D Gaussians, initialized from a sparse point cloud obtained via Structure from Motion (SfM) [4] or directly from LiDAR data [2]. A better initializa-

tion does not necessarily guarantee higher final reconstruction quality after full convergence, but it strongly influences convergence speed and the quality that can be achieved under constrained Gaussian budgets. In addition, when the initialization is very sparse or completely misses certain regions, these areas are difficult to recover during optimization and the final reconstruction quality can be fundamentally limited [5], [6].

RGB-only pipelines often produce unreliable point clouds in regions with low texture or challenging appearance, as feature matching fails to provide accurate depth estimates in these areas. Incorporating LiDAR data addresses this by supplying precise, metric depth measurements, resulting in more consistent and geometrically accurate priors than learned depth estimation alone. However, even with LiDAR-augmented pipelines, existing initialization strategies often rely on sparse, feature-based sampling or uniform distributions that do not adequately capture scene complexity, leading to inefficient use of the Gaussian budget and slower convergence. Recent dense initialization methods improve coverage but typically require computationally expensive multi-view matching and do not leverage LiDAR geometric priors, making them unsuitable for real-time, online applications where views arrive

sequentially.

This work proposes a dense, edge-guided initialization strategy based on a hierarchical tree partition of RGB-D images. The partitioning leverages both photometric and geometric cues to adaptively subdivide each image into regions that reflect scene complexity. Each leaf region generates a single Gaussian whose position and scale are derived from its spatial extent, depth information, and camera intrinsics and extrinsics, resulting in a structured distribution that concentrates primitives in complex regions while reducing redundancy in homogeneous areas. The method supports real-time initialization and optimization.

Experimental results demonstrate faster early convergence, competitive or improved reconstruction quality at lower Gaussian counts, and a more favorable quality-complexity trade-off compared to a state-of-the-art real-time LiDAR-based baseline.

The main contributions of this work are:

- A hierarchical tree-partitioning strategy based on fused photometric (RGB) and geometric (depth) edge detection to adaptively partition scenes according to their complexity
- A coverage-aware extension mechanism that prevents redundant Gaussian placement in already-represented regions during incremental mapping
- A real-time, online, dense initialization pipeline that integrates LiDAR depth completion, edge-guided partitioning, and structured Gaussian placement without modification to the underlying optimization framework
- Experimental validation showing that the proposed method achieves faster convergence and improved quality-complexity trade-offs compared to Gaussian-LIC2 [2], particularly at lower Gaussian budgets

II. RELATED WORK

A. LiDAR Integration with 3DGS

Standard 3DGS [3] relies on Structure-from-Motion (SfM) [4] to generate initial point clouds. In practice, SfM often produces incomplete geometry in textureless regions where few features can be matched, and introduces geometric noise under challenging lighting conditions. LiDAR-based approaches such as LI-GS [7] address these issues by providing direct distance measurements, leading to higher geometric accuracy and allowing the same quality to be achieved with fewer Gaussians. Recent work integrates LiDAR into 3DGS-based pipelines across a range of applications: DrivingGaussian [8] reconstructs dynamic surrounding scenes for autonomous driving, TCLC-GS [9] achieves real-time rendering at up to 120 FPS, and SplatAD [10] enables joint camera and LiDAR sensor rendering. For large-scale mapping, Gaussian-UDSR [11] reconstructs unbounded dynamic scenes, while EnerGS [12] models partial LiDAR observations as continuous energy fields. LiDAR-inertial-camera SLAM systems such as Coco-LIC [13] provide synchronized poses and world-frame alignment for downstream reconstruction. Building on this foundation, methods like Gaussian-LIC2 [2], LIVE-GS [14],

and RMGS-SLAM [15] tightly integrate LiDAR, inertial, and visual measurements directly into the Gaussian map representation, enabling real-time photorealistic mapping in large-scale outdoor environments.

B. Depth Completion

While LiDAR provides accurate measurements, its sparse nature requires completion to provide full scene coverage. Monocular methods like MiDaS [16] and DepthAnything V2 [17] estimate dense depth but suffer from scale ambiguity. LiDAR-based depth completion outperforms monocular approaches by fusing sparse LiDAR with RGB guidance [18]. Gaussian-LIC2 leverages SPNet [19] to fill LiDAR-blind areas.

C. Dense Initialization

Most existing dense initialization strategies operate in an offline setting, where all input views are available before optimization. They improve convergence by generating complete Gaussian coverage from the start, eliminating iterative densification [20]. EDGS [20] finds dense pixel correspondences across multiple views using matching networks like RoMa [21], then triangulates these matched pixels to directly initialize millions of 3D Gaussians in a single step. DUST3R-based pipelines [22] generate globally consistent point clouds from fixed image sets and have been integrated into multiple systems [23], [24], [25]. SDI-GS [26] combines MVS priors with segmentation-based downsampling, while MVS-Splatting [27] and NexusGS [28] leverage dense depth estimation from multi-view stereo. DensifyBeforehand [29] combines sparse LiDAR with monocular depth for content-aware sampling prior to optimization. Because these methods rely on global multi-view information, they cannot be directly applied in truly online settings where frames arrive sequentially, and their computationally expensive matching makes them difficult to use under strict real-time constraints.

D. Online Initialization

Existing online initialization methods differ in how strongly they adapt Gaussian placement to scene complexity. On-the-fly NVS [1] applies Laplacian-of-Gaussian (LoG) operators to detect photometric edges and concentrate Gaussians in textured regions, reducing densification overhead. However, this probabilistic sampling strategy lacks spatial coverage guarantees and captures only photometric complexity. Conversely, Splat-LOAM [30] weights initialization using depth map derivatives to focus on geometric discontinuities, but ignores photometric detail. Gaussian-LIC2 [2] provides strong geometric priors through LiDAR yet distributes Gaussians uniformly, without adapting to local scene complexity. All three operate in online pipelines where the Gaussian representation is updated incrementally as new frames or scans arrive, but none jointly exploits both photometric and geometric cues for detail-aware online initialization as our method does.

III. METHODOLOGY

The proposed real-time, online, dense initialization pipeline for 3DGS is illustrated in Figure 2. Posed images and a sparse point cloud are obtained from raw RGB, LiDAR, and IMU data using Coco-LIC [13]. Dense depth maps are then predicted via depth completion (Section III-A), after which photometric and geometric edges are detected and fused into a unified edge representation (Section III-B). This edge information guides the construction of a hierarchical tree that adaptively partitions the scene (Section III-C) and supports coverage-aware refinement to avoid redundant Gaussians in already represented regions (Section III-D). In the leaves of this structure, Gaussians are initialized with position, color, opacity, and scale parameters, and finally refined through differentiable rendering during the optimization stage (Section III-E).

A. Depth Completion

Dense depth maps are generated using SPNet [19], which fuses sparse LiDAR measurements with RGB guidance to fill LiDAR-blind regions while preserving geometric structure. This depth completion step provides a dense geometric prior that enhances both the edge detection process and the subsequent 3D Gaussian initialization.

B. Edge Detection

Both photometric and geometric edges are extracted to guide the later spatial partitioning. Photometric edges are detected from the RGB image using a Canny edge detector [31], capturing texture boundaries. Geometric edges are extracted from the completed depth map using the same operator, recovering structural discontinuities such as object silhouettes and depth jumps. The final edge representation is obtained by combining both maps via a pixel-wise logical OR, $E = E_{rgb} \cup E_{depth}$, ensuring that regions containing either type of edge are subdivided more finely. Canny thresholds for RGB and depth were tuned independently to balance boundary preservation and noise suppression in each modality.

C. Hierarchical Tree Partitioning

The combined edge map guides the construction of a hierarchical tree that adaptively partitions each image into regions of varying size. The tree operates under a fixed leaf budget, with each final leaf rectangle defining the spatial support for a single Gaussian primitive. This adaptive layout concentrates Gaussians in complex regions while covering homogeneous areas with fewer, larger primitives as seen in Figure 3.

The tree construction operates via a breadth-first frontier expansion rather than depth-first subdivision, to ensure balanced refinement across the image without localized over-subdivision. Nodes that meet the splitting criteria are divided to produce child rectangles for the next level. Splitting terminates when a node’s corresponding rectangle meets any of the following criteria: (1) one of its dimensions falls below a minimum size threshold, (2) it contains no edge pixels from

the combined edge map, or (3) the global maximum leaf count is reached.

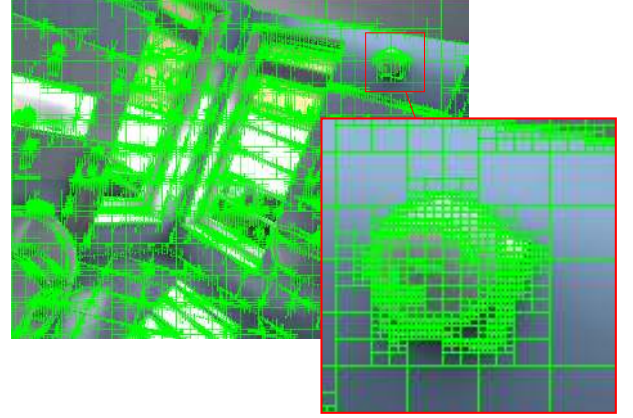


Fig. 3: Tree partitioning guided by the combined edge map. Complex regions are split into many small cells while homogeneous areas remain covered by larger, coarser cells. This adaptive layout concentrates Gaussian primitives where geometric and photometric detail is highest.

Two tree types are supported, each with distinct partitioning characteristics:

Quadtree mode: Each split divides a rectangle into four equal quadrants by halving both width and height. This produces uniform, rectangle-shaped cells suitable for isotropic refinement across the image.

KD-tree mode: Each split produces two child rectangles via an axis-aligned midpoint bisection. The split orientation is determined adaptively through two criteria:

- *Aspect-ratio constraint:* If the aspect ratio of the rectangle exceeds a configurable threshold (set to 1.5 in all experiments), the split is performed along the longest side to avoid highly elongated Gaussians. This value was found to balance between preventing extreme elongation and preserving sufficient freedom for the rectangle to adapt to local detail.
- *Edge distribution:* Otherwise, both candidate midpoint splits are evaluated by comparing the distribution of edge pixels between the resulting halves. The orientation that maximizes the absolute difference in edge-pixel count is selected, favoring splits that separate structurally distinct regions. Let $E(u, v) \in \{0, 1\}$ be the binary value of the edge map at pixel (u, v) . For a rectangle R divided into children R_1 and R_2 , the optimal orientation θ^* is:

$$\theta^* = \arg \max_{\theta} \left| \sum_{(u,v) \in R_1^\theta} E(u, v) - \sum_{(u,v) \in R_2^\theta} E(u, v) \right|$$

The split is always performed at the midpoint of the chosen axis, ensuring consistency in subdivision. When the width or height is odd, the central pixel column or row is assigned to the right (or lower) child; otherwise, the two children receive equal-sized halves, so that all pixels are covered exactly once.

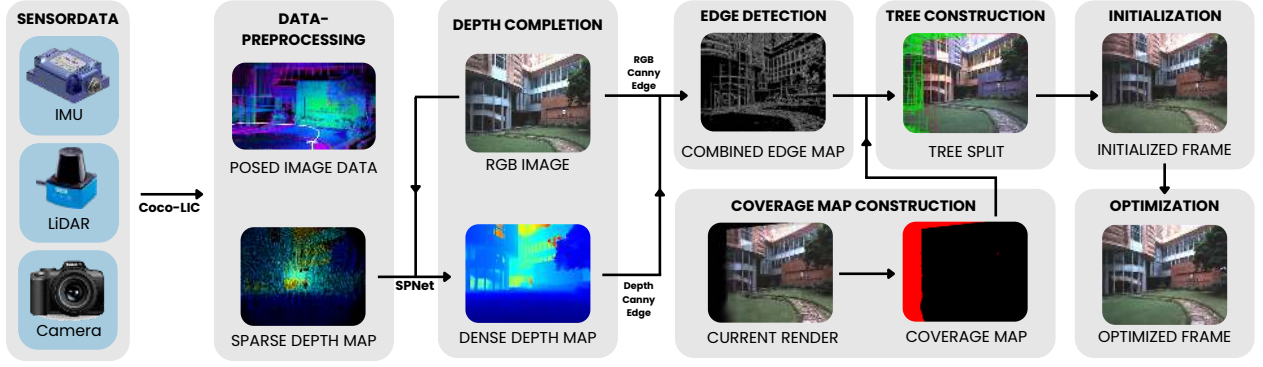


Fig. 2: Overview of the proposed real-time, dense initialization pipeline for 3DGS. Posed images and a sparse point cloud are obtained from raw RGB, LiDAR, and IMU data using Coco-LIC. Dense depth maps are then predicted via depth completion, after which photometric and geometric edges are detected and fused into a unified edge representation. This edge information guides the construction of a hierarchical tree that adaptively partitions the scene and supports coverage-aware refinement to avoid redundant Gaussians in already represented regions. In the leaves of this structure, Gaussians are initialized with position, color, and scale parameters, and finally refined through differentiable rendering during the optimization stage.

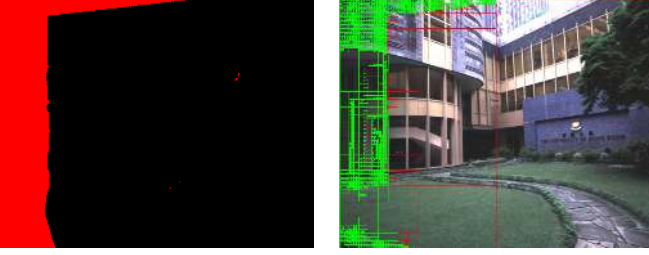


Fig. 4: Coverage-aware hierarchical tree construction. **Left:** Binary coverage mask computed with pixel-wise threshold $\tau = 0.7$, where pixels with accumulated opacity above this value are marked as covered (black). **Right:** Hierarchical tree overlay using the coverage mask $M(p)$: rectangles whose pixels are all covered (red borders) are early-stopped and receive no new Gaussians, while rectangles containing at least one uncovered pixel (green borders) are further subdivided, preventing redundant Gaussians and preserving the Gaussian budget.

D. Coverage-Aware Extension

For online scene reconstruction, a coverage-aware refinement mechanism prevents redundant Gaussian placement in already-represented regions. When processing a new keyframe, previously placed Gaussians visible in the current view are rendered to obtain a per-pixel accumulated opacity map $\mathcal{O}(p)$. A binary coverage mask $M(p)$ is generated by applying a pixel-wise threshold τ :

$$M(p) = \begin{cases} 1 & \text{if } \mathcal{O}(p) \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

where pixels with $M(p) = 1$ are considered covered and the remaining pixels are available for new Gaussian placement.

During hierarchical tree construction, the algorithm uses $M(p)$ to decide whether a rectangle still requires new Gaus-

sians. For each candidate rectangle, if all pixels within the rectangle satisfy $M(p) = 1$, the region is considered sufficiently covered and subdivision stops. Crucially, this rectangle is finalized as a leaf but **excluded from Gaussian initialization entirely**. Otherwise, the rectangle is subdivided further so the tree can isolate subregions that still require new Gaussians. This coverage-aware early stopping preserves the Gaussian budget for meaningful scene extensions and is illustrated in Figure 4.

E. Gaussian Initialization and Optimization

Once the hierarchical partitioning is complete, each leaf rectangle (except those excluded by coverage-aware early stopping) defines the spatial support for a single Gaussian primitive. For every candidate leaf, the center pixel is sampled, its depth retrieved from the completed depth map, and lifted to 3D world coordinates using camera intrinsics and extrinsics.

The parameters for each Gaussian are initialized as follows:

- **Position:** The computed world coordinate $\mathbf{p}_{\text{world}}$
- **Rotation:** Initially set to an identity quaternion $\mathbf{q} = (1, 0, 0, 0)$
- **Color:** The RGB value of the pixel at the leaf center (u, v) is converted to spherical harmonics, with only the DC term initialized to the sampled color to provide a uniform base appearance
- **Opacity:** Initialized using the *initial opacity* hyperparameter $\alpha_0 \in [0, 1]$ to allow for flexible refinement during optimization
- **Scale:** To initialize Gaussians with a depth extent that is consistent with their image-plane footprint, we set the screen-space scales from the leaf dimensions $(\Delta u, \Delta v)$, depth d , and camera intrinsics:

$$s_x = \frac{\Delta u}{2} \cdot \frac{d}{f_x}, \quad s_y = \frac{\Delta v}{2} \cdot \frac{d}{f_y}, \quad s_z = \frac{s_x + s_y}{2}.$$

with all scale parameters stored in log-space to ensure positivity during optimization. Defining s_z as the average of s_x and s_y yields an approximately isotropic initial 3D extent that avoids overly flattened or needle-like Gaussians.

By setting s_x and s_y to half the projected leaf size, we effectively place the rectangle boundaries at one standard deviation (1σ) from the mean. This implies that a large fraction of the Gaussian mass lies within the leaf, while the remaining mass smoothly decays beyond its boundaries. In practice, Gaussians are often evaluated only within a few standard deviations (e.g., up to 3σ), which yields effectively finite support while allowing overlap between neighboring primitives for smooth reconstruction.

After initialization, the Gaussian map is refined through the same gradient-based optimization used in Gaussian-LIC2 [2]. Rendered keyframes are compared with ground truth RGB images, and LiDAR depth provides geometric supervision. Optimization proceeds iteratively with Adam and CUDA-accelerated rendering, refining the initialized Gaussians into an accurate and geometrically consistent scene representation.

IV. RESULTS

A. Single Frame Runs

We first evaluate initialization and optimization on single frames of an r3live [32] sequence for both Gaussian-LIC2 and the proposed method (PM), reporting mean PSNR, SSIM, and LPIPS across different Gaussian budgets. Figure 5 shows mean PSNR as a function of wall-clock time, covering roughly the first 100 optimization iterations for each run.

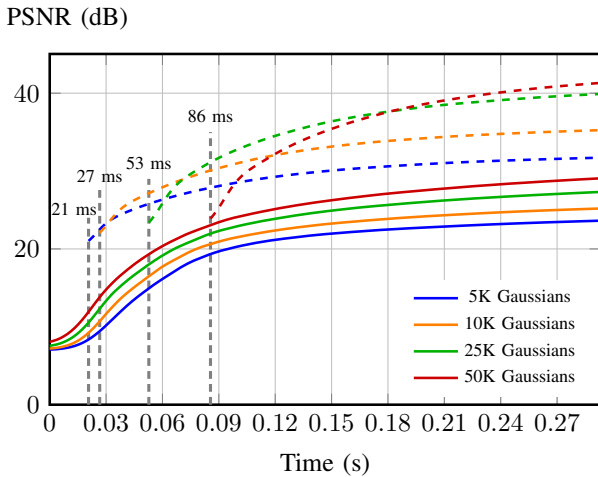


Fig. 5: Mean PSNR over optimization time for different numbers of Gaussians on the r3live sample [32]. Solid lines denote Gaussian-LIC2, dashed lines the PM. Vertical dashed lines and labels inside the plot indicate the start of PM optimization after tree construction.

Despite its additional initialization overhead, the PM starts from a higher PSNR than Gaussian-LIC2 has reached by the same time and continues to rise more steeply in the

early iterations. Asymptotically, the difference diminishes, indicating that both methods ultimately reach comparable reconstruction quality under the same Gaussian budget, while the PM converges substantially faster.

B. No optimization

In a second set of experiments, we evaluate different initialization strategies without any optimization to isolate the quality of the initial Gaussian placement. We compare three methods: the PM, the uniform sampling strategy from Gaussian-LIC2 [2], and the LoG-weighted sampling strategy from On-the-fly NVS [1]; their qualitative differences in coverage and detail preservation are illustrated in Fig. 1.

We evaluate on multiple r3live [32] sequences (*hku_campus*, *hku_park*, *hkust_campus*, and *degenerate_seq*), reporting PSNR, SSIM, and LPIPS directly after initialization. The results in Table I show that the PM consistently achieves the best metrics across all sequences while simultaneously using fewer Gaussians than both baselines.

Dataset	Metric	G-LIC2	O-t-f NVS	PM
hku_campus	Gaussians	672k	664k	525k
hku_campus	PSNR $\uparrow$	12.95	12.72	16.15
hku_campus	SSIM $\uparrow$	0.391	0.350	0.527
hku_campus	LPIPS $\downarrow$	0.610	0.532	0.526
hku_park	Gaussians	647k	633k	435k
hku_park	PSNR $\uparrow$	12.02	11.22	13.53
hku_park	SSIM $\uparrow$	0.335	0.301	0.372
hku_park	LPIPS $\downarrow$	0.589	0.570	0.532
hkust_campus	Gaussians	284k	262k	187k
hkust_campus	PSNR $\uparrow$	13.02	10.22	17.70
hkust_campus	SSIM $\uparrow$	0.424	0.242	0.565
hkust_campus	LPIPS $\downarrow$	0.640	0.608	0.418
degenerate_seq	Gaussians	312k	352k	228k
degenerate_seq	PSNR $\uparrow$	11.06	10.80	16.04
degenerate_seq	SSIM $\uparrow$	0.347	0.347	0.613
degenerate_seq	LPIPS $\downarrow$	0.529	0.458	0.292

TABLE I: Reconstruction quality without optimization on r3live sequences [32] for Gaussian-LIC2 [2], On-the-fly NVS [1], and the PM.

C. KD-tree vs. Quadtree Initialization

We compare quadtree and KD-tree based hierarchical initialization under the same single-frame r3live setting as in the first experiment. As shown in Fig. 6, we analyze the per-configuration PSNR difference (KD-tree minus Quadtree) over optimization iterations. These difference curves start slightly negative at iteration 0, reflecting a marginally better raw initialization for quadtree, but cross over after only a few iterations and remain positive thereafter, with the gap increasing as optimization progresses. This indicates that KD-tree consistently overtakes quadtree during training and makes more effective use of the same Gaussian budget.

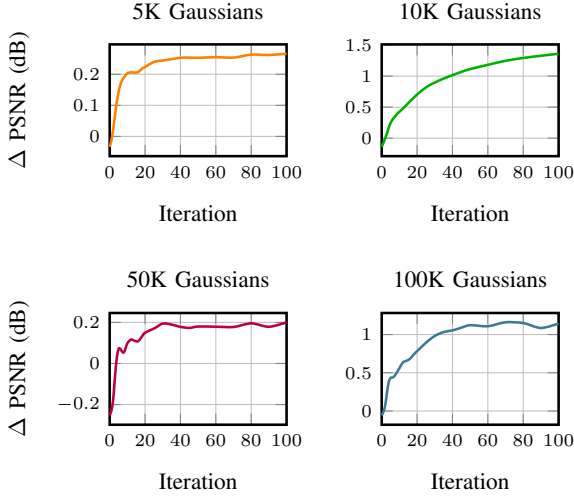


Fig. 6: Difference in mean PSNR (KD-tree minus quadtree) over optimization iterations for multiple Gaussian budgets. The curves start slightly negative at iteration 0, then cross zero after only a few iterations and remain positive, showing that KD-tree quickly overtakes quadtree and maintains a PSNR advantage as optimization progresses.

D. Hyperparameter Analysis

We perform a hyperparameter sweep, running over a thousand configurations across four r3live sequences and constructing Pareto fronts in novel-view PSNR versus total Gaussians for each parameter and sequence. We qualitatively inspect all these Pareto curves to identify consistent patterns that link specific settings to low- versus high-budget regimes, and illustrate one example in Figure 7, which shows the Pareto front for the initial opacity on the *hku_campus_seq*.

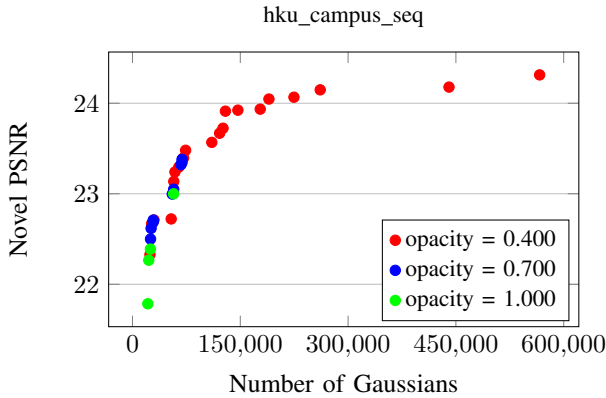


Fig. 7: Pareto-optimal configurations for the *hku_campus_seq* sequence, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the initial opacity, highlighting how different opacity values populate distinct regions of the quality–complexity trade-off.

For smaller total Gaussian counts, Pareto-optimal configurations tend to use higher initial opacities, lower pixel-wise coverage map thresholds τ , and higher Canny thresholds for the RGB edge maps. This makes Gaussians more opaque from the start, makes new Gaussian placement focus primarily on regions that are clearly not covered, limits the number of initialized primitives, and keeps the edge map relatively sparse, so that the limited capacity is concentrated on the strongest, most informative structures without over-subdivision.

At higher Gaussian budgets, the pattern reverses: Pareto-optimal configurations more often adopt lower initial opacity, higher pixel-wise coverage map thresholds τ , and lower RGB image edge map Canny ranges. In this regime, more transparent Gaussians and a higher leaf budget allow the model to place many finer primitives, while a less aggressive pixel-wise coverage map threshold and denser edge maps drive additional subdivision in already partially covered and detailed regions. Together, these settings allow the model to refine geometry and appearance more flexibly and ultimately achieve higher PSNR, at the expense of a substantially increased number of Gaussians.

E. Final Comparison to Gaussian-LIC2

In the final experiment, we compare the PM against Gaussian-LIC2 on full r3live sequences. Gaussian-LIC2 is evaluated both in its standard configuration and in variants where the initial Gaussians are downsampled to produce more constrained models that better match some budgets of our method. For every Gaussian-LIC2 configuration, we select a corresponding run of our method from the hyperparameter sweep, choosing a Pareto-optimal setting that always uses fewer total Gaussians than the matched Gaussian-LIC2 model. This setup ensures that any quality advantage of our method is achieved under a strictly tighter Gaussian budget.

Figure 8 plots the per-dataset PSNR difference as a function of the Gaussian-LIC2 Gaussian count. For most configurations and sequences, the curves remain close to zero, showing that our constrained models reach a final PSNR comparable to that of Gaussian-LIC2 even though they use fewer Gaussians. The slight advantage of Gaussian-LIC2 on *hkust_campus_seq* can be explained by its camera motion: the trajectory starts far from the scene and then repeatedly revisits the same content from much closer viewpoints, so our dense initialization already placed many Gaussians in these regions based on distant views and subsequently optimizes them, leaving less freedom to introduce new primitives when previously unresolved fine details become visible nearby. In contrast, on sequences such as *hku_park*, where the camera primarily rotates around roughly the same distance without strongly approaching regions that were already initialized, the initial dense allocation remains well matched to the observed content and the PM attains a clearer advantage over Gaussian-LIC2.

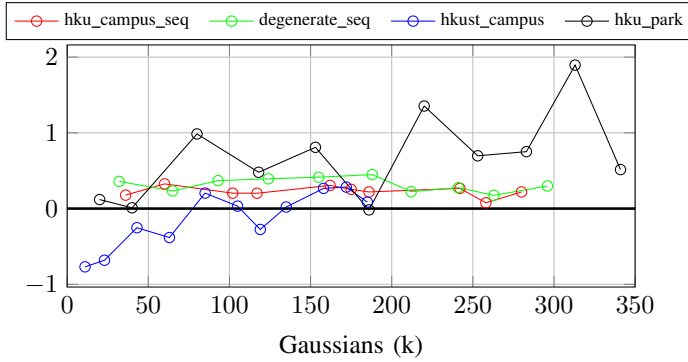


Fig. 8: Per-dataset PSNR differences between PM and Gaussian-LIC2

V. CONCLUSION

This work presented an online, real-time dense initialization pipeline for 3DGS that allocates Gaussians strategically, using more primitives in geometrically and photometrically complex regions and fewer in homogeneous areas. To this end, we developed an adaptive partitioning strategy over RGB-D images that fuses photometric and geometric edges, completes sparse LiDAR depth, and places Gaussians in a dense, structured manner aligned with scene geometry, while remaining compatible with the existing Gaussian-LIC2 optimization.

Experiments show that this structured initialization yields a higher-quality starting point and more efficient use of Gaussians, achieving comparable or improved reconstruction quality with fewer primitives and without sacrificing real-time performance. While incorporating LiDAR-derived depth edges provides additional robustness in low-texture or poorly constrained regions, the hierarchical initialization itself is agnostic to the depth source and can also be deployed in purely RGB-based pipelines, where it would inherit the usual limitations of monocular or SfM-based depth but still offer the same structured coverage and budget control.

Remaining challenges suggest several directions for future work, including more adaptive splitting criteria, richer guidance for identifying complex regions, mechanisms to selectively reinitialize or locally rebalance Gaussians when the camera later observes previously initialized regions from much closer viewpoints, and refined Gaussian placement within each leaf.

REFERENCES

- [1] A. Meuleman, I. Shah, A. Lanvin, B. Kerbl, and G. Drettakis, "On-the-fly reconstruction for large-scale novel view synthesis from unposed images," *ACM Transactions on Graphics*, vol. 44, no. 4, p. 1–14, 2025. [Online]. Available: <http://dx.doi.org/10.1145/3730913>
- [2] X. Lang, J. Lv, K. Tang, L. Li, J. Huang, L. Liu, Y. Liu, and X. Zuo, "Gaussian-lic2: Lidar-inertial-camera gaussian splatting slam," 2025. [Online]. Available: <https://arxiv.org/abs/2507.04004>
- [3] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 139:1–139:14, 2023.
- [4] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113.
- [5] J. Jung, J. Han, H. An, J. Kang, S. Park, and S. Kim, "Relaxing accurate initialization constraint for 3D gaussian splatting," *arXiv preprint arXiv:2403.09413*, 2024.
- [6] Z. Guo, J. Su, C. Qian, S. Wang, J. Fan, J. Zhang, W. Zhou, H. Amirpour, Y. Zhao, L. Han, and P. Wang, "Gp-gs: Gaussian processes densification for 3d gaussian splatting," 2026. [Online]. Available: <https://arxiv.org/abs/2502.02283>
- [7] C. Jiang, R. Gao, K. Shao, Y. Wang, R. Xiong, and Y. Zhang, "Li-gs: Gaussian splatting with lidar incorporated for accurate large-scale reconstruction," 2024. [Online]. Available: <https://arxiv.org/abs/2409.12899>
- [8] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [9] C. Zhao, S. Sun, R. Wang, Y. Guo, J.-J. Wan, Z. Huang, X. Huang, Y. V. Chen, and L. Ren, "TCLC-GS: Tightly coupled LiDAR-camera gaussian splatting for autonomous driving," *arXiv preprint arXiv:2404.02410*, 2024.
- [10] G. Hess, C. Lindström, M. Fatemi, C. Petersson, and L. Svensson, "SplatAD: Real-time LiDAR and camera rendering with 3D gaussian splatting for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [11] Y. Sun, Y. Zhou, B. Tian, H. Wang, Y. Zhao, and S.-T. Wu, "Gaussian-UDSR: Real-time unbounded dynamic scene reconstruction with 3D gaussian splatting," *Applied Sciences*, vol. 15, no. 11, p. 6262, 2025.
- [12] R. Song, T. Cai, M. Gross, Y. Zhang, W. Zimmer, Z. Huang, O. Wysocki, and J. Ma, "EnerGS: Energy-based gaussian splatting with partial geometric priors," *arXiv preprint, 2026*, semantic Scholar ID: 287874217.
- [13] X. Lang, C. Chen, K. Tang, Y. Ma, J. Lv, Y. Liu, and X. Zuo, "Coco-lic: Continuous-time tightly-coupled lidar-inertial-camera odometry using non-uniform b-spline," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, p. 7074–7081, Nov. 2023. [Online]. Available: <http://dx.doi.org/10.1109/LRA.2023.3315542>
- [14] J. Park, C. Park, M. Kim, M. Kim, and S. Kim, "LIVE-GS: Online LiDAR-inertial-visual state estimation and globally consistent mapping with 3D gaussian splatting," *arXiv preprint*, 2025, semantic Scholar ID: 280401025.
- [15] D. Li, Y. Liu, J. Liu, Z. Sun, Z. Huang, S. Sun, J. Liu, C. Yuan, H. Guo, F. E. Tay, and M. H. Ang, "RMGS-SLAM: Real-time multi-sensor gaussian splatting SLAM," *arXiv preprint*, 2026, semantic Scholar ID: 287473050.
- [16] R. Birkel, D. Wofk, and M. Müller, "Midas v3.1 – a model zoo for robust monocular relative depth estimation," 2023. [Online]. Available: <https://arxiv.org/abs/2307.14460>
- [17] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," 2024. [Online]. Available: <https://arxiv.org/abs/2406.09414>
- [18] C. Fu, C. Mertz, and J. Dolan, "Lidar and monocular camera fusion: On-road depth completion for autonomous driving," 10 2019, pp. 273–278.
- [19] H. Wang, M. Yang, X. Zheng, and G. Hua, "Scale propagation network for generalizable depth completion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 3, pp. 1908–1922, 2025.
- [20] D. Kotovenko, O. Grebenkova, and B. Ommer, "Edgs: Eliminating densification for efficient convergence of 3dgs," 2026. [Online]. Available: <https://arxiv.org/abs/2504.13204>
- [21] J. Edstedt, Q. Sun, G. Bökman, M. Wadenbäck, and M. Felsberg, "Roma: Robust dense feature matching," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2024, p. 19790–19800. [Online]. Available: <http://dx.doi.org/10.1109/CVPR52733.2024.01871>
- [22] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "DUST3R: Geometric 3D vision made easy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [23] X. Sun, R. Chen, M. Gong, D. Xu, and T. Liu, "Intern-GS: Vision model guided sparse-view 3D gaussian splatting," *arXiv preprint arXiv:2505.20729*, 2025.
- [24] N. Takama, S. Ito, K. Ito, H.-T. Chen, and T. Aoki, "Sparse2DGS: Sparse-view surface reconstruction using 2D gaussian splatting with dense point cloud," *arXiv preprint arXiv:2505.19854*, 2025, accepted at ICIP 2025.
- [25] J. Ha and J. Yi, "Fast and slim splat: Efficient 3D gaussian splatting via confidence and opacity-guided caching and sampling," in *Proceedings of*

the International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC), 2025.

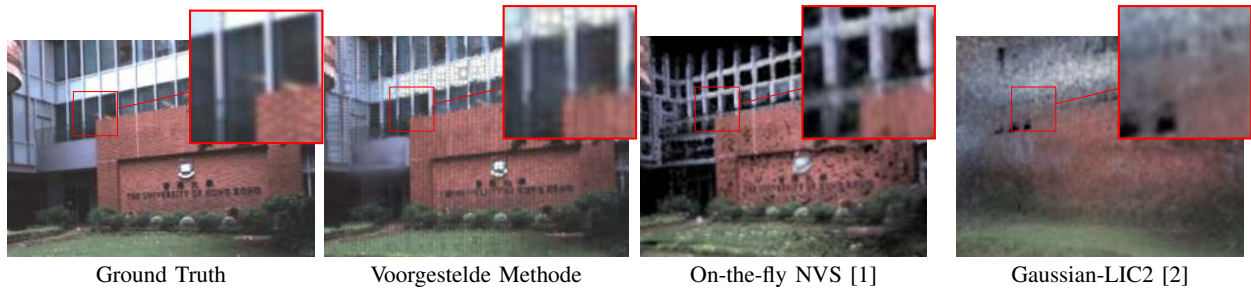
- [26] Y.-H. Li, T. Sikora, S. Knorr, and M. Sjöström, "Segmentation-driven initialization for sparse-view 3D gaussian splatting," *arXiv preprint arXiv:2509.11853*, 2025.
- [27] J. Artois, P. Lambert, and G. V. Wallendael, "MVS-Splatting: Fast multi-view stereo depth fusion for 3D gaussian splatting initialization," *IEEE Access*, 2025.
- [28] Y. Zheng, Z. Jiang, S. He, Y. Sun, J. Dong, H. Zhang, and Y. Du, "NexusGS: Sparse view synthesis with epipolar depth priors in 3D gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [29] P. Patt, L. Huang, Y. Zhang, and Y. Lei, "DensifyBeforehand: LiDAR-assisted content-aware densification for efficient and quality 3D gaussian splatting," *arXiv preprint arXiv:2511.19294*, 2025.
- [30] E. Giacomini, L. D. Giammarino, L. D. Rebotti, G. Grisetti, and M. R. Oswald, "Splat-loam: Gaussian splatting lidar odometry and mapping," 2025. [Online]. Available: <https://arxiv.org/abs/2503.17491>
- [31] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.
- [32] J. Lin and F. Zhang, "R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," 2021. [Online]. Available: <https://arxiv.org/abs/2109.07982>

Hiërarchische Dense Edge-Guided Initialisatie voor Real-Time 3D Gaussian Splatting

Timon Coucke

Promotoren: prof. dr. ir. Glenn van Wallendael, prof. dr. Peter Lambert

Begeleider: ir. Bert Ramlot



Figuur 1: Visuele vergelijking van de initialisatiekwaliteit op één frame zonder optimalisatie. **Links:** Ground-truth RGB-beeld. **Midden-links:** Onze voorgestelde dense, edge-guided initialisatie levert een heldere representatie met goed behoud van detail in complexe regio's en goede dekking in homogene gebieden. **Midden-rechts:** On-the-fly NVS [1], een 3DGS-reconstructiemethode die Gaussian-primitieven incrementeel genereert via directe, Laplacian-of-the-Gaussian (LoG)-gebaseerde gewogen sampling, legt gedetailleerde regio's effectief vast maar laat gaten in uniforme zones, wat resulteert in zichtbare zwarte gebieden. **Rechts:** Gaussian-LIC2 [2], een LiDAR-Inertial-Camera Gaussian Splatting SLAM-systeem, initialiseert Gaussians uniform op basis van LiDAR-diepte en aangevulde dieptemappen, wat resulteert in een grove en wazige dekking ondanks het grote aantal primitieven.

Samenvatting—3D Gaussian Splatting (3DGS) is een renderingstechniek die een scène voorstelt als een verzameling 3D Gaussians, geoptimaliseerd om overeen te komen met input-beelden. Deze thesis introduceert een dense initialisatiemethode voor 3DGS die real-time initialisatie en optimalisatie mogelijk maakt door gebruik te maken van LiDAR-enhanced RGB-D data. Voor elk beeld wordt een hiërarchische, edge-guided boompartitie geconstrueerd, waarbij zowel fotometrische als geometrische cues de plaatsing van Gaussians sturen. Elk blad genereert één Gaussian, waarvan positie en schaal worden afgeleid uit de ruimtelijke omvang van het blad, de gecompleteerde diepte en de camera-intrinsieken en -extrinsieken. Dit resulteert in een gestructureerde distributie die primitieven concentreert in complexe regio's en redundantie in homogene gebieden vermindert. Experimenten tonen een snellere vroege convergentie, concurrerende of verbeterde reconstructiekwaliteit bij lagere aantallen Gaussians, en een gunstigere kwaliteits-complexiteitsafweging ten opzichte van een state-of-the-art real-time LiDAR-gebaseerde baseline. Deze resultaten tonen aan dat verbeterde initialisatie zowel de efficiëntie als de prestaties van 3DGS-pijplijnen aanzienlijk kan verhogen. De implementatie van de voorgestelde methode is beschikbaar als open-sourcecode op <https://github.com/LGDTimou/gaussian-lic-research>.

I. INTRODUCTIE

3D Gaussian Splatting (3DGS) [3] representeert een scène als een verzameling optimaliseerbare 3D Gaussians,

geïnitieerd vanuit een schaarse puntenwolk verkregen via Structure from Motion (SfM) [4] of rechtstreeks uit LiDAR-data [2]. Een betere initialisatie garandeert niet noodzakelijk een hogere uiteindelijke reconstructiekwaliteit na volledige convergentie, maar beïnvloedt wel sterk de convergentiesnelheid en de kwaliteit onder beperkte Gaussian-budgetten. Wanneer de initialisatie zeer schaars is of bepaalde regio's volledig ontbreken, zijn deze gebieden bovendien moeilijk te herstellen tijdens de optimalisatie, wat de uiteindelijke reconstructiekwaliteit fundamenteel kan beperken [5], [6].

Puur RGB-gebaseerde pipelines produceren vaak onbetrouwbare puntenwolken in regio's met weinig textuur of complexe structuren, omdat feature matching daar geen nauwkeurige diepteschatting kan leveren. Het integreren van LiDAR-data lost dit probleem op door precieze metrische dieptemetingen te bieden, wat resulteert in consistentere en geometrisch accuratere priors dan louter geleerde diepteschatting. Zelfs met LiDAR-verrijkte pipelines steunen bestaande initialisatiestrategieën echter vaak op schaarse, feature-gebaseerde sampling of uniforme verdelingen die de scènecomplexiteit onvoldoende vatten, wat leidt tot inefficiënt gebruik van het Gaussian-budget en tragere convergentie. Recente dense initialisatiemethoden verbeteren de dekking, maar vereisen doorgaans

computationeel dure multi-view matching en benutten LiDAR-geometrische priors niet altijd, waardoor ze ongeschikt zijn voor real-time, online toepassingen waar views sequentieel binnenkomen.

Dit werk stelt een dense, edge-guided initialisatiestrategie voor op basis van een hiërarchische boompertitie van RGB-D-beelden. De partitie gebruikt zowel fotometrische als geometrische cues om elk beeld adaptief op te splitsen in regio's die de scènecomplexiteit weerspiegelen. Elk blad genereert één Gaussian, waarvan positie en schaal worden afgeleid uit de ruimtelijke omvang van de rechthoek horend bij het blad, in combinatie met de diepte-informatie en de camera-intrinsieken en -extrinsieken. Dit resulteert in een gestructureerde verdeling die primitieven concentreert in complexe regio's en redundantie in homogene gebieden vermindert. De methode ondersteunt real-time initialisatie en optimalisatie.

Experimentele resultaten tonen een snellere vroege convergentie, competitieve of verbeterde reconstructiekwaliteit bij lagere aantallen Gaussians, en een gunstigere kwaliteits-complexiteitsafweging ten opzichte van een state-of-the-art real-time LiDAR-gebaseerde baseline.

De belangrijkste bijdragen van dit werk zijn:

- Een hiërarchische boompertitiestrategie op basis van gefuseerde fotometrische (RGB) en geometrische (diepte) edge-detectie, die scènes adaptief opsplijst volgens hun complexiteit
- Een dekking-bewust extensiemechanisme dat redundante Gaussian-plaatsing in reeds gerepresenteerde regio's voorkomt tijdens incrementele mapping
- Een real-time, online dense initialisatiepipeline die LiDAR-diepteaanvulling, edge-guided partitie en gestructureerde Gaussian-plaatsing integreert zonder aanpassingen aan het onderliggende optimalisatieframework
- Experimentele validatie die aantoont dat de voorgestelde methode snellere convergentie en een betere kwaliteits-complexiteitsafweging bereikt dan Gaussian-LIC2 [2], in het bijzonder bij lage Gaussian-budgetten

II. GERELATEERD WERK

A. LiDAR-integratie met 3DGS

Standaard 3DGS [3] steunt op Structure-from-Motion (SfM) [4] om initiële puntenwolken te genereren. In de praktijk produceert SfM vaak onvolledige geometrie in textuurarme regio's, waar maar weinig features kunnen worden gematcht, en introduceert het geometrische ruis onder uitdagende lichtomstandigheden. LiDAR-gebaseerde benaderingen zoals LI-GS [7] pakken deze problemen aan door directe afstandsmetingen te voorzien, wat leidt tot hogere geometrische nauwkeurigheid en toelaat om met minder Gaussians dezelfde kwaliteit te behalen. Recente werken integreren LiDAR in 3DGS-gebaseerde pipelines voor uiteenlopende toepassingen: DrivingGaussian [8] reconstrueert dynamische omgevingsscènes voor autonoom rijden, TCLC-GS [9] bereikt real-time rendering tot 120 FPS, en SplatAD [10] maakt gezamenlijke

rendering van camera- en LiDAR-sensor mogelijk. Voor groot-schalige mapping reconstrueert Gaussian-UDSR [11] onbegrensde dynamische scènes, terwijl EnerGS [12] partiële LiDAR-observaties modelleert als continue energievelden. LiDAR-inertial-camera SLAM-systemen zoals Coco-LIC [13] leveren gesynchroniseerde poses en uitlijning in het wereldcoördinatenstelsel voor downstream reconstructie. Voortbouwend op deze basis integreren methoden als Gaussian-LIC2 [2], LIVE-GS [14] en RMGS-SLAM [15] LiDAR-, inertie en visuele metingen direct in de Gaussian maprepresentatie, wat real-time fotorealistische mapping in grootschalige buitenomgevingen mogelijk maakt.

B. Diepteaanvulling

Hoewel LiDAR nauwkeurige metingen biedt, vereist zijn schaarse karakter aanvulling om een volledige dekking van de scène te verkrijgen. Monoculaire methoden zoals MiDaS [16] en DepthAnything V2 [17] schatten dense diepte, maar lijden onder schaalambigüiteit. LiDAR-gebaseerde diepteaanvulling presteert beter dan monoculaire benaderingen door schaarse LiDAR-data te fuseren met RGB [18]. Gaussian-LIC2 maakt gebruik van SPNet [19] om LiDAR-blinde gebieden op te vullen.

C. Dense initialisatie

De meeste bestaande dense initialisatiestrategieën werken in een offline scenario, waarbij alle inputviews beschikbaar zijn vóór de optimalisatie. Ze verbeteren de convergentie door vanaf het begin een volledige Gaussian-dekking te genereren, waardoor iteratieve densificatie overbodig wordt [20]. EDGS [20] zoekt dense pixelcorrespondenties over meerdere views met behulp van matching-netwerken zoals RoMa [21], en trianguleert vervolgens deze gematchte pixels om in één stap rechtstreeks miljoenen 3D Gaussians te initialiseren. DUST3R-gebaseerde pipelines [22] genereren globaal consistente puntenwolken uit vaste beeldsets en is geïntegreerd in meerdere systemen [23], [24], [25]. SDI-GS [26] combineert MVS-priors met segmentatie-gebaseerde downsampling, terwijl MVS-Splatting [27] en NexusGS [28] dense diepteschatting uit multi-view stereo benutten. DensifyBeforehand [29] combineert schaarse LiDAR-data met monoculaire diepte voor content-bewuste sampling vóór de optimalisatie.

Omdat deze methoden vertrouwen op globale multi-view informatie, kunnen ze niet rechtstreeks worden toegepast in echt online scenario's waar frames sequentieel binnenkomen. Bovendien maakt hun computationeel dure matching ze moeilijk inzetbaar onder strikte real-time beperkingen.

D. Online initialisatie

Bestaande online initialisatiemethoden variëren in de mate waarin ze de Gaussian-plaatsing aan de scènecomplexiteit aanpassen. On-the-fly NVS [1] gebruikt Laplacian-of-Gaussian (LoG)-operatoren om fotometrische edges te detecteren en Gaussians te concentreren in getextureerde regio's, waardoor de overhead van densificatie wordt verminderd. Deze probabilistische samplingstrategie biedt echter geen garanties op

ruimtelijke dekking en gebruikt uitsluitend fotometrische complexiteit. Omgekeerd weegt Splat-LOAM [30] de initialisatie met behulp van afgeleiden van de dieptemap om zich te richten op geometrische discontinuïteiten, maar negeert fotometrisch detail. Gaussian-LIC2 [2] biedt sterke geometrische priors via LiDAR, maar verdeelt Gaussians uniform zonder zich aan te passen aan lokale scènecomplexiteit. Deze methoden werken in online pipelines waarin de Gaussian-representatie incrementeel wordt geüpdatet wanneer nieuwe frames binnenkomen, maar combineren nooit gelijktijdig fotometrische en geometrische cues voor detailbewuste, online initialisatie zoals in onze aanpak.

III. METHODOLOGIE

De voorgestelde real-time, online dense initialisatiepipeline voor 3DGS wordt geïllustreerd in Figuur 2. Gepositioneerde beelden en een schaarse puntenwolk worden verkregen uit ruwe RGB-, LiDAR- en IMU-data met Coco-LIC [13]. Dense dieptemappen worden vervolgens bekomen via diepteaanvulling (Sectie III-A), waarna fotometrische en geometrische edges worden gedetecteerd en samengevoegd tot een uniforme edge-representatie (Sectie III-B). Deze edge-informatie stuurt de constructie van een hiërarchische boom die de scène adaptief opdeelt (Sectie III-C) en ondersteunt dekking-bewuste uitbreiding om redundante Gaussians in reeds gerepresenteerde regio's te vermijden (Sectie III-D). In de bladeren van deze structuur worden Gaussians geïnitieerd met parameters voor positie, kleur, opaciteit, en schaal, die tijdens de optimalisatiefase (Sectie III-E) verder worden verfijnd via differentieerbare rendering.

A. Diepteaanvulling

Dense dieptemappen worden gegenereerd met SPNet [19], dat schaarse LiDAR-metingen fuseert met RGB-guidance om LiDAR-blinde regio's op te vullen en tegelijk de geometrische structuur te behouden. Deze diepteaanvulling levert een dense geometrische prior die zowel het edge-detectieproces als de daaropvolgende 3D Gaussian-initialisatie versterkt.

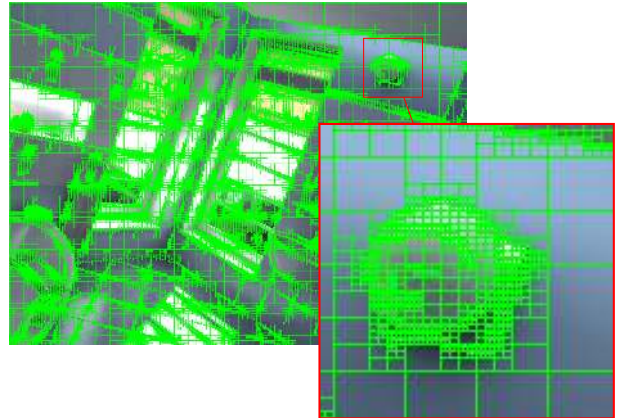
B. Edge-detectie

Zowel fotometrische als geometrische edges worden geëxtraheerd om de latere ruimtelijke partitionering te sturen. Fotometrische edges worden gedetecteerd in het RGB-beeld met een Canny-edge detector [31], die textuurgrenzen vastlegt. Geometrische edges worden geëxtraheerd uit de aangevulde dieptemap met dezelfde operator, waardoor structurele discontinuïteiten zoals objectsilhouetten en dieptesprongen worden gedetecteerd. De uiteindelijke edge-representatie wordt verkregen door beide kaarten pixelgewijs te combineren via een logische OR, $E = E_{rgb} \cup E_{depth}$, zodat regio's die één van beide soorten edge bevatten fijner worden opgesplitst. De Canny-drempels voor RGB en diepte worden onafhankelijk afgestemd om in elke modus een balans te vinden tussen grensbehoud en ruisonderdrukking.

C. Hiërarchische boompartitionering

De gecombineerde edge map stuurt de constructie van een hiërarchische boom die elk beeld adaptief opdeelt in regio's van verschillende grootte. De boom werkt met een vast blad-budget, waarbij elke uiteindelijke bladrechthoek één Gaussian definieert. Deze adaptieve layout concentreert Gaussians in complexe regio's, terwijl homogene gebieden worden afgedekt met minder, grotere primitieven, zoals te zien is in Figuur 3.

De boomconstructie gebeurt via een breadth-first frontier-expansie in plaats van depth-first subdivisie, om een evenwichtige verfijning over het hele beeld te garanderen zonder lokale oversubdivisie. Nodes die aan de splitscriteria voldoen, worden opgesplitst om kindrechthoeken voor het volgende niveau te produceren. Splitsen stopt wanneer de rechthoek van een node aan een van de volgende criteria voldoet: (1) een van de dimensies valt onder een minimale grootte, (2) er bevinden zich geen edge-pixels uit de gecombineerde edge map in de rechthoek, of (3) het globale maximale aantal bladeren is bereikt.



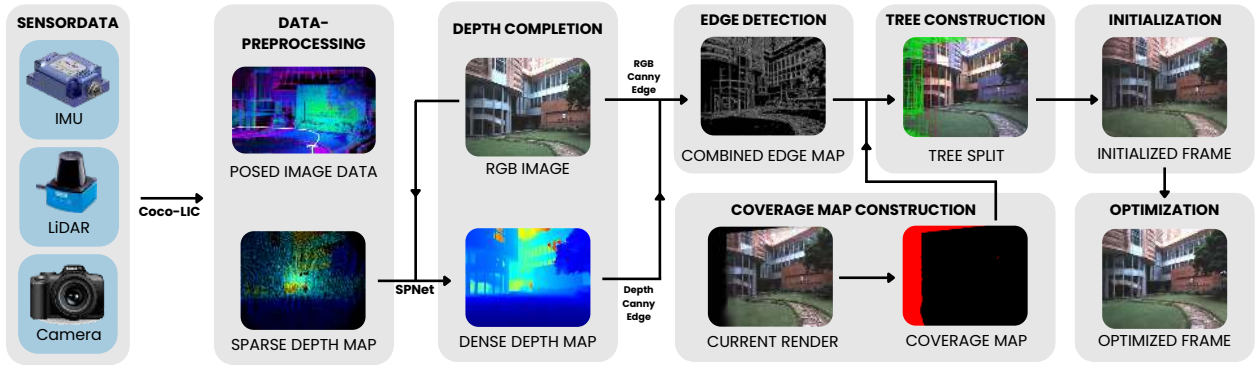
Figuur 3: Boompartitionering gestuurd door de gecombineerde edge map. Complexe regio's worden opgesplitst in veel kleine cellen, terwijl homogene gebieden worden afgedekt door grotere, grovere cellen. Deze adaptieve layout concentreert Gaussians in de regio's met de hoogste geometrische en fotometrische complexiteit.

Er worden twee boomtypes ondersteund, elk met specifieke partitioneringseigenschappen:

Quadtree-modus: Elke splitsing deelt een rechthoek in vier gelijke kwadranten door zowel de breedte als de hoogte te halveren. Dit levert uniforme, rechthoekige cellen op die geschikt zijn voor isotrope verfijning over het beeld.

KD-tree-modus: Elke splitsing produceert twee kindrechthoeken via een as-uitgelijnde halvering in het midden. De oriëntatie van de splitsing wordt adaptief bepaald op basis van twee criteria:

- **Aspect-ratio-beperking:** Als de aspect ratio van de rechthoek een configureerbare drempel overschrijdt (in alle experimenten ingesteld op 1,5), wordt de split uitgevoerd langs de langste zijde om sterk uitgerekte Gaussians te vermijden. Deze waarde bleek een goede balans te



Figuur 2: Overzicht van de voorgestelde real-time, dense initialisatiepipeline voor 3DGS. Gepositioneerde beelden en een schaarse puntenwolk worden verkregen uit ruwe RGB-, LiDAR- en IMU-data via Coco-LIC [13]. Densie dieptemappen worden vervolgens voorspeld via diepteaanvulling, waarna fotometrische en geometrische edges worden gedetecteerd en samengevoegd tot een uniforme edge-representatie. Deze edge-informatie stuurt de constructie van een hiërarchische boom die de scène adaptief opdeelt en ondersteunt dekking-bewuste uitbreiding om redundante Gaussians in reeds gerepresenteerde regio's te vermijden. In de bladeren van deze structuur worden Gaussians geïnitieerd met parameters voor positie, kleur, schaal en opaciteit, die vervolgens tijdens de optimalisatiefase verder worden verfijnd via differentieerbare rendering.

bieden tussen het voorkomen van extreme elongatie en het bewaren van voldoende vrijheid voor de rechthoek om zich aan lokaal detail aan te passen.

- *Edge-distributie:* In het andere geval worden beide kandidaat-splitsingen in het midden geëvalueerd door de verdeling van edge-pixels over de twee helften te vergelijken. De splitsingsrichting die het absolute verschil in het aantal edge-pixels maximaliseert, wordt gekozen, zodat de grenslijn bij voorkeur regio's met een verschillende structuur van elkaar scheidt. Laat $E(u, v) \in \{0, 1\}$ de binaire waarde van de edge map op pixel (u, v) zijn. Voor een rechthoek R die wordt opgesplitst in kinderen R_1 en R_2 , is de optimale oriëntatie θ^* :

$$\theta^* = \arg \max_{\theta} \left| \sum_{(u,v) \in R_1^\theta} E(u, v) - \sum_{(u,v) \in R_2^\theta} E(u, v) \right|.$$

De splitsing wordt altijd uitgevoerd in het midden van de gekozen as, zodat de onderverdeling consistent blijft. Wanneer de breedte of hoogte oneven is, wordt de centrale pixelkolom of -rij toegewezen aan het rechter (of onderste) kind; in het andere geval krijgen beide kinderen helften met gelijke grootte, zodat elke pixel precies één keer wordt toegewezen.

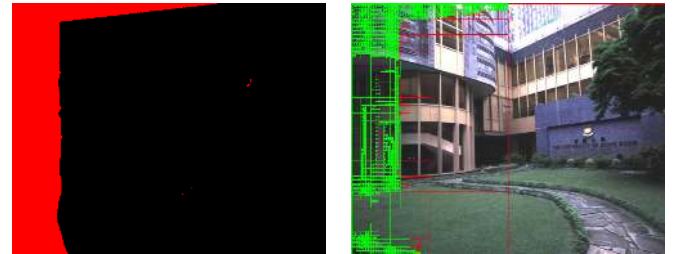
D. Dekkingbewuste extensie

Voor online scenereconstructie voorkomt een dekkingbewust uitbreidingsmechanisme redundante Gaussian-plaatsing in reeds gerepresenteerde regio's. Bij het verwerken van een nieuw keyframe worden eerder geplaatste Gaussians die in de huidige view zichtbaar zijn gerenderd om een per-pixel geaccumuleerde opaciteitskaart $\mathcal{O}(p)$ te verkrijgen. Een binair

dekkingsmasker $M(p)$ wordt vervolgens opgebouwd door een pixelgewijze drempel τ toe te passen:

$$M(p) = \begin{cases} 1 & \text{als } \mathcal{O}(p) \geq \tau \\ 0 & \text{anders.} \end{cases}$$

Pixels met $M(p) = 1$ worden daarbij als gedekt beschouwd, terwijl de overige pixels beschikbaar blijven voor nieuwe Gaussian-plaatsing.



Figuur 4: Dekkingsbewuste hiërarchische boomconstructie. **Links:** Binair dekkingsmasker berekend met pixelgewijze drempel $\tau = 0,7$, waarbij pixels met geaccumuleerde opaciteit boven deze waarde als gedekt worden gemarkeerd (zwart). **Rechts:** Hiërarchische boomoverlay op basis van het dekkingsmasker $M(p)$: rechthoeken waarvan alle pixels gedekt zijn (rode randen) worden vroegtijdig gestopt en krijgen geen nieuwe Gaussians, terwijl rechthoeken met minstens één ongedekte pixel (groene randen) verder worden opgesplitst. Dit vermijdt redundante Gaussians en spaart het Gaussian-budget.

Tijdens de hiërarchische boomconstructie gebruikt het algoritme $M(p)$ om te beslissen of een rechthoek nog nieuwe Gaussians nodig heeft. Voor elke kandidaat-rechthoek geldt: als alle pixels binnen de rechthoek aan $M(p) = 1$ voldoen, wordt de regio als voldoende gedekt beschouwd en stopt de

onderverdeling. Cruciaal is dat deze rechthoek wel als blad wordt afgerond, maar **nooit een Gaussian toegewezen krijgt**. In het andere geval wordt de rechthoek verder opgesplitst, zodat de boom subregio's kan isoleren die nog nieuwe Gaussians vereisen. Deze dekkingbewuste vroege stopping bewaart het Gaussian-budget voor zinvolle scène-extensies en wordt geïllustreerd in Figuur 4.

E. Gaussian-initialisatie en -optimalisatie

Zodra de hiërarchische partitionering voltooid is, definieert elke bladrechthoek (behalve de bladeren die door dekkingbewuste vroege stopping werden uitgesloten) de ruimtelijke support voor één Gaussian-primitief. Voor elk kandidaatblad wordt de centrale pixel gesampled, de bijbehorende diepte uit de aangevulde dieptemap opgehaald en omgezet naar 3D-wereldcoördinaten met behulp van de camera-intrinsieken en -extrinsieken.

De parameters voor elke Gaussian worden als volgt geïnitieerd:

- **Positie:** de berekende wereldcoördinaat $\mathbf{p}_{\text{world}}$
- **Rotatie:** initieel ingesteld op een identiteitquaternion $\mathbf{q} = (1, 0, 0, 0)$
- **Kleur:** de RGB-waarde van de pixel in het bladcentrum (u, v) wordt omgezet naar spherical harmonics (SH), waarbij enkel de DC-term wordt geïnitieerd met de gesampled kleur om een uniforme basisweergave te voorzien
- **Opaciteit:** geïnitieerd met de *initial opacity*-hyperparameter $\alpha_0 \in [0, 1]$ om flexibele verfijning tijdens de optimalisatie toe te laten
- **Schaal:** om Gaussians te initialiseren met een 3D-omvang die overeenkomt met hun grootte in het beeldvlak, worden de schalen in beeldruimte bepaald op basis van de bladafmetingen $(\Delta u, \Delta v)$, de diepte d en de camera-intrinsieken:

$$s_x = \frac{\Delta u}{2} \cdot \frac{d}{f_x}, \quad s_y = \frac{\Delta v}{2} \cdot \frac{d}{f_y}, \quad s_z = \frac{s_x + s_y}{2},$$

waarbij alle schaalparameters in log-space worden opgeslagen om positiviteit tijdens de optimalisatie te garanderen. Door s_z te definiëren als het gemiddelde van s_x en s_y ontstaat een ongeveer isotrope initiële 3D-omvang die te platte of naaldachtige Gaussians vermijdt.

Door s_x en s_y gelijk te stellen aan de helft van de geprojecteerde bladgrootte, plaatsen we de rechthoekgrenzen effectief op één standaarddeviatie (1σ) van het gemiddelde. Dit impliceert dat een groot deel van de Gaussian-massa binnen het blad ligt, terwijl de resterende massa soepel buiten de grenzen afneemt. In de praktijk worden Gaussians vaak slechts geëvalueerd binnen een paar standaarddeviaties (bijvoorbeeld tot 3σ), wat neerkomt op een effectief eindige support, terwijl overlap tussen naburige primitieven mogelijk blijft voor een vloeiende reconstructie.

Na de initialisatie wordt de Gaussian-map verfijnd met dezelfde gradientgebaseerde optimalisatie als in Gaussian-LIC2 [2]. Gerenderde keyframes worden vergeleken met

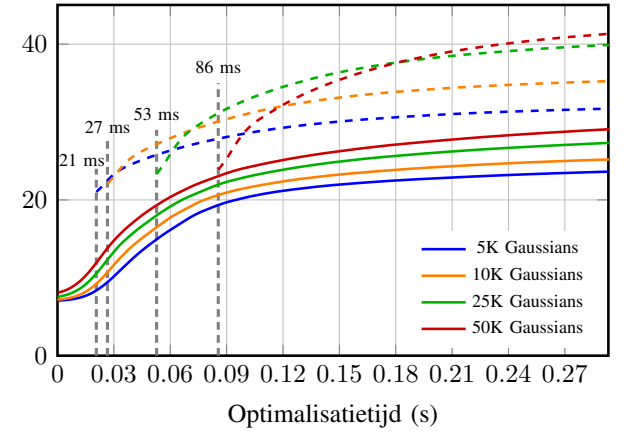
ground-truth RGB-beelden, en LiDAR-diepte levert geometrische supervisie. De optimalisatie verloopt iteratief met Adam en CUDA-versnelde rendering, waarbij de geïnitieerde Gaussians worden verfijnd tot een nauwkeurige en geometrisch consistente scenerepresentatie.

IV. RESULTATEN

A. Experimenten op afzonderlijke frames

We evalueren eerst initialisatie en optimalisatie op afzonderlijke frames van een r3live [32]-sequentie voor zowel Gaussian-LIC2 als de voorgestelde methode (VM), waarbij we de gemiddelde PSNR, SSIM en LPIPS rapporteren voor verschillende Gaussian-budgetten. Figuur 5 toont de gemiddelde PSNR als functie van de rekentijd en omvat ongeveer de eerste 100 optimalisatie-iteraties voor elke run.

PSNR (dB)



Figuur 5: Gemiddelde PSNR in functie van de optimalisatietijd voor verschillende aantallen Gaussians op de r3live-sample [32]. Doorgetrokken lijnen duiden Gaussian-LIC2 aan, stippellijnen de VM. Verticale stippellijnen en labels in de grafiek geven het begin van de VM-optimalisatie na boomconstructie aan.

Door de bijkomende initialisatiekosten start de VM later, maar vanaf het moment dat de optimalisatie start, ligt zijn PSNR al hoger dan de PSNR die Gaussian-LIC2 tegen dan heeft bereikt. Asymptotisch verdwijnt dit voordeel, zodat beide methoden uiteindelijk een vergelijkbare reconstructiekwiteit halen onder hetzelfde Gaussian-budget, maar de VM convergeert wel duidelijk sneller.

B. Geen optimalisatie

In een tweede reeks experimenten evalueren we verschillende initialisatiestrategieën zonder enige optimalisatie, om de kwaliteit van de initiële Gaussian-plaatsing te isoleren. We vergelijken drie methoden: de VM, de uniforme samplingstrategie uit Gaussian-LIC2 [2] en de LoG-gewogen samplingstrategie uit On-the-fly NVS [1]; hun kwalitatieve verschillen in dekking en detailbehoud worden geïllustreerd in Figuur 1.

We evalueren op meerdere r3live [32]-sequenties (*hku_campus*, *hku_park*, *hkust_campus* en *degenerate_seq*) en rapporteren PSNR, SSIM en LPIPS rechtstreeks na initialisatie. De resultaten in Tabel I tonen dat de VM consequent de beste metriekwaarden behaalt over alle sequenties, terwijl hij tegelijk minder Gaussians gebruikt dan beide baselines.

Dataset	Metriek	G-LIC2	O-t-f NVS	VM
hku_campus	Gaussians	672k	664k	525k
hku_campus	PSNR $\uparrow$	12.95	12.72	16.15
hku_campus	SSIM $\uparrow$	0.391	0.350	0.527
hku_campus	LPIPS $\downarrow$	0.610	0.532	0.526
hku_park	Gaussians	647k	633k	435k
hku_park	PSNR $\uparrow$	12.02	11.22	13.53
hku_park	SSIM $\uparrow$	0.335	0.301	0.372
hku_park	LPIPS $\downarrow$	0.589	0.570	0.532
hkust_campus	Gaussians	284k	262k	187k
hkust_campus	PSNR $\uparrow$	13.02	10.22	17.70
hkust_campus	SSIM $\uparrow$	0.424	0.242	0.565
hkust_campus	LPIPS $\downarrow$	0.640	0.608	0.418
degenerate_seq	Gaussians	312k	352k	228k
degenerate_seq	PSNR $\uparrow$	11.06	10.80	16.04
degenerate_seq	SSIM $\uparrow$	0.347	0.347	0.613
degenerate_seq	LPIPS $\downarrow$	0.529	0.458	0.292

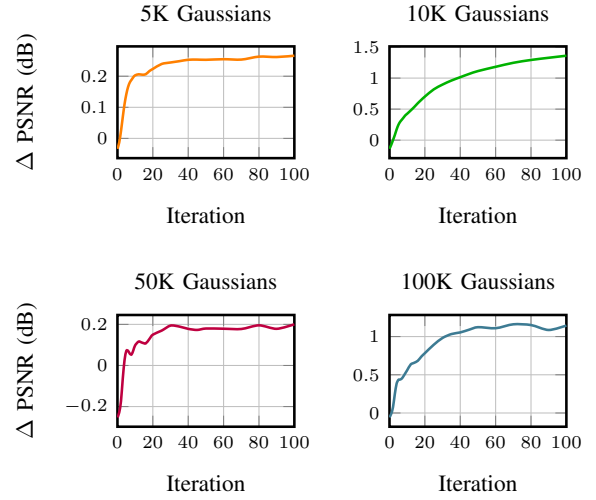
Tabel I: Reconstructiekwaliteit zonder optimalisatie op r3live-sequenties [32] voor Gaussian-LIC2 [2], On-the-fly NVS [1] en de VM.

C. KD-tree vs. Quadtree-initialisatie

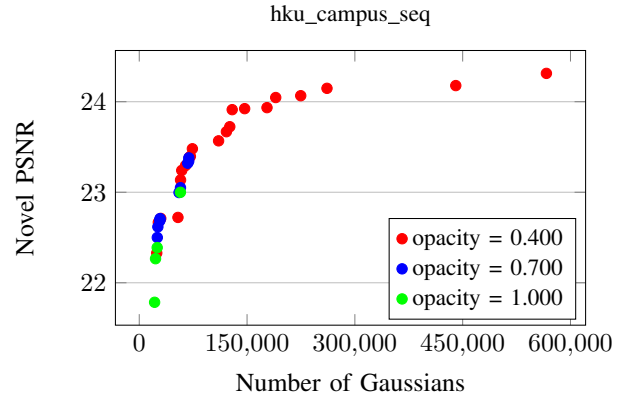
We vergelijken quadtree- en KD-tree-gebaseerde hiërarchische initialisatie onder dezelfde afzonderlijke frame r3live-instelling als in het eerste experiment. Zoals getoond in Figuur 6 analyseren we het per-configuratie-PSNR-verschil (KD-tree min quadtree) over de optimalisatie-iteraties. Deze verschilcurves starten licht negatief op iteratie 0, wat een iets betere ruwe initialisatie voor quadtree weerspiegelt, maar kruisen al na enkele iteraties en blijven daarna positief, waarbij de kloof groter wordt naarmate de optimalisatie vordert. Dit duidt erop dat de KD-tree tijdens training de quadtree consequent inhaalt en het beschikbare Gaussian-budget efficiënter benut.

D. Hyperparameteranalyse

We voeren een hyperparameter-sweep uit met meer dan duizend configuraties over vier r3live-sequenties en construeren Paretofronten in termen van novel-view-PSNR versus totaal aantal Gaussians voor elke parameter en sequentie. We inspecteren deze Pareto-curves kwalitatief om consistente patronen te identificeren die specifieke instellingen koppelen aan lage versus hoge Gaussian-budgetregimes, en illustreren één voorbeeld in Figuur 7, die het Paretofront toont voor de initiële opaciteit op de *hku_campus_seq*.



Figuur 6: Verschil in gemiddelde PSNR (KD-tree minus quadtree) over de optimalisatie-iteraties voor meerdere Gaussian-budgetten. De curves starten licht negatief op iteratie 0, kruisen daarna na slechts enkele iteraties de nul en blijven vervolgens positief, wat laat zien dat KD-tree quadtree snel inhaalt en zijn PSNR-voordeel behoudt naarmate de optimalisatie vordert.



Figuur 7: Pareto-optimale configuraties voor de *hku_campus_seq*-sequentie, met de novel-view-PSNR tegenover het totale aantal Gaussians. De punten zijn gekleurd op basis van de initiële opaciteit, wat benadrukt hoe verschillende opaciteitswaarden uiteenlopende regio's van de kwaliteits-complexiteitsverhouding invullen.

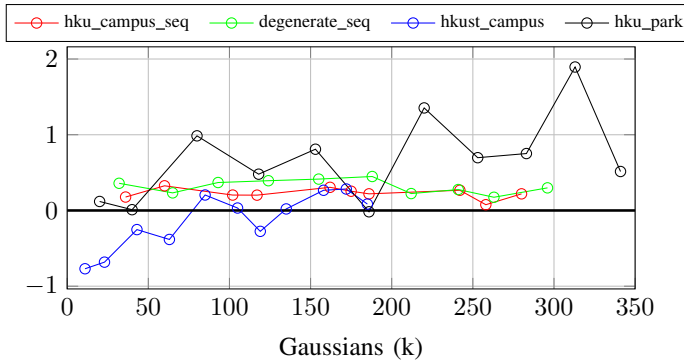
Voor kleinere totale aantallen Gaussians hebben Pareto-optimale configuraties de neiging om hogere initiële opaciteiten, lagere pixelgewijze drempels τ voor de dekkingsmap en hogere Canny-drempels voor de RGB-edgemappen te gebruiken. Dit maakt Gaussians vanaf het begin minder transparant, zorgt ervoor dat de plaatsing van nieuwe Gaussians vooral richt op regio's die duidelijk niet bedekt zijn, beperkt het aantal geïnitieerde primitieven en houdt de edgemap relatief schaars, zodat de beperkte capaciteit geconcentreerd wordt op de sterkste, meest informatieve structuren zonder

overmatige onderverdeling te veroorzaken.

Bij hogere Gaussian-budgetten keert dit patroon om: Pareto-optimale configuraties kiezen vaker voor een lagere initiële opaciteit, hogere pixelgewijze drempels voor de dekkingsmap τ en lagere Canny-drempels voor de RGB-edgemappen. In dit regime maken meer transparante Gaussians en een groter bladbudget het mogelijk om veel fijnere primitieven te plaatsen, terwijl een minder agressieve pixelgewijze dekkingsdrempel en dichtere edgemappen extra onderverdeling stimuleren in reeds gedeeltelijk bedekte en gedetailleerde regio's. Samen maken deze instellingen het mogelijk om geometrie en uiterlijk flexibeler te verfijnen en uiteindelijk een hogere PSNR te behalen, ten koste van een aanzienlijk groter aantal Gaussians.

E. Eindvergelijking met Gaussian-LIC2

In het laatste experiment vergelijken we de VM met Gaussian-LIC2 op volledige r3live-sequenties. Gaussian-LIC2 wordt zowel in zijn standaardconfiguratie geëvalueerd als in varianten waarbij de initiële Gaussians worden gedownsameld om beperktere modellen te verkrijgen die beter overeenkomen met bepaalde budgetten van onze methode. Voor elke Gaussian-LIC2-configuratie selecteren we een overeenkomstige run van onze methode uit de hyperparameter-sweep, waarbij we een Pareto-optimale instelling kiezen die steeds minder Gaussians gebruikt dan de bijbehorende Gaussian-LIC2-configuratie. Deze opzet zorgt ervoor dat elk kwaliteitsvoordeel van onze methode wordt behaald onder een strikt krasser Gaussian-budget.



Figuur 8: PSNR-verschillen per dataset tussen de VM en Gaussian-LIC2

Figuur 8 toont het PSNR-verschil per dataset als functie van het aantal Gaussians in Gaussian-LIC2. Voor de meeste configuraties en sequenties blijven de curves dicht bij nul, wat aantoont dat onze begrensde modellen een uiteindelijke PSNR bereiken die vergelijkbaar is met die van Gaussian-LIC2, ondanks het gebruik van minder Gaussians. Het lichte voordeel van Gaussian-LIC2 op *hkust_campus_seq* kan worden verklaard door de camerabeweging: het traject start ver van de scène en bezoekt vervolgens herhaaldelijk dezelfde regio vanuit een veel dichterbij gelegen standpunt, waardoor onze dichte initialisatie al veel Gaussians in deze regio's plaatst op basis van verre standpunten en deze vervolgens optimaliseert,

met minder speelruimte om nieuwe primitieven in te voeren wanneer voorheen ongeziene fijne details van dichtbij zichtbaar worden. Daarentegen, in sequenties zoals *hku_park*, waar de camera voornamelijk roteert rond ongeveer dezelfde afstand zonder duidelijk dichterbij reeds geïnitieerde regio's te komen, blijft de initiële dense allocatie goed afgestemd op de geobserveerde inhoud en behaalt de VM een duidelijk voordeel ten opzichte van Gaussian-LIC2.

V. CONCLUSIE

In dit werk hebben we een online, real-time dense initialisatiepipeline voor 3DGS voorgesteld die Gaussians strategisch toewijst, met meer primitieven in geometrisch en fotometrisch complexe regio's en minder in homogene gebieden. Daartoe ontwikkelden we een adaptieve partitioneringsstrategie op RGB-D-beelden die fotometrische en geometrische randen combineert, schaarse LiDAR-diepte aanvult en Gaussians op een dichte, gestructureerde manier plaatst in lijn met de scènegeometrie, terwijl de aanpak compatibel blijft met de bestaande Gaussian-LIC2-optimalisatie.

Onze experimenten tonen aan dat deze gestructureerde initialisatie een startpunt van hogere kwaliteit en een efficiënter gebruik van Gaussians oplevert, zodat een vergelijkbare of verbeterde reconstructiekwaliteit wordt bereikt met minder primitieven terwijl de real-time prestaties behouden blijven. Hoewel we in onze methode LiDAR-gebaseerde diepte-edges gebruiken om laag-textuur- en slecht geconditioneerde regio's beter aan te pakken, blijft de hiërarchische initialisatie in essentie onafhankelijk van de LiDAR data. Ze kan daarom ook worden toegepast in puur RGB-gebaseerde pipelines, waar ze de gebruikelijke beperkingen van monoculaire of SfM-gebaseerde diepte overneemt, maar nog steeds dezelfde gestructureerde dekking en controle over het Gaussian-budget biedt.

De resterende uitdagingen wijzen op verschillende pistes voor toekomstig werk, waaronder meer adaptieve splitscriteria, rijkere signalen om complexe regio's te identificeren, mechanismen om Gaussians selectief te herinitialiseren of lokaal te herverdelen wanneer de camera eerder geïnitieerde regio's later van veel dichterbij observeert, en verfijndere Gaussian-plaatsing binnen elk blad.

REFERENTIES

- [1] A. Meuleman, I. Shah, A. Lanvin, B. Kerbl, and G. Drettakis, "On-the-fly reconstruction for large-scale novel view synthesis from unposed images," *ACM Transactions on Graphics*, vol. 44, no. 4, p. 1–14, 2025. [Online]. Available: <http://dx.doi.org/10.1145/3730913>
- [2] X. Lang, J. Lv, K. Tang, L. Li, J. Huang, L. Liu, Y. Liu, and X. Zuo, "Gaussian-lic2: Lidar-inertial-camera gaussian splatting slam," 2025. [Online]. Available: <https://arxiv.org/abs/2507.04004>
- [3] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 139:1–139:14, 2023.
- [4] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113.
- [5] J. Jung, J. Han, H. An, J. Kang, S. Park, and S. Kim, "Relaxing accurate initialization constraint for 3D gaussian splatting," *arXiv preprint arXiv:2403.09413*, 2024.

- [6] Z. Guo, J. Su, C. Qian, S. Wang, J. Fan, J. Zhang, W. Zhou, H. Amirpour, Y. Zhao, L. Han, and P. Wang, "Gp-gs: Gaussian processes densification for 3d gaussian splatting," 2026. [Online]. Available: <https://arxiv.org/abs/2502.02283>
- [7] C. Jiang, R. Gao, K. Shao, Y. Wang, R. Xiong, and Y. Zhang, "Li-gs: Gaussian splatting with lidar incorporated for accurate large-scale reconstruction," 2024. [Online]. Available: <https://arxiv.org/abs/2409.12899>
- [8] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [9] C. Zhao, S. Sun, R. Wang, Y. Guo, J.-J. Wan, Z. Huang, X. Huang, Y. V. Chen, and L. Ren, "TCLC-GS: Tightly coupled LiDAR-camera gaussian splatting for autonomous driving," *arXiv preprint arXiv:2404.02410*, 2024.
- [10] G. Hess, C. Lindström, M. Fatemi, C. Petersson, and L. Svensson, "SplatAD: Real-time LiDAR and camera rendering with 3D gaussian splatting for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [11] Y. Sun, Y. Zhou, B. Tian, H. Wang, Y. Zhao, and S.-T. Wu, "Gaussian-UDSR: Real-time unbounded dynamic scene reconstruction with 3D gaussian splatting," *Applied Sciences*, vol. 15, no. 11, p. 6262, 2025.
- [12] R. Song, T. Cai, M. Gross, Y. Zhang, W. Zimmer, Z. Huang, O. Wysocki, and J. Ma, "EnerGS: Energy-based gaussian splatting with partial geometric priors," *arXiv preprint*, 2026, semantic Scholar ID: 287874217.
- [13] X. Lang, C. Chen, K. Tang, Y. Ma, J. Lv, Y. Liu, and X. Zuo, "Cocolic: Continuous-time tightly-coupled lidar-inertial-camera odometry using non-uniform b-spline," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, p. 7074–7081, Nov. 2023. [Online]. Available: <http://dx.doi.org/10.1109/LRA.2023.3315542>
- [14] J. Park, C. Park, M. Kim, M. Kim, and S. Kim, "LIVE-GS: Online LiDAR-inertial-visual state estimation and globally consistent mapping with 3D gaussian splatting," *arXiv preprint*, 2025, semantic Scholar ID: 280401025.
- [15] D. Li, Y. Liu, J. Liu, Z. Sun, Z. Huang, S. Sun, J. Liu, C. Yuan, H. Guo, F. E. Tay, and M. H. Ang, "RMGS-SLAM: Real-time multi-sensor gaussian splatting SLAM," *arXiv preprint*, 2026, semantic Scholar ID: 287473050.
- [16] R. Birkel, D. Wofk, and M. Müller, "Midas v3.1 – a model zoo for robust monocular relative depth estimation," 2023. [Online]. Available: <https://arxiv.org/abs/2307.14460>
- [17] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," 2024. [Online]. Available: <https://arxiv.org/abs/2406.09414>
- [18] C. Fu, C. Mertz, and J. Dolan, "Lidar and monocular camera fusion: On-road depth completion for autonomous driving," 10 2019, pp. 273–278.
- [19] H. Wang, M. Yang, X. Zheng, and G. Hua, "Scale propagation network for generalizable depth completion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 3, pp. 1908–1922, 2025.
- [20] D. Kotovenko, O. Grebenkova, and B. Ommer, "Edgs: Eliminating densification for efficient convergence of 3dgs," 2026. [Online]. Available: <https://arxiv.org/abs/2504.13204>
- [21] J. Edstedt, Q. Sun, G. Bökman, M. Wadenbäck, and M. Felsberg, "Roma: Robust dense feature matching," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2024, p. 19790–19800. [Online]. Available: <http://dx.doi.org/10.1109/CVPR52733.2024.01871>
- [22] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "DUST3R: Geometric 3D vision made easy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [23] X. Sun, R. Chen, M. Gong, D. Xu, and T. Liu, "Intern-GS: Vision model guided sparse-view 3D gaussian splatting," *arXiv preprint arXiv:2505.20729*, 2025.
- [24] N. Takama, S. Ito, K. Ito, H.-T. Chen, and T. Aoki, "Sparse2DGS: Sparse-view surface reconstruction using 2D gaussian splatting with dense point cloud," *arXiv preprint arXiv:2505.19854*, 2025, accepted at ICIP 2025.
- [25] J. Ha and J. Yi, "Fast and slim splat: Efficient 3D gaussian splatting via confidence and opacity-guided caching and sampling," in *Proceedings of the International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*, 2025.
- [26] Y.-H. Li, T. Sikora, S. Knorr, and M. Sjöström, "Segmentation-driven initialization for sparse-view 3D gaussian splatting," *arXiv preprint arXiv:2509.11853*, 2025.
- [27] J. Artois, P. Lambert, and G. V. Wallendaël, "MVS-Splatting: Fast multi-view stereo depth fusion for 3D gaussian splatting initialization," *IEEE Access*, 2025.
- [28] Y. Zheng, Z. Jiang, S. He, Y. Sun, J. Dong, H. Zhang, and Y. Du, "NexusGS: Sparse view synthesis with epipolar depth priors in 3D gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [29] P. Patt, L. Huang, Y. Zhang, and Y. Lei, "DensifyBeforehand: LiDAR-assisted content-aware densification for efficient and quality 3D gaussian splatting," *arXiv preprint arXiv:2511.19294*, 2025.
- [30] E. Giacomini, L. D. Giammarino, L. D. Rebotti, G. Grisetti, and M. R. Oswald, "Splat-loam: Gaussian splatting lidar odometry and mapping," 2025. [Online]. Available: <https://arxiv.org/abs/2503.17491>
- [31] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.
- [32] J. Lin and F. Zhang, "R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," 2021. [Online]. Available: <https://arxiv.org/abs/2109.07982>

Table of Contents

Acknowledgements	iv
Admission to loan	v
Thesis as part of an exam	vi
Abstract	vii
Table of Contents	xxv
List of Figures	xxvii
List of Tables	xxx
List of Acronyms	xxxi
1 Introduction	1
2 Related Work	2
2.1 LiDAR integration	2
2.2 Depth Completion	3
2.3 Initialization	3
2.3.1 Limitations of standard 3DGS initialization	3
2.3.2 Dense Initialization	3
2.3.3 Detail-Aware Initialization	4
3 Preliminary	6
3.1 3DGS	6
3.2 Gaussian-LIC2 refinement	7
4 Methodology	8
4.1 System Overview	8
4.2 Depth Completion	9
4.3 Photometric and Geometric Edge Detection	9
4.4 Hierarchical Tree Construction	11
4.5 Coverage Aware Gaussian Extension	12
4.5.1 Coverage Mask Generation	13
4.5.2 Coverage Aware Tree Refinement	13
4.6 Gaussian Initialization	14
4.7 Gaussian Optimization	16
5 Results	17
5.1 Single-frame test runs	17
5.2 Evaluation without optimization	18
5.3 Comparing KD-Tree to Quadtree	20
5.4 Hyperparameter analysis	22
5.4.1 Effect of the max amount of leaves per frame	23
5.4.2 Effect of initial opacity	23
5.4.3 Effect of opacity threshold τ	24
5.4.4 Effect of RGB image Canny edge threshold range	25
5.5 Comparison	26

6 Conclusion	29
Bibliography	30
A Algorithm	32
B Pareto-optimal runs	33
C Comparison Runs	36

List of Figures

2.1	Visual comparison of initialization quality on a single frame without optimization. Left: Ground truth RGB image. Center-left: Our proposed dense, edge-guided initialization produces a clear representation with well-preserved detail in complex regions and adequate coverage in homogeneous areas. Center-right: On-the-fly NVS [1] using direct, Laplacian of the Gaussian (LoG)-based weighted sampling of primitive positions and shapes, captures high-detail regions effectively but leaves gaps in uniform areas, resulting in visible black holes. Right: Gaussian-LIC2 [2] initializes Gaussians uniformly from LiDAR depth and depth-completed maps, resulting in coarse, blurry coverage despite using a significant amount of primitives.	5
3.1	Overview of the standard 3DGS pipeline. The process begins with SfM points and initializes 3D Gaussians, which are then projected and rendered via a differentiable tile-based rasterizer. During training, gradients backpropagate through the rasterizer to optimize Gaussian parameters and trigger adaptive density control to refine the scene representation. This diagram is from the original 3DGS paper [3]	6
4.1	Overview of the proposed real-time, dense initialization pipeline for 3DGS. Posed images and a sparse point cloud are obtained from raw RGB, LiDAR, and IMU data using Coco-LIC [4]. Dense depth maps are then predicted via depth completion (Section 4.2), after which photometric and geometric edges are detected and fused into a unified edge representation (Section 4.3). This edge information guides the construction of a hierarchical tree that adaptively partitions the scene (Section 4.4) and supports coverage-aware refinement to avoid redundant Gaussians in already represented regions (Section 4.5). In the leaves of this structure, Gaussians are initialized with position, color, and scale parameters (Section 4.6), and finally refined through differentiable rendering during the optimization stage (Section 4.7).	8
4.2	Output from the Coco-LIC SLAM framework [4]. The white line represents the estimated camera trajectory through the scene, while the multi-colored points illustrate the registered depth map and structural point cloud of the environment.	9
4.3	Depth completion using SPNet [5]. Given a sparse LiDAR depth input, SPNet synthesizes a dense depth map that fills LiDAR-blind regions and occlusions while preserving geometric structure and object boundaries consistent with the input RGB image. The resulting completed depth serves as an enhanced geometric prior for Gaussian initialization and hierarchical tree partitioning in the pipeline.	9
4.4	Effect of the range on the photometric edge map. A low threshold range (10, 30) yields a dense set of edges that also includes weak gradients and fine texture, while a high range (50, 150) retains mainly strong, high-contrast boundaries and suppresses low-contrast noise, providing more selective guidance for the subsequent hierarchical partitioning.	10
4.5	Example of photometric and geometric edge extraction. The RGB image edge map primarily captures color and texture boundaries, whereas the depth-based edge map recovers structural discontinuities such as object silhouettes and depth jumps that are weak or less pronounced in the RGB signal. The complementarity of these two cues motivates the use of a fused edge mask for the hierarchical tree.	11
4.6	Comparison of quadtree and KD-tree partitioning. Left: The quadtree structure, where each split produces four equal subcells. Right: The KD-tree structure, which splits rectangles into two axis-aligned halves. The magnified regions demonstrate how KD-tree splits adapt to edge structure and aspect ratio, often leading to more efficient refinement than the uniform quadtree layout.	12

4.7	Quadtree partitioning guided by the combined edge map. Structural boundaries in the image drive finer subdivision, so complex regions are split into many small cells while homogeneous areas remain covered by larger, coarser cells. This adaptive layout concentrates Gaussian primitives where geometric and photometric detail is highest. . . .	12
4.8	Coverage mask generation for a new keyframe. The rendered frame shows previously placed Gaussians that are visible in the current view, including a clear empty gap. The lower-threshold coverage mask leaves only clearly uncovered regions available for new Gaussian placement, while the higher-threshold mask treats even slightly occupied areas as uncovered, making the extension stage more restrictive and encouraging new Gaussians to align closely with the existing model.	13
4.9	Coverage-aware hierarchical tree construction. The binary coverage mask is computed, where pixels with accumulated opacity above threshold τ are marked as covered (black). In the tree refinement stage, rectangles in which all pixels are covered (red borders) are early stopped and receive no new Gaussians, while rectangles containing at least one uncovered pixel (green borders) are further subdivided. This prevents new Gaussians from being placed in already represented regions and preserves the Gaussian budget for meaningful extensions.	14
4.10	Gaussian initialization in image and 1D space. (Left) A 2D Gaussian blob centered in a rectangular leaf of size $\Delta u \times \Delta v$, where the standard deviations are $s_x = \Delta u/2$ and $s_y = \Delta v/2$, so that the leaf boundaries lie at approximately one standard deviation from the mean. This choice places a large portion of the Gaussian mass inside the leaf while leaving a decaying tail outside, which naturally creates both strong coverage of the cell, but also some bleeding into neighboring leaves. (Right) The marginal probability density function of a standard normal distribution along one axis, with vertical dashed lines at integer multiples of σ . The total area under the curve equals 1, and most of the Gaussian mass is concentrated within $\pm 3\sigma$, meaning that the bulk of the contribution is captured near the mean while outer standard deviations contribute progressively less, supporting smooth blending between adjacent primitives.	15
4.11	Hierarchical Gaussian initialization under varying budgets. The hierarchical tree partitioning progressively subdivides the image, with each cell initializing a Gaussian. At very low budgets (e.g., 100 or 1k), only a few large cells are formed, resulting in large Gaussians and a coarse, blurry approximation of the scene. As the budget increases, the tree refines into smaller cells, enabling more localized Gaussian placement. Around 5k Gaussians, the representation becomes structurally consistent, while higher budgets (e.g., 25k–100k) allow fine-grained partitioning that captures significantly more detail, approaching ground-truth appearance.	16
5.1	Mean PSNR over optimization time for different numbers of Gaussians on an r3live sequence [6]. Solid lines show the mean PSNR across all frames for Gaussian-LIC2, while vertical dashed lines and their labels mark the mean Peak Signal-to-Noise Ratio (PSNR) of the Proposed Method (PM) at the moment it starts optimization after tree construction.	17
5.2	Single-frame reconstruction quality at 5K Gaussians after 100 optimization iterations. Both methods use the same Gaussian budget, but Gaussian-LIC2 produces a noticeably blurrier image in certain regions (tiled ground, wall at the top left of the image), whereas the PM shows better quality, consistent with the higher PSNR reported in Table 5.1 and the faster convergence trend in Figure 5.1.	18
5.3	Qualitative comparison of reconstruction without optimization for different initialization strategies. From left to right and top to bottom: Gaussian-LIC2, On-the-fly Novel View Synthesis (NVS), and the proposed method.	19
5.4	Mean PSNR evolution over 100 optimization iterations comparing KD-Tree (dashed lines) and Quadtree (solid lines) initialization strategies across different Gaussian counts. While Quadtree achieves marginally higher initial PSNR values, KD-Tree consistently surpasses it after optimization for all tested configurations.	20
5.5	Difference in mean PSNR (KD-Tree minus Quadtree) over optimization iterations for each Gaussian count. Positive values indicate KD-Tree outperforms Quadtree. The shaded area emphasizes the performance gap, which grows larger with increased Gaussian counts. . . .	21

5.6	Visual comparison of KD-Tree and Quadtree reconstruction quality after 100 optimization iterations with 10K Gaussians. The red box in (a) indicates the zoomed region shown in (b) and (c). The KD-Tree produces sharper details and better preserves fine structures compared to Quadtree.	21
5.7	Scatter plots of hyperparameter sweep with Pareto frontiers for every scene from the r3live dataset [6]	22
5.8	Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the initial opacity, highlighting how different opacity values populate distinct regions of the quality–complexity trade-off. . . .	23
5.9	Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the initial opacity, highlighting how different opacity values populate distinct regions of the quality–complexity trade-off. . . .	24
5.10	Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the RGB image edge threshold range, highlighting how different canny thresholds populate distinct regions of the quality–complexity trade-off.	25
5.11	Effect of the RGB image Canny edge threshold range on a frame from the <i>hku-park</i> sequence. With the low range (10,30), almost all fine texture in the vegetation and pebble ground is detected as an edge, resulting in a heavily populated edge map with little emphasis on dominant structures. The higher range (50,150) filters out most of these weak responses and preserves a sparser set of strong contours, providing a more meaningful edge signal for guiding Gaussian placement and explaining the higher PSNR obtained for this range at comparable Gaussian budgets.	26
5.12	Per-dataset metric differences between the proposed method (PM) and Gaussian-LIC2 as a function of the Gaussian count of Gaussian-LIC2. Each point in the curves represents a matched pair of runs (one Gaussian-LIC2 run and one PM run) that share the same total number of Gaussians in the Gaussian-LIC2 configuration; the x-axis shows this Gaussian count (in thousands), and the y-axis shows the difference in the corresponding quality metric between the two methods. For PSNR and SSIM we plot $\text{PM} - \text{Gaussian-LIC2}$, while for LPIPS we plot $\text{Gaussian-LIC2} - \text{PM}$, so that in all three subplots positive values always indicate better performance of PM.	28

List of Tables

5.1	Initial and early-iteration mean PSNR comparison on single frames between Gaussian-LIC2 and the PM. Each cell lists initial–100-iteration values for the mean PSNR, followed by the absolute and relative improvements over the same interval. We focus on the first 100 iterations because they highlight the strong advantage of the PM at initialization and the slower convergence of Gaussian-LIC2; asymptotically, the PSNR gap narrows as both methods approach similar values, but Gaussian-LIC2 still converges to a slightly lower PSNR, indicating that the proposed method uses its limited Gaussians more effectively. .	18
5.2	Reconstruction quality without optimization on r3live sequences [6] for Gaussian-LIC2 [2], On-the-fly NVS [1], and the PM.	19
5.3	PSNR Comparison: Iteration 0, 50 and 100	20
B.1	Pareto-optimal runs for hku_campus.seq	33
B.2	Pareto-optimal runs for hkust_campus	34
B.3	Pareto-optimal runs for hku_park	34
B.4	Pareto-optimal runs for degenerate.seq	35
C.1	Comparison of Gaussian-LIC2 vs PM Runs - hku_campus.seq	36
C.2	Comparison of Gaussian-LIC2 vs PM Runs - degenerate.seq	36
C.3	Comparison of Gaussian-LIC2 vs PM Runs - hkust_campus	37
C.4	Comparison of Gaussian-LIC2 vs PM Runs - hku_park	37

List of Acronyms

3DGS	3D Gaussian Splatting
NeRF	Neural Radiance Fields
SfM	Structure-from-Motion
LiDAR	Light Detection and Ranging
LI-GS	LiDAR Incorporated Gaussian Splatting
SPNet	Structure Preserving Network for depth completion
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
LPIPS	Learned Perceptual Image Patch Similarity
NVS	Novel View Synthesis
LoG	Laplacian-of-Gaussian
LoD	Level of Detail
Splat-LOAM	Gaussian Splatting LiDAR Odometry and Mapping
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization And Mapping
DI	Dense Initialization
EDGS	Eliminating Densification for Efficient Convergence of 3DGS
GRay	Ray Tracing 3D Gaussians Near the Speed of Splats
MVS	Multi-View Stereo
PM	Proposed Method



Introduction

3DGS [3] is a recent technique that generates photorealistic 3D reconstructions and enables real-time rendering. The process starts from a set of 2D images, from which a sparse point cloud of the environment is first computed using Structure-from-Motion (SfM) [7]. These points serve as the basis for the initialization of millions of 3D Gaussians: deformable ellipsoids that represent the visual properties of the scene. Through an optimization algorithm, the position, shape, color, and transparency of these Gaussians are continuously adjusted until the model matches the original images.

Although 3DGS achieves impressive visual results using only camera images, the overall reconstruction quality remains highly dependent on the initial point cloud. Reconstructions based solely on RGB data often suffer in regions with little texture, reflective surfaces, or uniform colors, leading not only to geometric inaccuracies but also to degraded texture quality and visual artifacts.

Recent techniques aim to address the limitations of RGB-only reconstructions by incorporating LiDAR data. LiDAR sensors provide accurate depth information, resulting in a more consistent initial point cloud and a better representation of geometric structure. This removes the SfM reconstruction step, reducing preprocessing time and simplifying the overall pipeline. By combining LiDAR data with camera images, the reliability of the 3D initialization can be significantly improved, leading to more stable and realistic Gaussian optimizations.

Despite these improvements, the quality of the initialization remains a critical factor in 3D Gaussian Splatting. In practice, the initial placement and distribution of Gaussians are not always optimal, which leads to inefficiencies and slower or less accurate convergence during optimization [8, 9]. Ideally, the initialization step should already provide a high-resolution and geometrically strong representation of the scene at a limited computational cost. The lack of such a high-quality starting point therefore remains an important limitation in existing pipelines.

To address this issue, this thesis focuses on an alternative initialization strategy in which RGBD images are subdivided using a hierarchical tree structure based on specific criteria. Each leaf node of this structure corresponds to a region where a Gaussian is initialized. This structured partitioning allows for a more representative and higher-quality dense initial distribution with a lower number of total Gaussians, forming the basis for more efficient subsequent optimization.

Chapter 2 reviews prior work on LiDAR-augmented 3DGS, dense initialization, and other initialization strategies relevant to this thesis. Chapter 3 introduces the mathematical background needed to understand 3DGS and the additions of Gaussian LIC2 [2]. Chapter 4 describes the proposed system in detail and explains the underlying theoretical design choices. Chapter 5 presents the experiments performed and discusses their results to demonstrate the effectiveness of the proposed method. Finally, Chapter 6 concludes the thesis and summarizes the main findings.

Related Work

2.1 LiDAR integration

3DGS [3] represents scenes explicitly as collections of 3D Gaussians. Although it delivers impressive results, geometric accuracy in large-scale outdoor environments remains a challenge due to its dependence on SfM [7]. Literature points to three fundamental shortcomings of this visual approach [10]:

- **Geometric noise:** In the absence of sufficient texture, inaccurate points arise that are initialized as floating points, which forces an inaccurate optimization.
- **Incompleteness:** In the presence of reflections or difficult lighting, matching corresponding points fails, resulting in holes in the reconstruction.
- **Poor surface modeling:** It often fails to generate a coherent surface on flat regions, leading to irregular structures.

LiDAR data serves as a geometric basis that eliminates these uncertainties. By providing direct distance measurements, the geometric foundation is based on physical reality rather than visual estimates. As shown in LiDAR Incorporated Gaussian Splatting (LI-GS) [10], this integration translates into significant quantitative gains: the method reports an improvement in geometric accuracy of 68.7%, an increase of approximately 23.07% in F1-score, and a reduction of 45.22% in the number of required Gaussians compared with standard 3DGS.

Recent work integrate LiDAR into 3DGS-based pipelines across a range of applications. In autonomous driving, DrivingGaussian [11] uses LiDAR priors to reconstruct dynamic surrounding scenes with high panoramic consistency, while TCLC-GS [12] designs a tightly coupled LiDAR-camera representation that achieves real-time rendering at up to 120 FPS on urban datasets. SplatAD [13] extends this to joint camera and LiDAR sensor rendering, including sensor-specific effects such as rolling shutter and intensity dropouts. For large-scale mapping, Gaussian-UDSR [14] fuses LiDAR and SfM point clouds to reconstruct unbounded dynamic scenes in real time. EnerGS [15] models partial LiDAR observations as a continuous energy field, providing soft geometric guidance that avoids over-fitting when coverage is spatially uneven.

In parallel, LiDAR-inertial-camera Simultaneous Localization And Mapping (SLAM) systems such as Coco-LIC [4] provide temporally consistent trajectories and registered multi-modal measurements by tightly coupling LiDAR, Inertial Measurement Unit (IMU), and camera data. Such systems are relevant because they supply the synchronized poses and world-frame alignment needed for downstream LiDAR-aware 3D scene reconstruction.

More recent online systems such as Gaussian-LIC2 [2], LIVE-GS [16] and RMGS-SLAM [17] tightly integrate LiDAR, inertial, and visual measurements directly into the Gaussian map representation, enabling globally consistent photorealistic mapping in large-scale outdoor environments.

2.2 Depth Completion

Monocular depth completion, which relies solely on RGB images without depth input, has been advanced by models like MiDaS [18] and Depth Anything V2 [19]. These zero-shot methods estimate dense depth from visual cues such as texture gradients and object shapes but often suffer from scale ambiguity and lower geometric precision, especially in textureless regions or under varying lighting.

LiDAR-based depth completion models outperform pure monocular approaches by fusing accurate sparse LiDAR points with the corresponding RGB image, achieving superior structure preservation and boundary sharpness. This precision stems from LiDAR’s direct distance measurements, reducing errors by up to 30% in fused setups compared to RGB-only estimation [20].

Depth maps have also been used directly for 3DGS initialization. Unprojecting monocular or fused depth into a point cloud is a well-established alternative to SfM [21, 22, 2]. Multi-View Stereo (MVS)-Splatting [21] demonstrates that fusing multi-view stereo depth maps for initialization improves PSNR by 1.27 dB after only 1000 training iterations compared to the default pipeline. NexusGS [22] achieves dense point cloud coverage for sparse-view 3DGS by computing accurate depth maps from epipolar geometry, bypassing SfM entirely. Gaussian-LIC2 [2] leverages Structure Preserving Network for depth completion (SPNet) [5], which is a lightweight zero-shot depth completion model to fill in LiDAR-blind areas, enabling reliable 3DGS initialization.

2.3 Initialization

Although the integration of LiDAR provides a better geometric basis, the way in which the Gaussians are initially distributed across the scene is decisive for both convergence speed and final visual quality. Recent developments focus on improving this distribution by taking scene complexity into account.

2.3.1 Limitations of standard 3DGS initialization

In the original formulation of 3DGS, the scene is initialized with a point cloud derived from SfM [3]. The algorithm places Gaussians at the positions of these points and determines their initial size based on the average distance to the three nearest neighboring points, in order to create an initial spatial coverage of the scene.

Although this provides a useful starting point for reconstruction, the initialization in standard 3DGS is essentially a direct lift of the SfM point cloud: each SfM point becomes the center of a single Gaussian, and its initial size is set from local point spacing, without taking image content, depth priors, or the desired Gaussian density distribution into account [3].

Meuleman et al. point out in their work on *On-the-fly NVS* [1] that this mismatch can lead to two fundamental problems during optimization:

- **Increased computational overhead:** Because the initial distribution is often not well aligned with the desired Gaussian density, the algorithm has to rely heavily on iterative densification during training (splitting and cloning Gaussians). This significantly increases the required training time before a satisfactory level of detail is reached.
- **Visual artifacts and blurring:** In areas where the initial point cloud is too sparse for the reconstruction task, the optimizer may enlarge the remaining Gaussians to fill gaps in the representation. This can reduce local precision and lead to blurred fine details.

2.3.2 Dense Initialization

Recent work addresses these issues by replacing sparse SfM initialization with Dense Initialization (DI) strategies that use multi-view dense correspondences from matching networks like RoMa [23] to triangulate millions of small Gaussians covering all visible surfaces right from the start. The Eliminating Densification for Efficient Convergence of 3DGS (EDGS) method [24] initializes Gaussians using triangulated pixel correspondences across reference views and nearest neighbors, assigning initial parameters like position, color via spherical harmonics, and scale based on depth.

This approach bypasses iterative densification entirely, as the dense coverage provides early photometric supervision to all regions, leading to shorter optimization trajectories which reduces coordinate displacement by 50x and trajectory length by 30x compared to standard 3DGS.

Several other methods exploit dense geometric priors for initialization. DUST3R-based pipelines [25] have been integrated into multiple 3DGS systems [26, 27, 28] to generate dense, globally consistent point clouds directly from unposed images, bypassing SfM entirely. Segmentation-Driven Initialization (SDI-GS) [29] combines a dense MVS prior with segmentation-based downsampling to reduce Gaussian count by up to 50% while maintaining rendering quality. DensifyBeforehand [30] combines sparse LiDAR with monocular depth to build a dense, content-aware initialization point cloud, applying a region-of-interest sampling scheme to concentrate Gaussians on semantically important areas and bypass adaptive density control entirely.

Furthermore, DI synergizes with ray tracing-based 3DGS rendering. While DI slows rasterization due to linear scaling with Gaussian count, it accelerates ray tracing by reducing ray-Gaussian intersections thanks to smaller, more precise primitives, enabling logarithmic scaling. The Ray Tracing 3D Gaussians Near the Speed of Splats (GRay) method exploits this by combining EDGS’ DI with optimized ray tracing, achieving near 3DGS rendering speeds and optimization time, with good quality.

2.3.3 Detail-Aware Initialization

Recent methods attempt to adapt Gaussian initialization to scene complexity, but each exhibits limitations that motivate our approach.

Visual detail-based methods. On-the-Fly NVS [1] uses the **Laplacian-of-Gaussian (LoG)** operator to detect photometric edges and textured regions by measuring the second derivative of image intensity, then initializes additional Gaussians in areas with strong local detail. This achieves comparable image quality up to three times faster than standard 3DGS and reduces adaptive densification by approximately 40%. However, the probabilistic sampling strategy lacks explicit spatial control: it can leave holes in under-sampled regions and over-populate already dense zones. While LoG identifies where photometric detail exists, it cannot guarantee complete and well-balanced 3D coverage, limiting flexibility for applications requiring specific Level of Detail (LoD) settings.

Geometric detail-based methods. Splat-LOAM [31] uses weighted sampling initialization using the first derivative of the depth map to concentrate Gaussians on geometric discontinuities, but this approach has two fundamental shortcomings:

- **Incomplete focus on detail:** On-the-fly NVS captures only photometric complexity, while Splat-LOAM captures only geometric complexity. A textured flat plane is ignored by Splat-LOAM, while a uniformly colored edge is overlooked by On-the-fly NVS.
- **Lack of spatial control:** Both methods rely on probabilistic sampling without guarantees of consistent spatial coverage or explicit control over point density for different LoD requirements.

The current literature thus reveals a clear gap: no method simultaneously weighs photometric and geometric complexity while retaining direct control over point density and spatial coverage, especially when combined with LiDAR geometry and advanced dense initialization strategies like EDGS that eliminate densification. Figure 2.1 illustrates this by comparing the unoptimized initialization output of the uniform sampling (Gaussian-LIC2), photometric weighted sampling (On-the-fly NVS), and our proposed method.

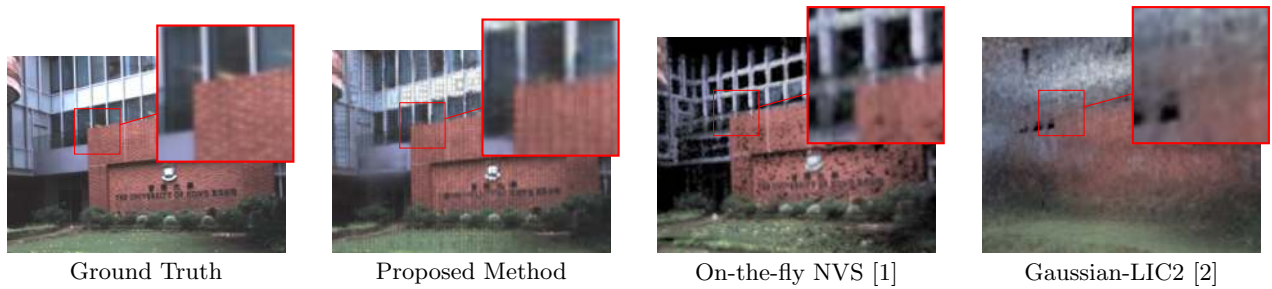


Figure 2.1: Visual comparison of initialization quality on a single frame without optimization. **Left:** Ground truth RGB image. **Center-left:** Our proposed dense, edge-guided initialization produces a clear representation with well-preserved detail in complex regions and adequate coverage in homogeneous areas. **Center-right:** On-the-fly NVS [1] using direct, Laplacian of the Gaussian (LoG)-based weighted sampling of primitive positions and shapes, captures high-detail regions effectively but leaves gaps in uniform areas, resulting in visible black holes. **Right:** Gaussian-LIC2 [2] initializes Gaussians uniformly from LiDAR depth and depth-completed maps, resulting in coarse, blurry coverage despite using a significant amount of primitives.

Preliminary

3.1 3DGS

3DGS [3] represents a scene as a set of anisotropic 3D Gaussians. Each Gaussian is defined by a position, scale, rotation, opacity, and appearance parameters encoded with spherical harmonics. In contrast to implicit radiance-field methods, the scene is stored explicitly as a collection of primitives that can be rendered efficiently.

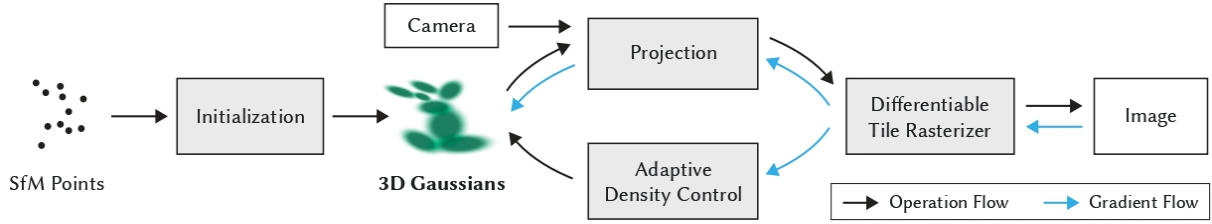


Figure 3.1: Overview of the standard 3DGS pipeline. The process begins with SfM points and initializes 3D Gaussians, which are then projected and rendered via a differentiable tile-based rasterizer. During training, gradients backpropagate through the rasterizer to optimize Gaussian parameters and trigger adaptive density control to refine the scene representation. This diagram is from the original 3DGS paper [3]

A 3D Gaussian is parameterized as $G_i = \{\mu_i, \Sigma_i, \alpha_i, c_i\}$, where $\mu_i \in \mathbb{R}^3$ denotes its center, $\Sigma_i \in \mathbb{R}^{3 \times 3}$ its covariance, α_i its opacity, and c_i its view-dependent color coefficients. To allow for anisotropic shapes that align with local scene geometry, the covariance is expressed as $\Sigma_i = R_i S_i S_i^\top R_i^\top$, where R_i is a rotation matrix and S_i is a diagonal scaling matrix.

For rendering, each Gaussian is projected into the image plane of a given camera. Around the projected center, the perspective transformation is locally approximated using a first-order linearization. This yields a 2D Gaussian $G_i^{2D} = \{\mu_i^{2D}, \Sigma_i^{2D}, \alpha_i, c_i\}$, where the covariance transforms as $\Sigma_i^{2D} = J_i \Sigma_i J_i^\top$, with J_i the Jacobian of the projection evaluated at μ_i .

The contribution of Gaussian i to a pixel p is then given by

$$g_i(p) = \alpha_i \exp\left(-\frac{1}{2}(p - \mu_i^{2D})^\top (\Sigma_i^{2D})^{-1} (p - \mu_i^{2D})\right),$$

which can be interpreted as a weighted 2D Gaussian density centered at μ_i^{2D} .

The final pixel color is obtained through front-to-back alpha compositing of all Gaussians sorted by depth:

$$C(p) = \sum_{i=1}^N T_i(p) g_i(p) c_i, \quad \text{where} \quad T_i(p) = \prod_{j=1}^{i-1} (1 - g_j(p))$$

accounts for the accumulated opacity of all preceding Gaussians.

This same compositing scheme can be used to render additional quantities such as expected depth and accumulated opacity. The depth map

$$D(p) = \sum_{i=1}^N T_i(p) g_i(p) d_i$$

corresponds to an opacity-weighted expected depth along the ray, rather than a hard surface depth. Similarly, the accumulated opacity is given by

$$O(p) = \sum_{i=1}^N T_i(p) g_i(p).$$

During training, the Gaussian parameters are optimized by comparing rendered images to ground truth observations. The loss combines an L_1 photometric term with a structural similarity term:

$$\mathcal{L}_{rgb} = (1 - \lambda) \|C - C^{gt}\|_1 + \lambda \mathcal{L}_{D-SSIM}(C, C^{gt}),$$

where both terms operate on RGB values, with SSIM encouraging perceptual and structural consistency. To better capture fine details, the model further employs adaptive densification, dynamically cloning or splitting Gaussians in regions with high gradients.

3.2 Gaussian-LIC2 refinement

Gaussian-LIC2 [2] builds on the same 3DGS formulation but adapts it to a LiDAR-inertial-camera setting. The main difference is that the scene initialization and optimization are guided by LiDAR and depth information, which improves geometric accuracy and makes the method better suited for outdoor incremental mapping.

Instead of initializing Gaussians only from an SfM point cloud, Gaussian-LIC2 uses LiDAR points together with completed depth estimates to create a denser and more reliable initialization. This reduces under-reconstruction in LiDAR blind regions and provides a stronger geometric prior for mapping.

The optimization stage is also extended with depth supervision. In addition to the RGB rendering loss, Gaussian-LIC2 adds a depth loss:

$$\mathcal{L}_{depth} = \|D - D^{gt}\|_1,$$

where D is the rendered depth and D^{gt} is the sparse LiDAR depth map. The full mapping loss becomes

$$\mathcal{L} = \mathcal{L}_{rgb} + \lambda_d \mathcal{L}_{depth},$$

with λ_d balancing appearance and geometry.

This refinement makes the map more geometrically consistent while preserving the visual quality of standard 3DGS. In practice, Gaussian-LIC2 therefore keeps the same core rendering mechanism as 3DGS, but improves initialization and training supervision by incorporating LiDAR-derived depth information. The optimization remains gradient-based and iterative, using the same basic Gaussian parameters and rendering formulation, but with stronger geometric constraints.

Methodology

4.1 System Overview

This chapter introduces the proposed method for initializing 3DGS based on combined RGB and LiDAR data. The pipeline consists of a series of sequential steps that together construct a structured and efficient representation of the scene. As a prerequisite, raw RGB, LiDAR, and IMU data are processed using the Coco-LIC SLAM framework [4] to obtain synchronized, posed RGB images and sparse LiDAR depth maps (see Figure 4.2). Next, the intrinsically sparse LiDAR information is densified through depth completion to obtain a dense depth map (see Section 4.2).

Based on this data, both photometric and geometric edges are detected and combined into a single consistent edge representation (see Section 4.3). This edge information guides the construction of a hierarchical tree structure that adaptively partitions the scene (see Section 4.4). For incremental extension, coverage-aware refinement prevents redundant Gaussians in already-represented regions (see Section 4.5). In the leaves of this structure, Gaussians are initialized with appropriate position, color, and scale parameters (see Section 4.6). Finally, an optimization phase is performed in which this initial representation is further refined via differentiable rendering (see Section 4.7).

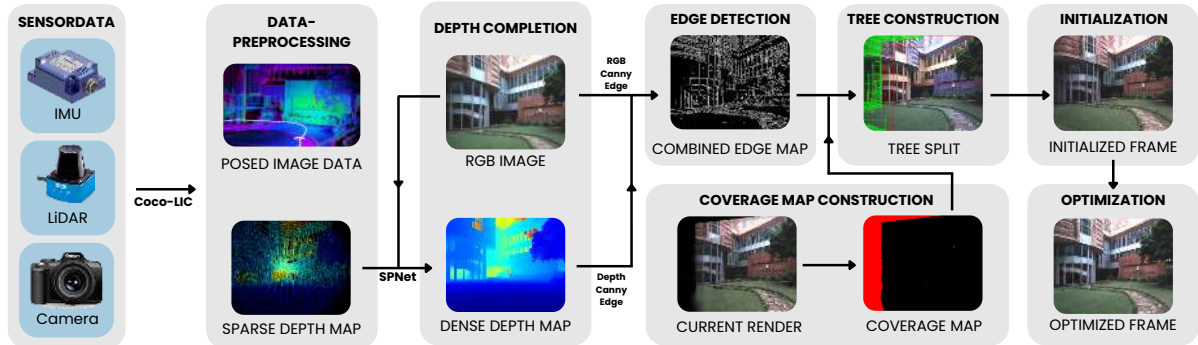


Figure 4.1: Overview of the proposed real-time, dense initialization pipeline for 3DGS. Posed images and a sparse point cloud are obtained from raw RGB, LiDAR, and IMU data using Coco-LIC [4]. Dense depth maps are then predicted via depth completion (Section 4.2), after which photometric and geometric edges are detected and fused into a unified edge representation (Section 4.3). This edge information guides the construction of a hierarchical tree that adaptively partitions the scene (Section 4.4) and supports coverage-aware refinement to avoid redundant Gaussians in already represented regions (Section 4.5). In the leaves of this structure, Gaussians are initialized with position, color, and scale parameters (Section 4.6), and finally refined through differentiable rendering during the optimization stage (Section 4.7).

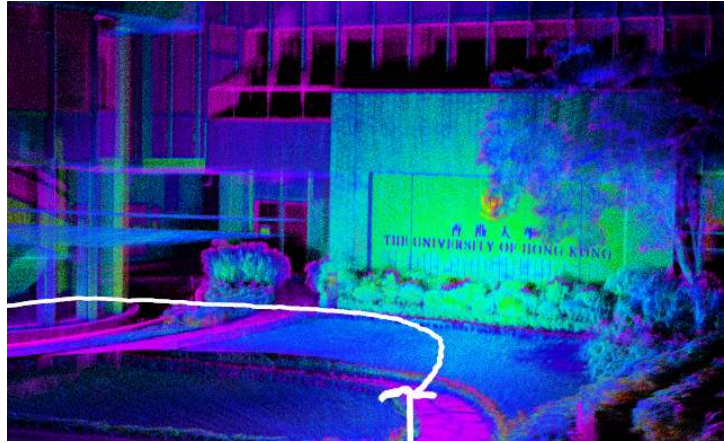


Figure 4.2: Output from the Coco-LIC SLAM framework [4]. The white line represents the estimated camera trajectory through the scene, while the multi-colored points illustrate the registered depth map and structural point cloud of the environment.

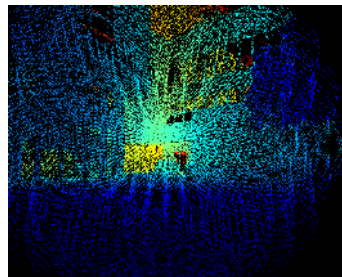
4.2 Depth Completion

In this stage, a dense depth map is generated to enhance the geometric representation used throughout the pipeline. The completed depth maps improve the geometric component of the hierarchical tree partitioning and support accurate lifting from 2D image space to 3D space during Gaussian initialization.

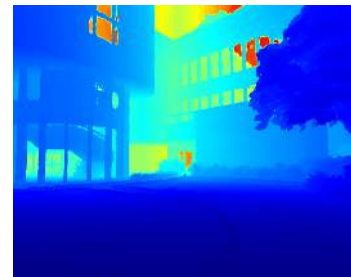
For each frame obtained from Coco-LIC, the initially sparse LiDAR-derived depth image is processed using the zero-shot depth completion network SPNet [5]. Given the corresponding RGB image, the incomplete depth map, and a validity mask indicating where LiDAR measurements are available, SPNet predicts a dense and geometrically consistent depth map that fills in missing regions while preserving structural details from both the image and the available depth data which is seen in figure 4.3.



RGB image



Sparse LiDAR depth



SPNet completed depth

Figure 4.3: Depth completion using SPNet [5]. Given a sparse LiDAR depth input, SPNet synthesizes a dense depth map that fills LiDAR-blind regions and occlusions while preserving geometric structure and object boundaries consistent with the input RGB image. The resulting completed depth serves as an enhanced geometric prior for Gaussian initialization and hierarchical tree partitioning in the pipeline.

4.3 Photometric and Geometric Edge Detection

To obtain a splitting criterion that captures both appearance-based and structure-based discontinuities, we extract edges from the RGB image as well as from the completed depth map. While photometric edges are capable of capturing geometric boundaries, they often lack the precision and reliability offered by LiDAR measurements. By integrating geometric edges, which highlight true surface discontinuities, with photometric data, the resulting edge map becomes significantly more reliable than using either modality alone.

For the photometric branch, the RGB image is first converted to an 8-bit representation and then transformed to grayscale. A Canny edge detector [32] is applied to the grayscale image to obtain thin and well-localized contours. Internally, the Canny operator performs Gaussian smoothing, gradient computation, non-maximum suppression, and hysteresis thresholding. The lower and upper thresholds determine how sensitive the detector is to weak and strong gradient responses, respectively. In practice, these thresholds are often chosen such that their *ratio* lies in the approximate range 1:2-1:3. This scaling balances sensitivity for fine edges against handling of noise.

Figure 4.4 illustrates how the choice of RGB Canny thresholds influences the photometric edge map. A low threshold range (10, 30) produces a dense edge map that includes many weak gradients and fine texture, whereas a high range (50, 150) uses a higher low value and a higher high value, yielding a sparser map that focuses on strong, high-contrast boundaries and suppresses low-contrast noise. This change in edge density directly affects which regions are likely to be subdivided more finely during the subsequent hierarchical tree construction.

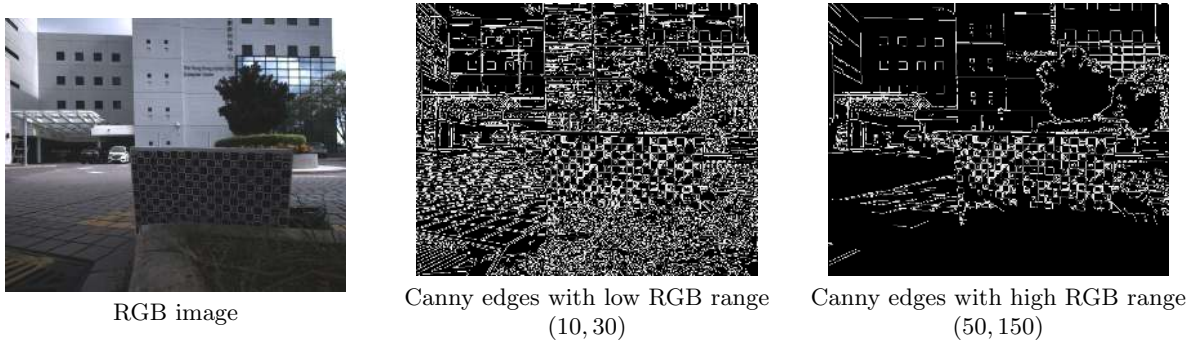


Figure 4.4: Effect of the range on the photometric edge map. A low threshold range (10, 30) yields a dense set of edges that also includes weak gradients and fine texture, while a high range (50, 150) retains mainly strong, high-contrast boundaries and suppresses low-contrast noise, providing more selective guidance for the subsequent hierarchical partitioning.

For the geometric branch, the completed depth map is normalized to the 8-bit range before edge extraction. This normalization makes the depth image compatible with the same Canny-based pipeline used for RGB, while preserving depth discontinuities after rescaling. The depth edge map therefore emphasizes object boundaries, depth jumps, and other structural transitions in the scene. The thresholds for the Canny detector were selected independently for RGB and depth, since both modalities exhibit different intensity statistics.

The final edge representation is obtained by combining the RGB and depth edge maps using a pixel-wise logical OR ($\vee$) operation. This merged map is used as the edge mask for the hierarchical tree construction, such that regions containing edges are subdivided more finely, while homogeneous regions remain represented by larger cells. In this way, the tree adapts its spatial resolution to the structure of the scene and concentrates detail around informative boundaries.

The combined edge map can be written as

$$E = E_{\text{rgb}} \vee E_{\text{depth}},$$

where E_{rgb} denotes the edge map extracted from the RGB image and E_{depth} denotes the edge map extracted from the depth map.

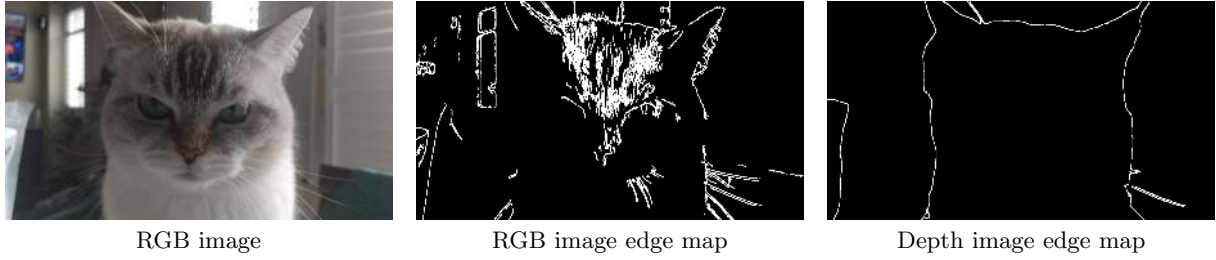


Figure 4.5: Example of photometric and geometric edge extraction. The RGB image edge map primarily captures color and texture boundaries, whereas the depth-based edge map recovers structural discontinuities such as object silhouettes and depth jumps that are weak or less pronounced in the RGB signal. The complementarity of these two cues motivates the use of a fused edge mask for the hierarchical tree.

4.4 Hierarchical Tree Construction

The combined edge map E guides the construction of a hierarchical 2D partition tree that adaptively subdivides the image into regions of varying size. The purpose of this partitioning is to define spatial supports for Gaussian primitives, with each leaf rectangle corresponding to a single primitive.

The subdivision follows the structural content of the image, capturing both photometric and geometric variations. Regions with high structural complexity, such as intensity changes, depth discontinuities, and fine details, are subdivided more extensively and represented by multiple small Gaussians. In contrast, regions that are homogeneous in both appearance and geometry are covered by fewer, larger primitives. This adaptive allocation results in a more efficient representation by concentrating modeling capacity in informative regions while avoiding unnecessary primitives elsewhere.

The tree construction operates as a breadth-first frontier expansion rather than depth-first subdivision, to ensure balanced refinement across the image without localized over-subdivision. At each level, the current set of nodes (rectangles) is processed sequentially. Nodes that meet the splitting criteria are divided to produce child rectangles for the next level. Splitting terminates when a node’s corresponding rectangle meets any of the following criteria: (1) one of its dimensions falls below a minimum size threshold, (2) it contains no edge pixels from the combined edge map, or (3) the global maximum leaf count is reached.

Two tree types are supported, each with distinct efficiency characteristics, as illustrated in figure 4.6:

- **Quadtree mode:** Each split divides a rectangle into four equal quadrants by halving both width and height. This produces uniform, square-shaped cells suitable for isotropic refinement.
- **KD-tree mode:** Each split produces two child rectangles via an axis-aligned midpoint bisection. The split orientation $\theta \in \{\text{horizontal, vertical}\}$ is determined adaptively:
 - *Aspect-ratio constraint:* If the aspect ratio of the rectangle exceeds a configurable threshold (set to 1.5 in all experiments), the split is performed along the longest side to avoid highly elongated Gaussians. This value was found to balance between preventing extreme elongation and preserving sufficient freedom for the rectangle to adapt to local detail.
 - *Edge distribution:* Otherwise, both candidate midpoint splits are evaluated by comparing the distribution of edge pixels between the resulting halves. The orientation that maximizes the absolute difference in edge-pixel counts is selected, favoring splits that separate structurally distinct regions. Let $E(u, v) \in \{0, 1\}$ be the binary value of the edge map at pixel (u, v) . For a rectangle R divided into children R_1^θ and R_2^θ , the optimal orientation θ^* is selected as:

$$\theta^* = \arg \max_{\theta} \left| \sum_{(u,v) \in R_1^\theta} E(u, v) - \sum_{(u,v) \in R_2^\theta} E(u, v) \right|.$$

The split is always performed at the midpoint of the chosen axis, ensuring consistency in subdivision. When the width or height is odd, the central pixel column or row is assigned to the right (or lower) child; otherwise, the two children receive equal-sized halves, so that all pixels are covered exactly once.

Algorithm 1 in Appendix A provides an overview of this hierarchical partition process in pseudocode. The final outcome of this process is an adaptive partition layout, as seen in Figure 4.7, which efficiently concentrates the modeling capacity in regions with the highest structural detail.

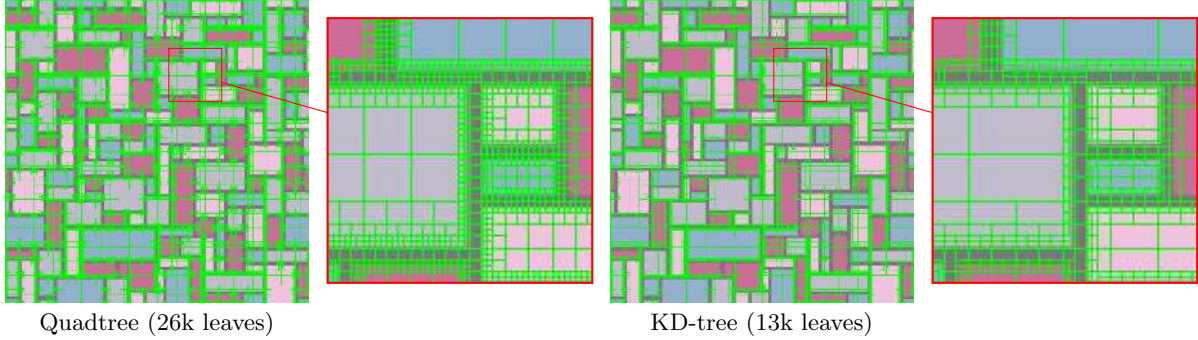


Figure 4.6: Comparison of quadtree and KD-tree partitioning. **Left:** The quadtree structure, where each split produces four equal subcells. **Right:** The KD-tree structure, which splits rectangles into two axis-aligned halves. The magnified regions demonstrate how KD-tree splits adapt to edge structure and aspect ratio, often leading to more efficient refinement than the uniform quadtree layout.

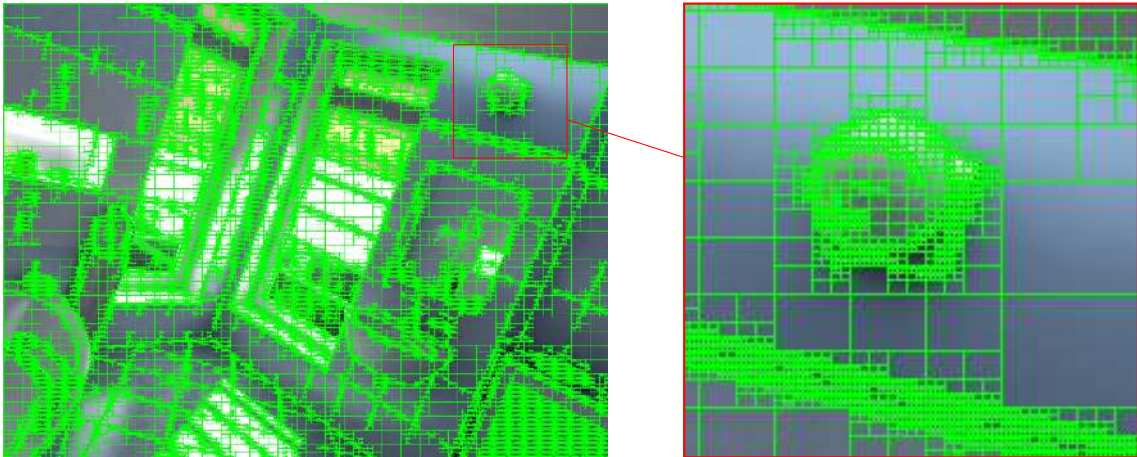


Figure 4.7: Quadtree partitioning guided by the combined edge map. Structural boundaries in the image drive finer subdivision, so complex regions are split into many small cells while homogeneous areas remain covered by larger, coarser cells. This adaptive layout concentrates Gaussian primitives where geometric and photometric detail is highest.

4.5 Coverage Aware Gaussian Extension

When extending the scene in an online setting, the goal is to avoid placing new Gaussians on top of regions that have already been represented by previously placed Gaussians. Instead, new primitives are introduced only in areas that are still insufficiently covered by the current model. Since these initial Gaussians will subsequently be fine-tuned during the optimization phase, placing redundant primitives in already covered areas wastes the available Gaussian budget without contributing meaningful new information.

4.5.1 Coverage Mask Generation

To implement this strategy, the pipeline must first determine which parts of the current view are already occupied. When encountering a new keyframe, the previously placed Gaussians that are visible in the new frame are rendered to obtain a per-pixel accumulated opacity map. Following the volumetric rendering formulation from Section 3.1, the accumulated opacity $O(p)$ for a pixel p indicates the density of the existing representation. To identify regions requiring further refinement, a binary coverage mask $M(p)$ is generated by applying a threshold τ :

$$M(p) = \begin{cases} 1 & \text{if } O(p) \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

Pixels with $M(p) = 1$ are considered covered, while the remaining pixels are treated as available for new Gaussian placement. A higher threshold τ makes it harder for a region to be classified as occupied, resulting in newly added Gaussians being more densely aligned with the existing model which can be seen in Figure 4.8.

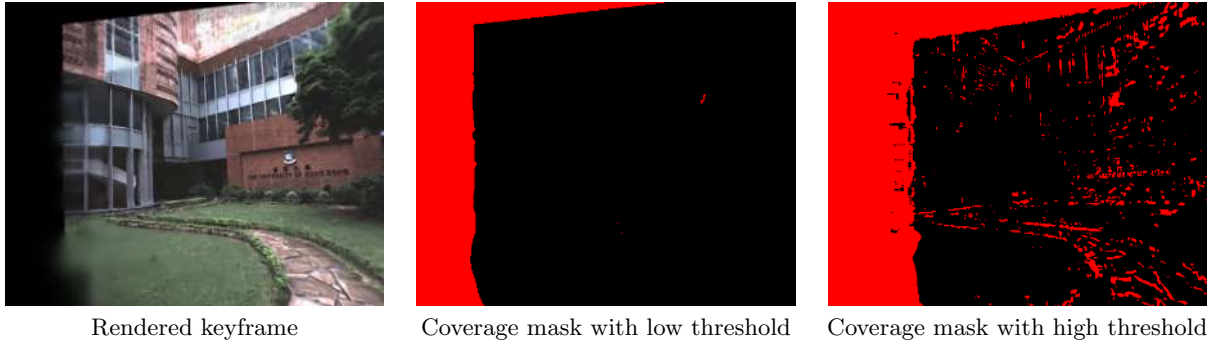


Figure 4.8: Coverage mask generation for a new keyframe. The rendered frame shows previously placed Gaussians that are visible in the current view, including a clear empty gap. The lower-threshold coverage mask leaves only clearly uncovered regions available for new Gaussian placement, while the higher-threshold mask treats even slightly occupied areas as uncovered, making the extension stage more restrictive and encouraging new Gaussians to align closely with the existing model.

4.5.2 Coverage Aware Tree Refinement

The second stage integrates this coverage information into the hierarchical tree construction. When the next frame is processed, the image plane is partitioned using the same tree-based decomposition as described in the previous section. However, before a region is further subdivided, the algorithm checks the binary coverage mask to determine whether all pixels inside the node’s corresponding rectangle are already covered. If every pixel in the region satisfies $M(p) = 1$, the region is considered sufficiently represented, subdivision stops, and the region is finalized as a leaf that will not receive any new Gaussians. Otherwise, if at least one pixel remains uncovered, the region is refined further so that the tree can isolate subregions that still require new Gaussians. Importantly, Gaussians are never sampled from these early-stopped, fully covered leaf rectangles.

This coverage-aware early stopping is particularly important when the total number of Gaussians that can be placed per frame is limited. Without it, the algorithm would continue splitting regions that are already fully covered into excessive leaves, consuming the available Gaussian budget and preventing more thorough refinement in detailed, insufficiently covered areas that need additional primitives.

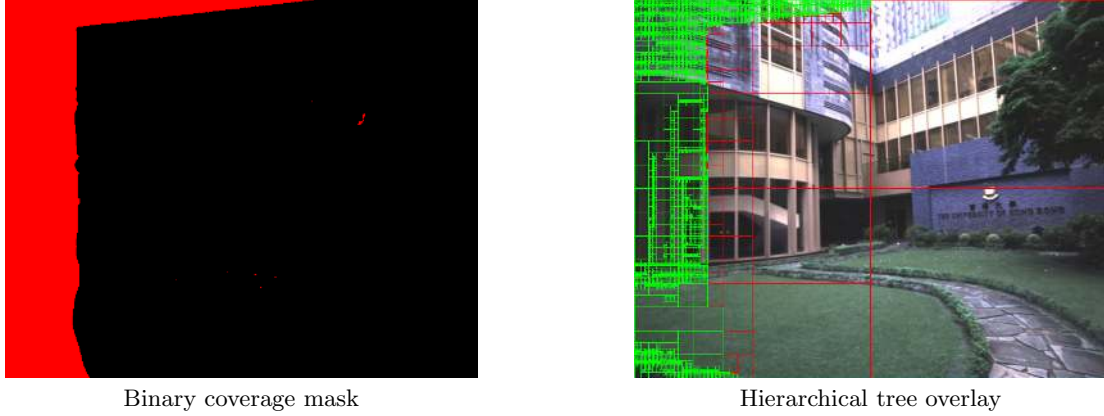


Figure 4.9: Coverage-aware hierarchical tree construction. The binary coverage mask is computed, where pixels with accumulated opacity above threshold τ are marked as covered (black). In the tree refinement stage, rectangles in which all pixels are covered (red borders) are early stopped and receive no new Gaussians, while rectangles containing at least one uncovered pixel (green borders) are further subdivided. This prevents new Gaussians from being placed in already represented regions and preserves the Gaussian budget for meaningful extensions.

4.6 Gaussian Initialization

Once the hierarchical partitioning is complete, the scene is represented by a collection of leaf. Each leaf’s rectangle in the constructed tree, except those filtered out by coverage-based early stopping, acts as a spatial support for one Gaussian, which is initialized using the available photometric and geometric data. This initialization step bridges the gap between the discrete 2D partition and the continuous 3D scene representation required for 3DGS.

For each candidate leaf, we sample a representative pixel at its center with coordinates (u, v) . The depth d at this location is retrieved from the completed depth map. This 2D point is then lifted to a 3D point in the camera coordinate system:

$$\mathbf{p}_{\text{cam}} = \left(\frac{(u - c_x)d}{f_x}, \frac{(v - c_y)d}{f_y}, d \right)^\top,$$

where (f_x, f_y) are the focal lengths and (c_x, c_y) the principal point of the camera. This point is transformed to the world coordinate system using the pose of the keyframe:

$$\mathbf{p}_{\text{world}} = \mathbf{R}_{\text{wc}}\mathbf{p}_{\text{cam}} + \mathbf{t}_{\text{wc}},$$

where $\mathbf{R}_{\text{wc}}$ and $\mathbf{t}_{\text{wc}}$ denote the camera-to-world rotation and translation, respectively.

The parameters for each Gaussian are initialized as follows:

- **Position:** The computed world coordinate $\mathbf{p}_{\text{world}}$.
- **Rotation:** Initially set to an identity quaternion $\mathbf{q} = (1, 0, 0, 0)$.
- **Color:** The RGB value of the pixel at the leaf center (u, v) is converted to spherical harmonics, with only the DC term initialized to the sampled color to provide a uniform base appearance.
- **Opacity:** Initialized using the *initial opacity* hyperparameter $\alpha_0 \in [0, 1]$ to allow for flexible refinement during optimization.
- **Scale:** To initialize Gaussians with a depth extent that is consistent with their image-plane footprint, we set the screen-space scales from the leaf dimensions $(\Delta u, \Delta v)$, depth d , and camera intrinsics:

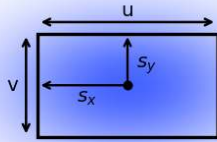
$$s_x = \frac{\Delta u}{2} \cdot \frac{d}{f_x}, \quad s_y = \frac{\Delta v}{2} \cdot \frac{d}{f_y}, \quad s_z = \frac{s_x + s_y}{2}.$$

All scale parameters are stored in log-space to ensure positivity during optimization. Defining s_z as the average of s_x and s_y yields an approximately isotropic initial 3D extent that avoids overly flattened or needle-like Gaussians.

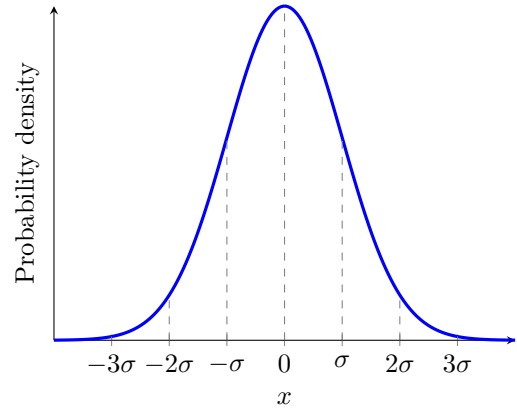
Although Gaussians have infinite support, the scale parameters (s_x, s_y, s_z) correspond to the standard deviations $(\sigma_x, \sigma_y, \sigma_z)$ of the underlying 3D Gaussian distribution. By setting s_x and s_y to half the projected leaf size, we effectively place the rectangle boundary at one standard deviation (1σ) from the mean. This implies that a large fraction of the Gaussian mass lies within the leaf, while the remaining mass smoothly decays beyond its boundaries, as illustrated in Figure 4.10. In practice, Gaussians are often evaluated only within a few standard deviations (e.g., up to 3σ), which yields effectively finite support while allowing overlap between neighboring primitives for smooth reconstruction.

With the initial identity rotation, this parameterization corresponds to a diagonal covariance matrix so that the principal axes of the Gaussian are aligned with the image plane.

$$\Sigma = \begin{bmatrix} s_x^2 & 0 & 0 \\ 0 & s_y^2 & 0 \\ 0 & 0 & s_z^2 \end{bmatrix},$$



(a) 2D Gaussian blob



(b) 1D Gaussian distribution

Figure 4.10: Gaussian initialization in image and 1D space. (Left) A 2D Gaussian blob centered in a rectangular leaf of size $\Delta u \times \Delta v$, where the standard deviations are $s_x = \Delta u/2$ and $s_y = \Delta v/2$, so that the leaf boundaries lie at approximately one standard deviation from the mean. This choice places a large portion of the Gaussian mass inside the leaf while leaving a decaying tail outside, which naturally creates both strong coverage of the cell, but also some bleeding into neighboring leaves. (Right) The marginal probability density function of a standard normal distribution along one axis, with vertical dashed lines at integer multiples of σ . The total area under the curve equals 1, and most of the Gaussian mass is concentrated within $\pm 3\sigma$, meaning that the bulk of the contribution is captured near the mean while outer standard deviations contribute progressively less, supporting smooth blending between adjacent primitives.

This initialization provides a dense, well-structured starting point for the scene, with primitives appropriately placed and sized to match the captured geometric and photometric details. Subsequent optimization then refines these parameters to improve the accuracy of the reconstruction.



Figure 4.11: Hierarchical Gaussian initialization under varying budgets. The hierarchical tree partitioning progressively subdivides the image, with each cell initializing a Gaussian. At very low budgets (e.g., 100 or 1k), only a few large cells are formed, resulting in large Gaussians and a coarse, blurry approximation of the scene. As the budget increases, the tree refines into smaller cells, enabling more localized Gaussian placement. Around 5k Gaussians, the representation becomes structurally consistent, while higher budgets (e.g., 25k–100k) allow fine-grained partitioning that captures significantly more detail, approaching ground-truth appearance.

4.7 Gaussian Optimization

After the Gaussian map has been initialized or expanded, it is further refined through gradient-based optimization. In this work, the optimization procedure follows the same strategy as Gaussian-LIC2 [2], without introducing additional changes.

During optimization, a set of keyframes is sampled and used to update the Gaussian parameters. The rendered RGB image is compared with the corresponding input image, and the sparse LiDAR depth is used as an additional geometric supervision signal. This joint objective improves both visual fidelity and geometric consistency of the reconstructed map.

The optimization uses Adam with CUDA-accelerated rendering and backpropagation. It is performed iteratively for the selected keyframes only, while the remaining Gaussian parameters are kept fixed. In this way, the map is refined incrementally without changing the underlying optimization scheme introduced by Gaussian-LIC2 [2].

5.1 Single-frame test runs

In a first set of experiments, we initialized the Gaussians on a single frame and trained this frame with both methods. We selected a representative sequence from the r3live dataset [6] and, for each configuration, re-initialized the Gaussians from scratch for every frame in the sequence. Each frame was then optimized, and we report the mean PSNR across all frames in the sequence.

Figure 5.1 shows how the mean PSNR evolves over wall-clock optimization time for each Gaussian configuration, covering roughly the first 100 optimization iterations for both methods. The vertical dashed lines and corresponding labels indicate when the PM starts optimizing, and we see that it begins from a substantially higher mean PSNR than Gaussian-LIC2 and then climbs much more quickly in the early phase. Table 5.1 reports the corresponding initial and early-iteration mean PSNR values (at initialization and after 100 iterations), together with their absolute and relative improvements, confirming that across all Gaussian counts the PM dominates Gaussian-LIC2 in this initial time window, with higher PSNR and a narrower spread across frames, indicating more stable behavior.

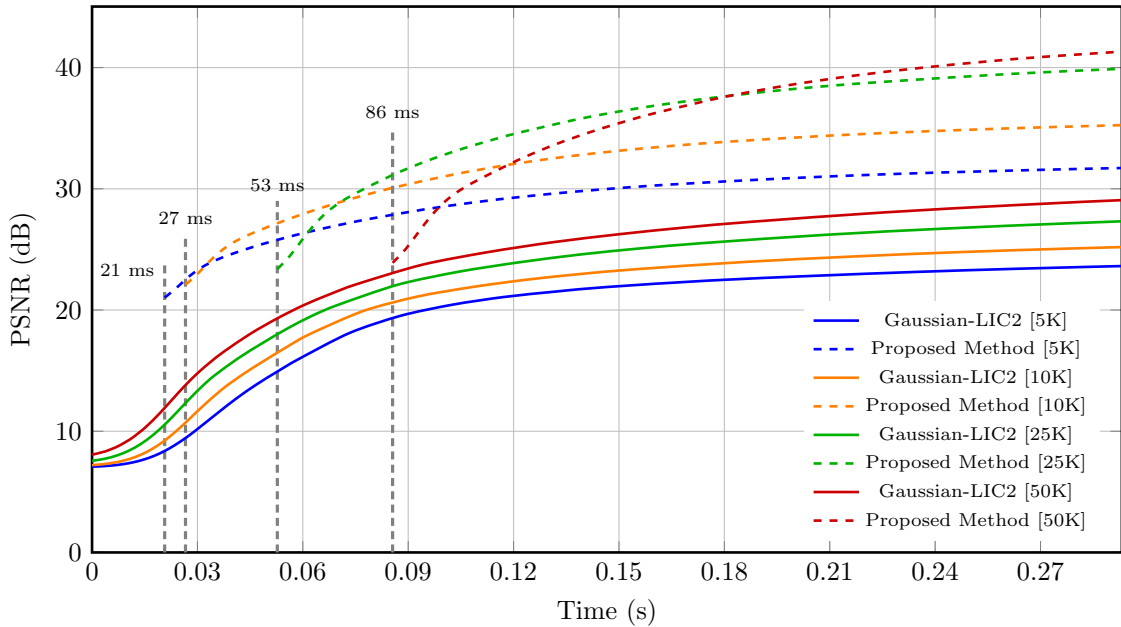


Figure 5.1: Mean PSNR over optimization time for different numbers of Gaussians on an r3live sequence [6]. Solid lines show the mean PSNR across all frames for Gaussian-LIC2, while vertical dashed lines and their labels mark the mean PSNR of the PM at the moment it starts optimization after tree construction.

Max Gaussians	Gaussian-LIC2 (dB)	PM (dB)	Difference (dB)	Improvement (%)
5K	7.076 – 23.628	21.024 – 31.837	13.948 – 8.209	197.1 – 34.74
10K	7.207 – 25.314	22.089 – 35.517	14.882 – 10.203	206.4 – 40.31
25K	7.561 – 27.807	23.379 – 40.834	15.818 – 13.027	209.3 – 46.85
50K	8.068 – 29.887	23.894 – 43.136	15.826 – 13.249	196.2 – 44.33

Table 5.1: Initial and early-iteration mean PSNR comparison on single frames between Gaussian-LIC2 and the PM. Each cell lists initial–100-iteration values for the mean PSNR, followed by the absolute and relative improvements over the same interval. We focus on the first 100 iterations because they highlight the strong advantage of the PM at initialization and the slower convergence of Gaussian-LIC2; asymptotically, the PSNR gap narrows as both methods approach similar values, but Gaussian-LIC2 still converges to a slightly lower PSNR, indicating that the proposed method uses its limited Gaussians more effectively.

We focus on this early phase in time because it clearly exposes both the strong advantage of the PM at initialization and the slower convergence of Gaussian-LIC2. Asymptotically, the PSNR curves for the two methods approach similar values in time, but Gaussian-LIC2 still converges to a slightly lower mean PSNR than our PM, indicating that our method makes more effective use of the available Gaussian budget; this is also reflected qualitatively in Figure 5.2.



Figure 5.2: Single-frame reconstruction quality at 5K Gaussians after 100 optimization iterations. Both methods use the same Gaussian budget, but Gaussian-LIC2 produces a noticeably blurrier image in certain regions (tiled ground, wall at the top left of the image), whereas the PM shows better quality, consistent with the higher PSNR reported in Table 5.1 and the faster convergence trend in Figure 5.1.

5.2 Evaluation without optimization

In a second set of experiments, we evaluate the performance of different initialization strategies without any subsequent optimization, with the goal of isolating the quality of the initial Gaussian placement. We compare three methods: the PM, the uniform sampling strategy of Gaussian-LIC2 [2], and the LoG-based weighted sampling strategy of On-the-fly NVS [1].

We evaluate these methods on multiple sequences from the r3live dataset [6], including *hku_campus*, *hku_park*, *hkust_campus*, and *degenerate_seq*. For each dataset, we report reconstruction quality metrics obtained directly after initialization, without any optimization steps.

As shown in Table 5.2 and Figure 5.3, this comparison highlights how well each method performs purely based on its initialization strategy: the proposed method consistently achieves the highest PSNR and Structural Similarity Index Measure (SSIM) and the lowest Learned Perceptual Image Patch Similarity (LPIPS) across all sequences, indicating superior initialization quality and demonstrating the effectiveness of the PM across different environments and scene characteristics.

Dataset	Metric	Gaussian-LIC2	On-the-fly NVS	PM
hku_campus	Gaussians	672k	664k	525k
hku_campus	PSNR $\uparrow$	12.95	12.72	16.15
hku_campus	SSIM $\uparrow$	0.391	0.350	0.527
hku_campus	LPIPS $\downarrow$	0.610	0.532	0.526
hku_park	Gaussians	647k	633k	435k
hku_park	PSNR $\uparrow$	12.02	11.22	13.53
hku_park	SSIM $\uparrow$	0.335	0.301	0.372
hku_park	LPIPS $\downarrow$	0.589	0.570	0.532
hkust_campus	Gaussians	284k	262k	187k
hkust_campus	PSNR $\uparrow$	13.02	10.22	17.70
hkust_campus	SSIM $\uparrow$	0.424	0.242	0.565
hkust_campus	LPIPS $\downarrow$	0.640	0.608	0.418
degenerate	Gaussians	312k	352k	228k
degenerate	PSNR $\uparrow$	11.06	10.80	16.04
degenerate	SSIM $\uparrow$	0.347	0.347	0.613
degenerate	LPIPS $\downarrow$	0.529	0.458	0.292

Table 5.2: Reconstruction quality without optimization on r3live sequences [6] for Gaussian-LIC2 [2], On-the-fly NVS [1], and the PM.



(a) Gaussian-LIC2



(b) On-the-fly NVS



(c) Proposed Method

Figure 5.3: Qualitative comparison of reconstruction without optimization for different initialization strategies. From left to right and top to bottom: Gaussian-LIC2, On-the-fly NVS, and the proposed method.

5.3 Comparing KD-Tree to Quadtree

In a third set of experiments, we compare the use of a KD-tree and a quadtree for Gaussian initialization. As in Section 5.1, these experiments are conducted on individual still images, where each frame is initialized and optimized independently using an equal number of Gaussians. The evaluation is performed over a dataset of 105 frames, and we report the average PSNR across all frames at different stages of the optimization process.

Table 5.3 reports the mean PSNR at initialization (iteration 0), mid-optimization (iteration 50), and after convergence (iteration 100). At iteration 0, the quadtree consistently achieves slightly higher PSNR values across all Gaussian counts, indicating a modest advantage in initialization quality. However, this trend reverses as optimization progresses. Already by iteration 50, the KD-tree matches or surpasses the quadtree in all configurations, with the gap becoming more pronounced at higher Gaussian counts.

Table 5.3: PSNR Comparison: Iteration 0, 50 and 100

Max Gaussians	Mean PSNR at iter 0		Mean PSNR at iter 50		Mean PSNR at iter 100	
	KD-Tree	Quadtree	KD-Tree	Quadtree	KD-Tree	Quadtree
1,000	16.94	16.97	20.68	20.65	21.25	21.21
5,000	18.79	18.82	25.63	25.38	26.54	26.28
10,000	19.70	19.84	28.66	27.55	29.86	28.50
50,000	22.20	22.45	38.20	38.02	40.42	40.22
100,000	22.94	23.00	41.59	40.47	44.02	42.87

Figure 5.4 illustrates the complete evolution of mean PSNR over all 100 iterations for both methods. The dashed lines represent KD-tree initialization, while solid lines indicate quadtree initialization, with matching colors denoting the same Gaussian count. The visualization clearly shows the crossover point where KD-tree performance overtakes quadtree, occurring progressively earlier for higher Gaussian counts.

After 100 iterations, the KD-tree consistently achieves higher PSNR for all tested Gaussian counts. The difference is particularly noticeable at 10K and 100K Gaussians, as shown in Figure 5.5, which plots the PSNR difference (KD-tree minus quadtree) for each configuration. Positive values indicate superior KD-tree performance, with the performance gap growing substantially larger at higher Gaussian counts. This suggests that the KD-tree structure facilitates more effective optimization and better utilization of increased model capacity. Overall, while the quadtree provides a marginally better starting point, the KD-tree leads to superior final reconstruction quality.

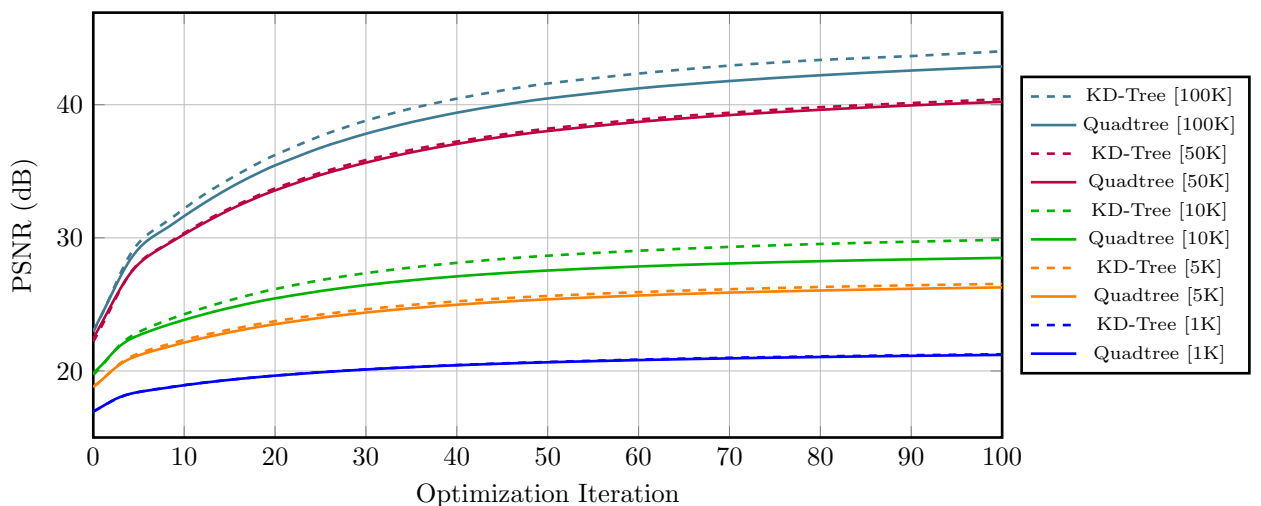


Figure 5.4: Mean PSNR evolution over 100 optimization iterations comparing KD-Tree (dashed lines) and Quadtree (solid lines) initialization strategies across different Gaussian counts. While Quadtree achieves marginally higher initial PSNR values, KD-Tree consistently surpasses it after optimization for all tested configurations.

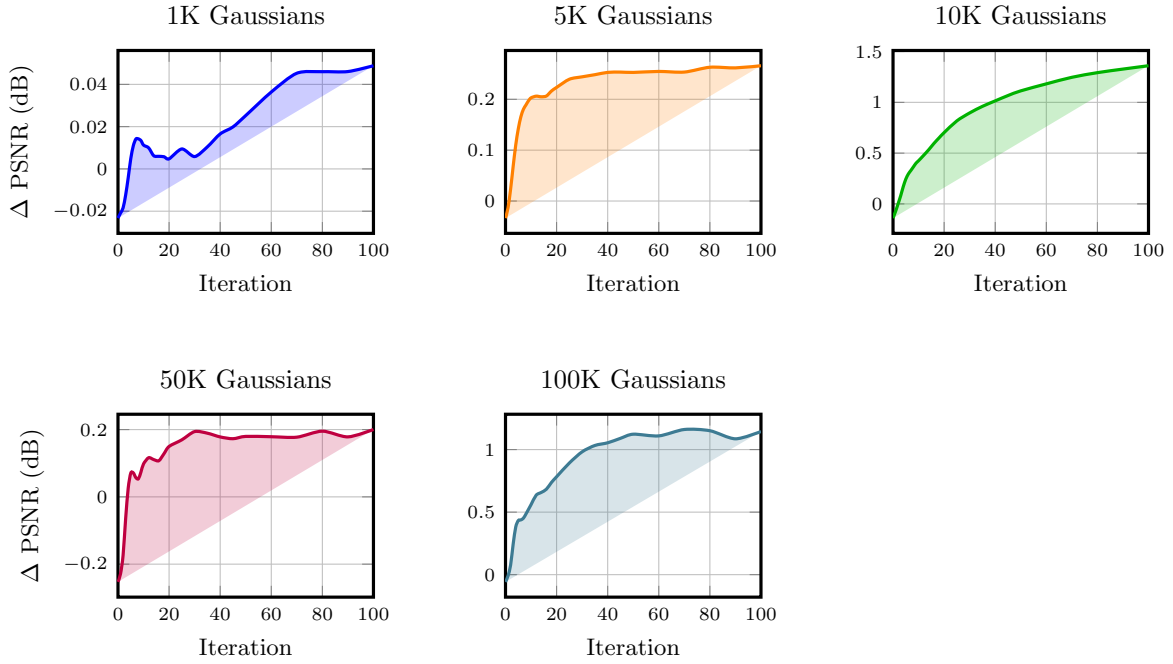


Figure 5.5: Difference in mean PSNR (KD-Tree minus Quadtree) over optimization iterations for each Gaussian count. Positive values indicate KD-Tree outperforms Quadtree. The shaded area emphasizes the performance gap, which grows larger with increased Gaussian counts.

Figure 5.6 provides qualitative evidence for the quantitative improvements observed in the KD-tree approach. The zoomed-in region reveals that the KD-tree method initializes Gaussians in a way that better aligns with the actual structure of the scene, particularly visible in the pebble pavement texture. While the quadtree reconstruction in Figure 5.6b shows some blurring and loss of individual pebble definition, the KD-tree reconstruction in Figure 5.6c preserves finer details and maintains clearer boundaries between individual pebbles. This superior structural alignment during initialization allows the Gaussians to better fit the actual objects present in the scene throughout optimization, ultimately contributing to the higher PSNR values observed in Table 5.3.

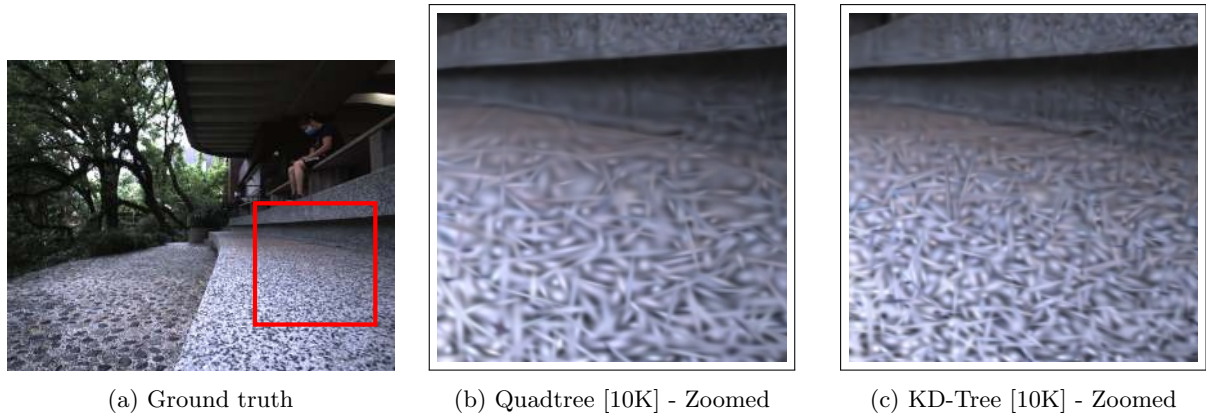


Figure 5.6: Visual comparison of KD-Tree and Quadtree reconstruction quality after 100 optimization iterations with 10K Gaussians. The red box in (a) indicates the zoomed region shown in (b) and (c). The KD-Tree produces sharper details and better preserves fine structures compared to Quadtree.

5.4 Hyperparameter analysis

In the next set of experiments, we study the influence of several key initialization hyperparameters of the proposed method on reconstruction quality and model size. To this end, we perform a systematic sweep over four parameters on four representative sequences from the r3live dataset [6]: *hku_campus_seq*, *hku_park*, *hkust_campus*, and a *degenerate_seq*. For each configuration, we measure both training and novel-view PSNR, SSIM, and LPIPS, and we record the total number of Gaussians. We then construct Pareto frontiers with respect to novel-view PSNR and total Gaussians, and analyze which hyperparameter settings appear on these fronts across datasets.

The following hyperparameter ranges are explored:

- **Max amount of leaves per frame:** {1000, 2500, 5000, 7500, 10000, 25000, 50000}
- **Gaussian initial opacity:** {0.4, 0.7, 1.0}
- **Opacity threshold τ :** {0.4, 0.7, 0.9}
- **Edge Canny range:** {(10, 30), (30, 90), (50, 150)}

To keep the hyperparameter sweep computationally tractable while still covering diverse scene content, all sequences were temporally truncated to approximately 30 seconds. This is shorter than the full length of some r3live sequences. The shortened segments were selected to remain representative of the scene structure and motion, ensuring that the observed trends in reconstruction quality and model complexity remain meaningful while significantly reducing total training time.

Figure 5.7 visualizes the resulting scatter plots for all four sequences. Each point corresponds to a single run and is placed according to the number of Gaussians and the novel-view PSNR. Blue points denote dominated configurations, while the red line traces the Pareto-optimal frontier. The corresponding Pareto-optimal hyperparameter settings and metrics are listed in the longtables in Appendix B.

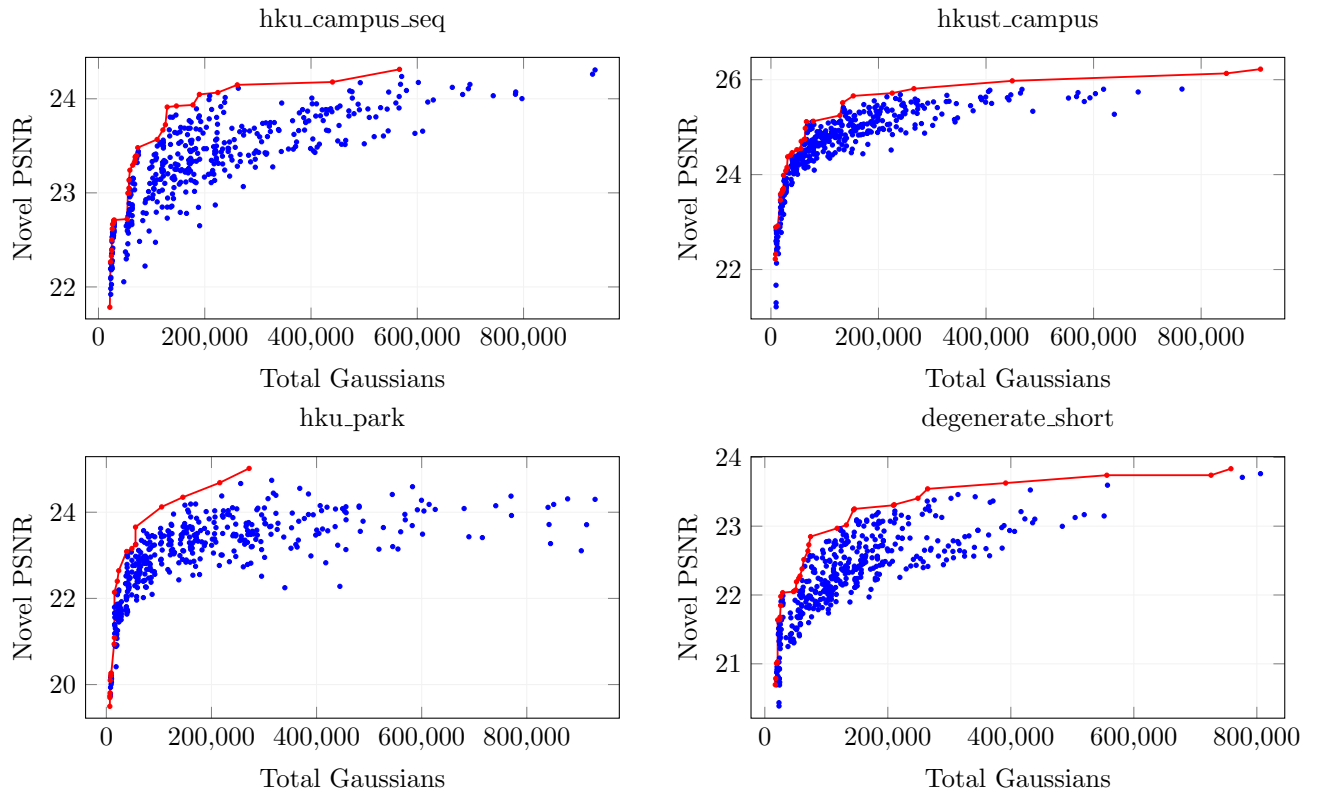


Figure 5.7: Scatter plots of hyperparameter sweep with Pareto frontiers for every scene from the r3live dataset [6]

5.4.1 Effect of the max amount of leaves per frame

The *Tree Max Leaves* parameter directly controls how many Gaussians are initialized per frame, and therefore strongly influences the total number of Gaussians in the final model. As expected, lower values of this parameter result in fewer Gaussians overall, which in turn generally leads to lower novel view PSNR. This reflects a reduced representational capacity, especially in scenes with complex geometry or fine details.

However, increasing the number of leaves does not necessarily lead to equally strong improvements in final reconstruction quality. While higher values produce a much sharper and more accurate initialization, as also reflected by the higher initial visual quality in Figure 4.11, the final PSNR after optimization only increases moderately. In contrast, configurations with fewer Gaussians often start from noticeably blurrier initial reconstructions, yet still converge to strong results due to the optimization capabilities of 3DGS.

5.4.2 Effect of initial opacity

The *Initial Opacity* parameter controls the starting transmittance of Gaussians at initialization. Across all four sequences, Pareto-optimal runs are obtained for all three tested values (0.4, 0.7, and 1.0). As displayed in Figure 5.8, smaller Gaussian budgets configurations are more frequently associated with higher initial opacities (0.7 and 1.0), indicating that making Gaussians more opaque from the start helps to concentrate capacity on salient structures when only few primitives are available.

In contrast, runs with lower initial opacity (0.4) increasingly dominate the high-budget end of the frontier, achieving the best PSNR at the largest model sizes. In this regime, a more transparent initialization appears to better exploit the larger capacity: Gaussians can gradually increase their opacity during optimization, allowing the model to refine geometry and appearance more flexibly before saturating. This effect is consistently visible across all four sequences, where the right-most, highest-PSNR points along the front are predominantly associated with lower initial opacity.

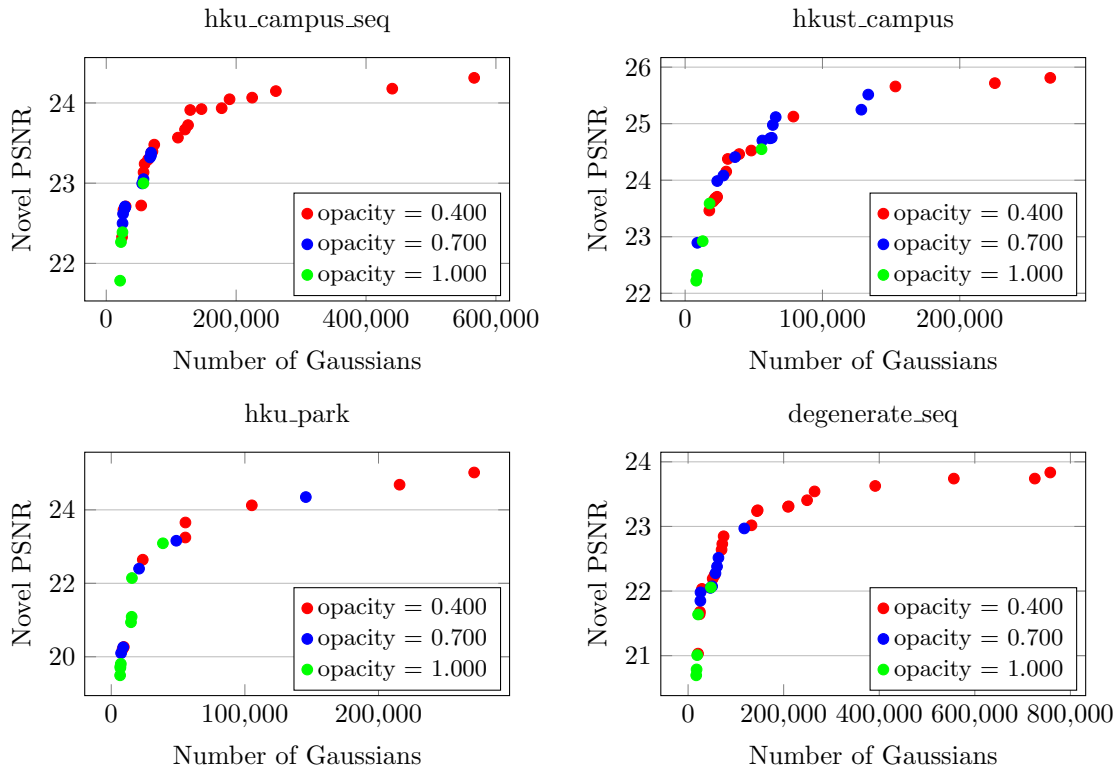


Figure 5.8: Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the initial opacity, highlighting how different opacity values populate distinct regions of the quality-complexity trade-off.

5.4.3 Effect of opacity threshold τ

The *opacity threshold* τ controls how strictly the coverage mask classifies pixels as already represented by existing Gaussians (Section 4.5.1). Lower values of τ mark pixels as covered as soon as they reach a modest accumulated opacity, which aggressively prunes regions from further refinement and prevents new Gaussians from being placed in already semi-represented areas. In contrast, higher values of τ require near-saturated opacity before a pixel is considered covered, allowing the tree to keep subdividing partially represented regions and to invest more Gaussians in further refining them as we saw in Figure 4.8. Figure 5.9 summarizes the impact of the τ threshold on the quality–complexity trade-off across the four r3live sequences.

Lower τ values appear on the Pareto front precisely in the low-budget regime. When only few Gaussians can be initialized per frame, a conservative strategy that avoids spending capacity on semi-represented regions is beneficial: the coverage mask quickly marks such regions as covered and concentrates the limited budget on genuinely new, previously unseen parts of the scene. This yields compact yet Pareto-optimal reconstructions for small total Gaussian counts.

In contrast, runs with $\tau = 0.9$ systematically shift toward the high-budget, high-PSNR end of the front, where the per-frame leaf budget is much larger. In this regime, it becomes advantageous to also refine semi-represented regions: the more permissive coverage mask keeps these areas available for new Gaussians, allowing the model to spend part of its larger budget on densifying boundaries, filling small holes, and sharpening fine structures. As a result, the reconstruction quality improves, at the cost of a substantial increase in total Gaussians.

The intermediate value $\tau = 0.7$ tends to populate the middle of the Pareto front, where per-frame budgets are neither extremely tight nor extremely generous. It prunes clearly covered regions early, like $\tau = 0.4$, but still allows additional Gaussians to be placed in partially opaque regions when the budget permits, similar to $\tau = 0.9$. This explains why its Pareto-optimal points typically lie between the $\tau = 0.4$ and $\tau = 0.9$ configurations in both PSNR and total number of Gaussians.

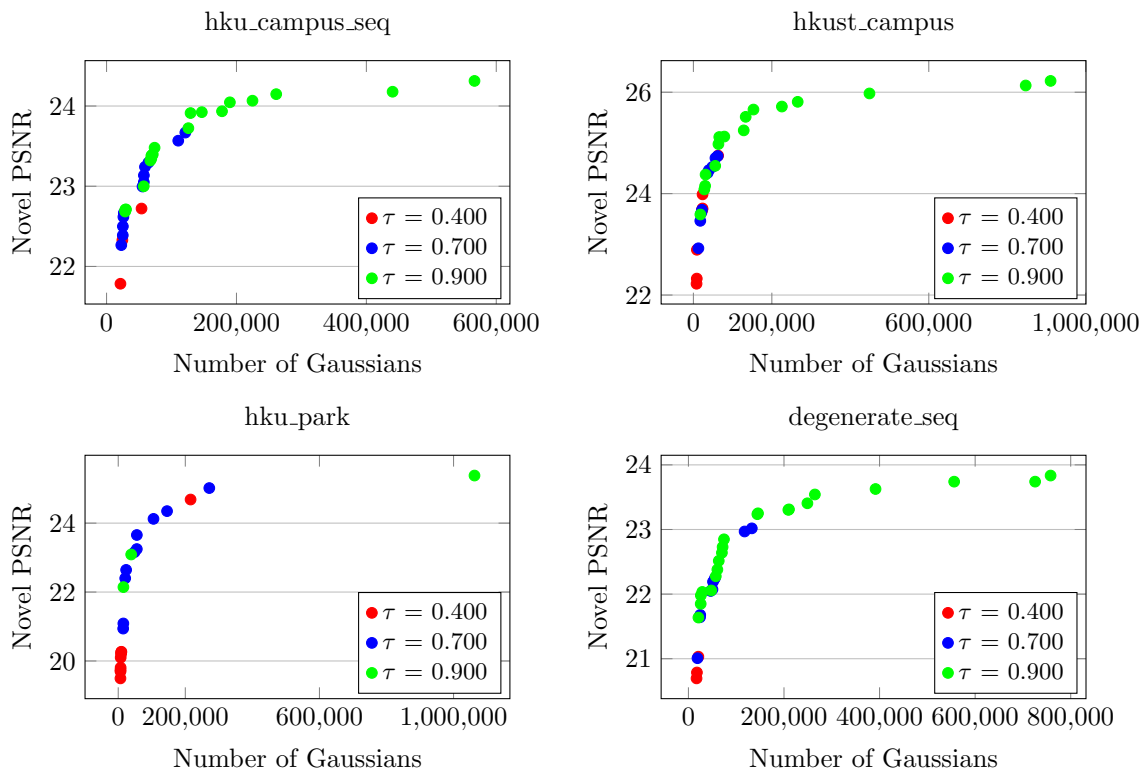


Figure 5.9: Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the initial opacity, highlighting how different opacity values populate distinct regions of the quality–complexity trade-off.

5.4.4 Effect of RGB image Canny edge threshold range

The *RGB image Canny edge threshold range* controls the lower and upper thresholds used by the Canny detector to extract image gradients that guide Gaussian placement. Lower thresholds such as (10, 30) detect many weak gradients and fine textures as edges, producing a much denser edge map that causes the hierarchical tree to split more aggressively wherever the leaf budget allows. In contrast, higher thresholds like (50, 150) raise both the low and high values, so only strong, high-contrast contours are retained; large parts of the image then contain no edges, as illustrated in Figure 4.4, the tree stops subdividing earlier in those regions, and fewer Gaussians are initialized there.

Figure 5.10 shows how the three tested RGB ranges (10, 30), (30, 90), and (50, 150) populate the Pareto front across the four r3live sequences. For the *hku_campus_seq*, *hkust_campus* and *degenerate_seq* sequences, the lowest range (10, 30) (red) clearly dominates the high-PSNR, high-Gaussian end of the front: almost all right-most points with the largest Gaussian counts use (10, 30). This matches the behavior of the hierarchical tree: with a large max-leaves budget and a very dense edge map, the tree keeps splitting and the method can spend the additional Gaussians on resolving a lot of fine-scale structure, at the cost of a very large model.

In these same sequences, the highest range (50, 150) (green) appears more frequently at lower Gaussian counts and in the mid-PSNR regime. The sparser edge map produced by (50, 150) causes the tree to stop splitting earlier in many regions, which fits better when the Gaussian budget per frame is small: the method focuses on the strongest edges and avoids exploding the number of leaves. The mid-range (30, 90) (blue) mostly fills the transition between these two extremes, occupying intermediate budgets and PSNR values.

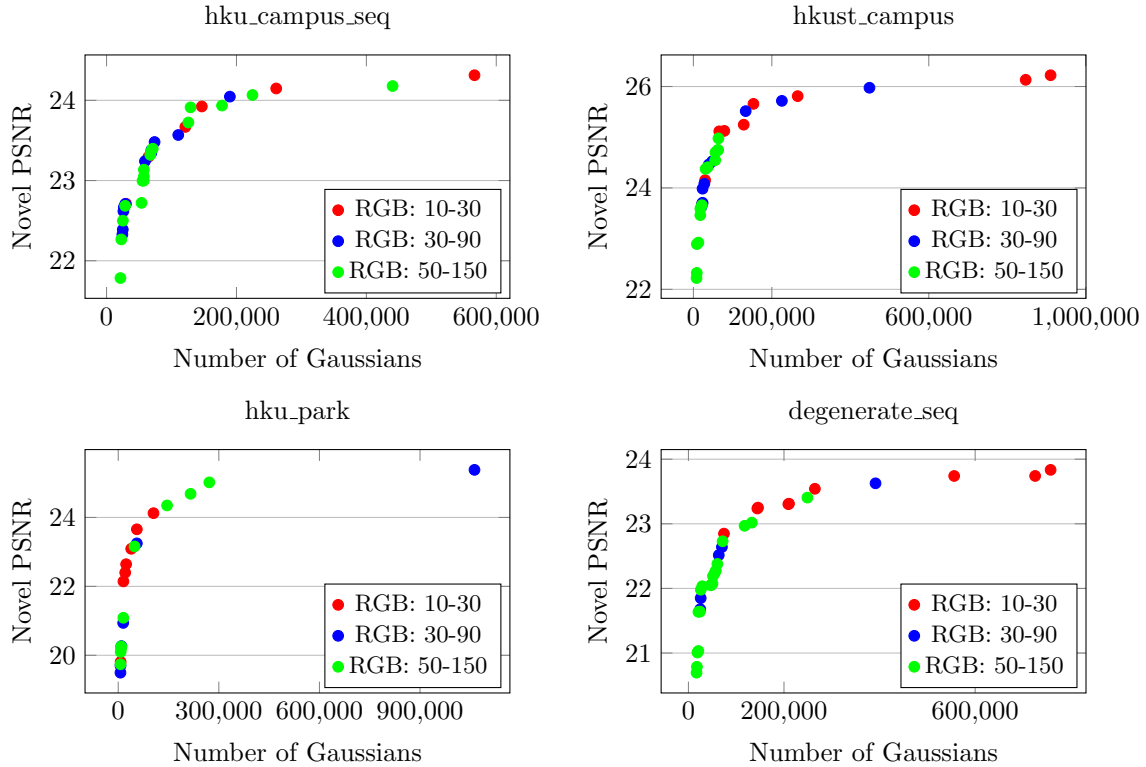


Figure 5.10: Pareto-optimal configurations for four r3live sequences, showing novel-view PSNR versus total number of Gaussians. Points are color-coded by the RGB image edge threshold range, highlighting how different canny thresholds populate distinct regions of the quality–complexity trade-off.

The *hku_park* sequence deviates from the behaviour observed in the other three datasets. Here, the (10, 30) setting appears mainly at low to moderate Gaussian counts and reaches only mid-range PSNR values, while the highest-PSNR Pareto points are obtained with the higher ranges (50, 150) and, at the extreme high-budget end, (30, 90). This is plausible given the content of the scene: the large amount of vegetation and pebble pavement produces many weak gradients, so (10, 30) detects a very dense set of edges that can be exploited at moderate budgets, but when the budget becomes large enough, concentrating additional Gaussians on the stronger structures emphasized by (50, 150) (and eventually (30, 90)) yields better PSNR.

Figure 5.11 shows a ground-truth frame from *hku_park* together with the corresponding Canny edges for (10, 30) and (50, 150). The (10, 30) setting produces a very dense edge map in which almost every small texture and intensity fluctuation in the vegetation and pebble ground is marked as an edge, so the result no longer provides a clear notion of dominant boundaries. In contrast, the (50, 150) range suppresses most of these weak responses and retains a much sparser set of strong, structurally meaningful contours, which aligns with the higher PSNR observed for (50, 150) at comparable Gaussian budgets.

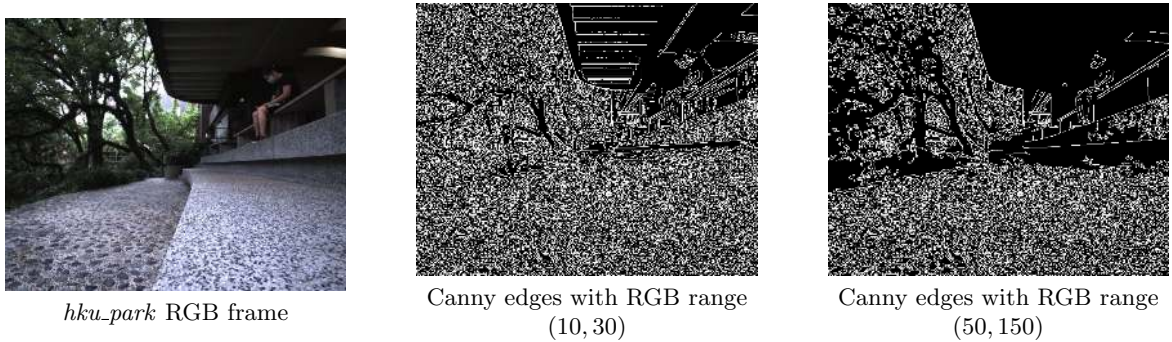


Figure 5.11: Effect of the RGB image Canny edge threshold range on a frame from the *hku_park* sequence. With the low range (10, 30), almost all fine texture in the vegetation and pebble ground is detected as an edge, resulting in a heavily populated edge map with little emphasis on dominant structures. The higher range (50, 150) filters out most of these weak responses and preserves a sparser set of strong contours, providing a more meaningful edge signal for guiding Gaussian placement and explaining the higher PSNR obtained for this range at comparable Gaussian budgets.

5.5 Comparison

In the previous section, we analyzed how different initialization hyperparameters shape the quality–complexity trade-off of our method and identified Pareto-optimal configurations across multiple sequences. While this provides insight into the internal behavior and efficiency of the approach, it does not yet position the method relative to existing work.

We now turn to a direct comparison with Gaussian-LIC2 [2], which currently stands as the state of the art in real-time LiDAR-assisted 3DGS. Using the Pareto-optimal runs identified earlier, we evaluate both methods under comparable conditions to understand in which settings our approach achieves higher reconstruction quality with smaller models, and where Gaussian-LIC2 remains superior.

Evaluation Setup For Gaussian-LIC2, we conducted two types of evaluations to ensure a fair and informative comparison. First, we ran the method using its default configuration, allowing it to operate without constraints on the number of Gaussians introduced per frame, reflecting its standard performance. To better align with our controlled experiments, we additionally introduced a constrained variant in which the number of Gaussians added per frame is limited, mirroring the mechanism used in our method. This enables a direct comparison of the quality–complexity trade-off under similar conditions.

For the quantitative comparison in Appendix C, we report Pareto-optimal runs of our method alongside Gaussian-LIC2. We enforce that the total number of Gaussians in our selected runs is always less than the final number of Gaussians produced by the corresponding Gaussian-LIC2 run. This ensures that any observed quality differences are achieved under stricter model complexity constraints.

Furthermore, Figure 5.12 presents per-dataset difference curves for the reconstruction quality metrics (PSNR, SSIM, and LPIPS). Each point on these curves corresponds to a pair of runs: one from Gaussian-LIC2 and one from our PM, matched by the total number of Gaussians used by Gaussian-LIC2. The x-axis shows this Gaussian count of the Gaussian-LIC2 run, while the y-axis shows the difference in the corresponding metric between the two methods. For PSNR and SSIM, we plot differences as $\text{PM} - \text{Gaussian-LIC2}$, whereas for LPIPS we plot $\text{Gaussian-LIC2} - \text{PM}$, so that in all three plots positive values indicate better performance of the PM. This asymmetric sign convention keeps the direction of improvement consistent across metrics, even though higher PSNR/SSIM and lower LPIPS are desirable.

Observations Inspecting the results in Appendix C together with the difference plots in Figure 5.12, several trends become apparent. Across most datasets, our PM achieves higher PSNR than Gaussian-LIC2 at lower Gaussian counts, with the notable exception of the *hkust_campus* sequence, where Gaussian-LIC2 consistently outperforms our method across all three metrics.

This behavior on *hkust_campus* can be explained by the camera motion: the trajectory starts far from the scene and then repeatedly revisits the same content from much closer viewpoints, so our dense initialization already placed many Gaussians in these regions based on distant views and subsequently optimizes them, leaving less freedom to introduce new primitives when previously unresolved fine details become visible nearby. Gaussian-LIC2, which initializes less aggressively per frame, retains more flexibility to extend its representation as the camera approaches, giving it a structural advantage on such inward-moving trajectories.

In contrast, on the *hku_park* dataset the camera primarily rotates around roughly the same distance without strongly approaching regions that were already initialized, so the initial dense allocation remains well matched to the observed content and our method outperforms Gaussian-LIC2 more consistently, which is reflected by differences that favor the PM across a wider range of Gaussian counts.

For the *hku_campus_seq* and *degenerate_seq* datasets, the differences between both methods are less pronounced. The SSIM and LPIPS curves remain close to zero over most Gaussian counts, with neither method consistently dominating the other, and a similar pattern is observed for PSNR, where differences fluctuate slightly around zero but remain within a narrow margin overall.

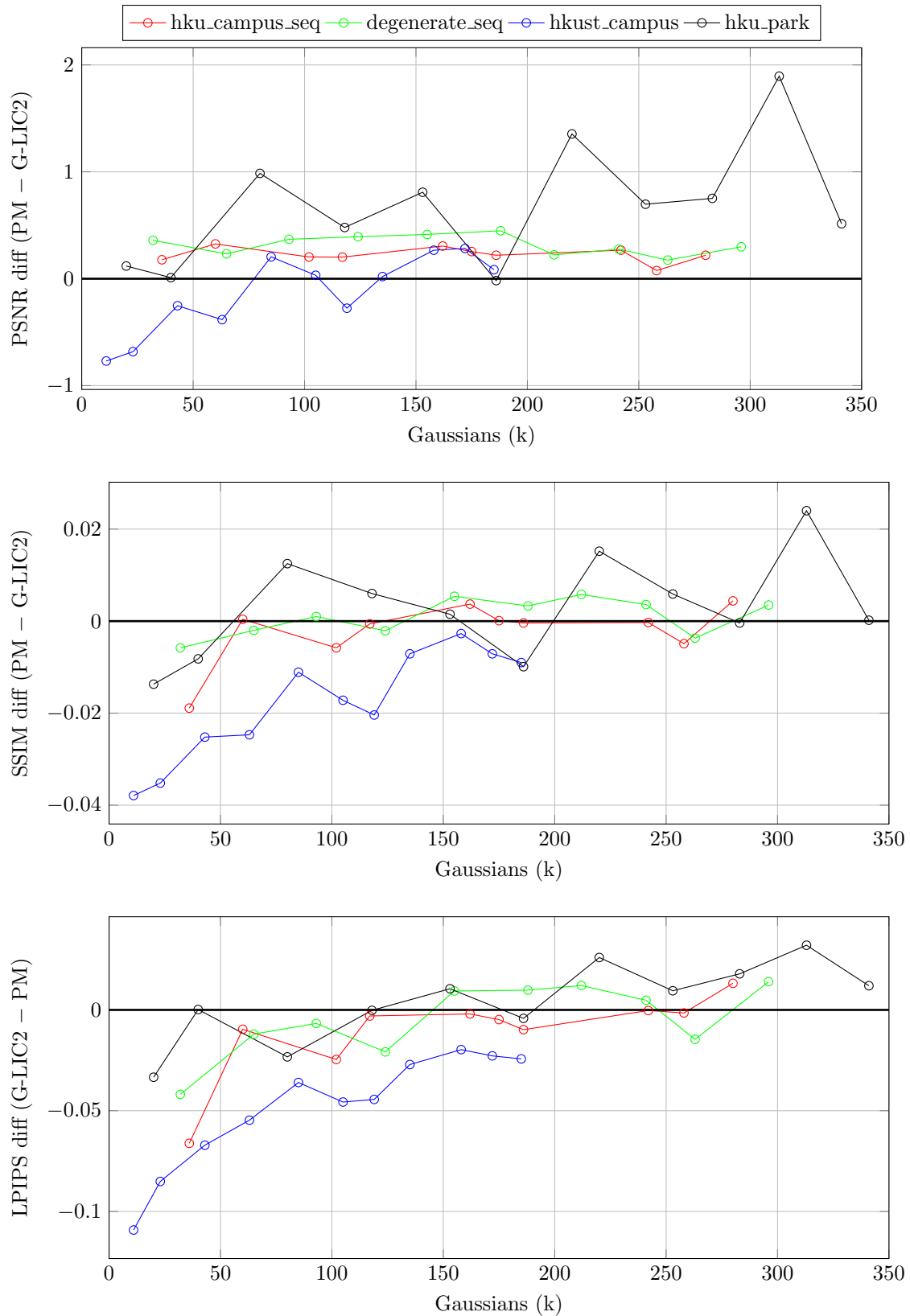


Figure 5.12: Per-dataset metric differences between the proposed method (PM) and Gaussian-LIC2 as a function of the Gaussian count of Gaussian-LIC2. Each point in the curves represents a matched pair of runs (one Gaussian-LIC2 run and one PM run) that share the same total number of Gaussians in the Gaussian-LIC2 configuration; the x-axis shows this Gaussian count (in thousands), and the y-axis shows the difference in the corresponding quality metric between the two methods. For PSNR and SSIM we plot $\text{PM} - \text{Gaussian-LIC2}$, while for LPIPS we plot $\text{Gaussian-LIC2} - \text{PM}$, so that in all three subplots positive values always indicate better performance of PM.

Conclusion

This work investigated how to design an online, real-time dense initialization pipeline for 3DGS that allocates Gaussians strategically, using more primitives in geometrically and photometrically complex regions and fewer in homogeneous areas. Rather than modifying the optimization stage, the focus was on improving how Gaussians are initialized so that a compact set of primitives provides strong coverage and a good starting point for refinement.

To this end, we first complete the sparse LiDAR depth to obtain a dense depth map, and then fuse photometric and geometric edges into a unified edge map. Based on this fused representation, we perform an adaptive partitioning of RGB-D images using a hierarchical tree. The tree partitions each image under a fixed leaf budget, so that complex regions are split into many small cells while homogeneous regions are covered by fewer, larger cells. Each leaf cell initializes a single Gaussian whose position and scale are derived from the spatial extent of the leaf’s corresponding rectangle, along with the depth and camera intrinsics and extrinsics. This yields a dense, structured initialization that can be optimized by Gaussian-LIC2 without changes to the underlying training loop.

Experiments show that this structured initialization yields a higher-quality starting point and more efficient use of Gaussians, achieving comparable or improved reconstruction quality with fewer primitives and without sacrificing real-time performance. In single-frame experiments, the proposed method consistently attains higher initial PSNR and faster early convergence than Gaussian-LIC2 under the same Gaussian budgets. In full-sequence evaluations, the method matches or surpasses Gaussian-LIC2 on most trajectories while using fewer Gaussians overall, with the main weakness appearing on sequences where the camera starts far away and later revisits the same content from much closer viewpoints, a setting in which our dense early allocation leaves less flexibility to specialize new Gaussians for newly visible fine details.

While incorporating LiDAR-derived depth edges provides additional robustness in low-texture or poorly constrained regions, the hierarchical initialization itself is agnostic to the depth source and can also be deployed in purely RGB-based pipelines. In such cases, it would inherit the usual limitations of monocular or SfM-based depth, but still offer the same structured coverage and budget control, suggesting that dense, edge-guided initialization is generally useful for 3DGS beyond LiDAR-assisted settings.

Remaining challenges suggest several directions for future work, including more adaptive splitting criteria that better respond to local scene structure, richer guidance for identifying important regions (for example via learned importance or semantic cues), mechanisms to selectively reinitialize or locally rebalance Gaussians when the camera later observes previously initialized regions from much closer viewpoints, and refined Gaussian placement within each leaf to reduce boundary artifacts and small unfilled areas. Overall, the results highlight initialization as a key lever for improving the efficiency and compactness of 3DGS reconstructions, with LiDAR acting as a valuable but not strictly necessary enhancement to the proposed pipeline.

Bibliography

- [1] A. Meuleman, I. Shah, A. Lanvin, B. Kerbl, and G. Drettakis, “On-the-fly reconstruction for large-scale novel view synthesis from unposed images,” *ACM Transactions on Graphics*, vol. 44, no. 4, p. 1–14, 2025.
- [2] X. Lang, J. Lv, K. Tang, L. Li, J. Huang, L. Liu, Y. Liu, and X. Zuo, “Gaussian-lic2: Lidar-inertial-camera gaussian splatting slam,” 2025.
- [3] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3d gaussian splatting for real-time radiance field rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 139:1–139:14, 2023.
- [4] X. Lang, C. Chen, K. Tang, Y. Ma, J. Lv, Y. Liu, and X. Zuo, “Coco-lic: Continuous-time tightly-coupled lidar-inertial-camera odometry using non-uniform b-spline,” *IEEE Robotics and Automation Letters*, vol. 8, p. 7074–7081, Nov. 2023.
- [5] H. Wang, M. Yang, X. Zheng, and G. Hua, “Scale propagation network for generalizable depth completion,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 3, pp. 1908–1922, 2025.
- [6] J. Lin and F. Zhang, “R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package,” 2021.
- [7] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4104–4113, 2016.
- [8] J. Jung, J. Han, H. An, J. Kang, S. Park, and S. Kim, “Relaxing accurate initialization constraint for 3D gaussian splatting,” *arXiv preprint arXiv:2403.09413*, 2024.
- [9] Z. Guo, J. Su, C. Qian, S. Wang, J. Fan, J. Zhang, W. Zhou, H. Amirpour, Y. Zhao, L. Han, and P. Wang, “Gp-gs: Gaussian processes densification for 3d gaussian splatting,” 2026.
- [10] C. Jiang, R. Gao, K. Shao, Y. Wang, R. Xiong, and Y. Zhang, “Li-gs: Gaussian splatting with lidar incorporated for accurate large-scale reconstruction,” 2024.
- [11] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, “DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [12] C. Zhao, S. Sun, R. Wang, Y. Guo, J.-J. Wan, Z. Huang, X. Huang, Y. V. Chen, and L. Ren, “TCLC-GS: Tightly coupled LiDAR-camera gaussian splatting for autonomous driving,” *arXiv preprint arXiv:2404.02410*, 2024.
- [13] G. Hess, C. Lindström, M. Fatemi, C. Petersson, and L. Svensson, “SplatAD: Real-time LiDAR and camera rendering with 3D gaussian splatting for autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [14] Y. Sun, Y. Zhou, B. Tian, H. Wang, Y. Zhao, and S.-T. Wu, “Gaussian-UDSR: Real-time unbounded dynamic scene reconstruction with 3D gaussian splatting,” *Applied Sciences*, vol. 15, no. 11, p. 6262, 2025.
- [15] R. Song, T. Cai, M. Gross, Y. Zhang, W. Zimmer, Z. Huang, O. Wysocki, and J. Ma, “EnerGS: Energy-based gaussian splatting with partial geometric priors,” *arXiv preprint*, 2026. Semantic Scholar ID: 287874217.
- [16] J. Park, C. Park, M. Kim, M. Kim, and S. Kim, “LIVE-GS: Online LiDAR-inertial-visual state estimation and globally consistent mapping with 3D gaussian splatting,” *arXiv preprint*, 2025. Semantic Scholar ID: 280401025.

- [17] D. Li, Y. Liu, J. Liu, Z. Sun, Z. Huang, S. Sun, J. Liu, C. Yuan, H. Guo, F. E. Tay, and M. H. Ang, “RMGS-SLAM: Real-time multi-sensor gaussian splatting SLAM,” *arXiv preprint*, 2026. Semantic Scholar ID: 287473050.
- [18] R. Birkel, D. Wofk, and M. Müller, “Midas v3.1 – a model zoo for robust monocular relative depth estimation,” 2023.
- [19] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, “Depth anything v2,” 2024.
- [20] C. Fu, C. Mertz, and J. Dolan, “Lidar and monocular camera fusion: On-road depth completion for autonomous driving,” pp. 273–278, 10 2019.
- [21] J. Artois, P. Lambert, and G. V. Wallendael, “MVS-Splatting: Fast multi-view stereo depth fusion for 3D gaussian splatting initialization,” *IEEE Access*, 2025.
- [22] Y. Zheng, Z. Jiang, S. He, Y. Sun, J. Dong, H. Zhang, and Y. Du, “NexusGS: Sparse view synthesis with epipolar depth priors in 3D gaussian splatting,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [23] J. Edstedt, Q. Sun, G. Bökman, M. Wadenbäck, and M. Felsberg, “Roma: Robust dense feature matching,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 19790–19800, IEEE, 2024.
- [24] D. Kotovenko, O. Grebenkova, and B. Ommer, “Edgs: Eliminating densification for efficient convergence of 3dgs,” 2026.
- [25] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, “DUST3R: Geometric 3D vision made easy,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [26] X. Sun, R. Chen, M. Gong, D. Xu, and T. Liu, “Intern-GS: Vision model guided sparse-view 3D gaussian splatting,” *arXiv preprint arXiv:2505.20729*, 2025.
- [27] N. Takama, S. Ito, K. Ito, H.-T. Chen, and T. Aoki, “Sparse2DGS: Sparse-view surface reconstruction using 2D gaussian splatting with dense point cloud,” *arXiv preprint arXiv:2505.19854*, 2025. Accepted at ICIP 2025.
- [28] J. Ha and J. Yi, “Fast and slim splat: Efficient 3D gaussian splatting via confidence and opacity-guided caching and sampling,” in *Proceedings of the International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*, 2025.
- [29] Y.-H. Li, T. Sikora, S. Knorr, and M. Sjöström, “Segmentation-driven initialization for sparse-view 3D gaussian splatting,” *arXiv preprint arXiv:2509.11853*, 2025.
- [30] P. Patt, L. Huang, Y. Zhang, and Y. Lei, “DensifyBeforehand: LiDAR-assisted content-aware densification for efficient and quality 3D gaussian splatting,” *arXiv preprint arXiv:2511.19294*, 2025.
- [31] E. Giacomini, L. D. Giammarino, L. D. Rebotti, G. Grisetti, and M. R. Oswald, “Splat-loam: Gaussian splatting lidar odometry and mapping,” 2025.
- [32] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.



Algorithm

Algorithm 1 Pseudocode of the breadth-first hierarchical partitioning procedure (quadtree & KD-tree)

Require: Edge map E , Budget B , MinSize S_{min} , Mode $\in \{\text{Quad}, \text{KD}\}$

```
1:  $Leaves \leftarrow \emptyset$ 
2:  $Frontier \leftarrow \{\text{ImageBoundingBox}\}$ 
3: while  $Frontier \neq \emptyset$  do
4:    $NextFrontier \leftarrow \emptyset$ 
5:   for each  $R \in Frontier$  do
6:     if  $|Leaves| + |Frontier| + |NextFrontier| \geq B$  then
7:        $Leaves \leftarrow Leaves \cup \{R\}$  ▷ Budget reached, force leaf
8:       continue
9:     end if
10:    if  $\text{width}(R) \leq S_{min}$  or  $\text{height}(R) \leq S_{min}$  or  $\sum_{(u,v) \in R} E(u,v) == 0$  then
11:       $Leaves \leftarrow Leaves \cup \{R\}$ 
12:    else
13:      if Mode == Quad then
14:         $Children \leftarrow \text{SplitIntoFourQuadrants}(R)$ 
15:      else if Mode == KD then
16:        if AspectRatio( $R$ ) > Threshold then
17:           $\theta^* \leftarrow \text{LongestAxis}(R)$ 
18:        else
19:           $\theta^* \leftarrow \arg \max_{\theta \in \{\text{horz}, \text{vert}\}} \left| \sum_{R_1^\theta} E - \sum_{R_2^\theta} E \right|$ 
20:        end if
21:         $Children \leftarrow \text{SplitIntoTwoHalves}(R, \theta^*)$ 
22:      end if
23:       $NextFrontier \leftarrow NextFrontier \cup Children$ 
24:    end if
25:  end for
26:   $Frontier \leftarrow NextFrontier$ 
27: end while
28: return  $Leaves$ 
```



Pareto-optimal runs

Table B.1: Pareto-optimal runs for hku_campus_seq

Gaussians	Max Gaussians	Opacity	Coverage	RGB Canny	Novel Metrics		
					PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
21360	1000	1.000	0.400	50.0–150.0	21.7844	0.6989	0.3682
22656	1000	1.000	0.700	50.0–150.0	22.2659	0.7105	0.3357
24336	1000	0.400	0.400	30.0–90.0	22.3284	0.7084	0.3548
24973	1000	1.000	0.700	30.0–90.0	22.3887	0.7140	0.3282
25112	1000	0.700	0.700	50.0–150.0	22.4997	0.7166	0.3303
25898	1000	0.700	0.700	30.0–90.0	22.6176	0.7193	0.3291
26536	1000	0.400	0.700	30.0–90.0	22.6669	0.7225	0.3274
28621	1000	0.700	0.900	50.0–150.0	22.6846	0.7229	0.3187
28806	1000	0.700	0.900	30.0–90.0	22.6974	0.7256	0.3184
28905	1000	0.700	0.900	30.0–90.0	22.7089	0.7257	0.3215
29925	1000	0.400	0.900	10.0–30.0	22.7107	0.7227	0.3330
53848	2500	0.400	0.400	50.0–150.0	22.7218	0.7367	0.2879
55346	2500	0.700	0.700	50.0–150.0	22.9966	0.7449	0.2642
57244	2500	1.000	0.900	50.0–150.0	22.9993	0.7446	0.2537
57361	2500	0.700	0.700	50.0–150.0	23.0504	0.7489	0.2529
57457	2500	0.400	0.700	50.0–150.0	23.1356	0.7511	0.2563
59105	2500	0.400	0.700	30.0–90.0	23.2408	0.7546	0.2499
64424	2500	0.400	0.700	10.0–30.0	23.2947	0.7564	0.2434
67394	2500	0.700	0.900	50.0–150.0	23.3206	0.7574	0.2454
68840	2500	0.700	0.900	30.0–90.0	23.3434	0.7617	0.2318
68973	2500	0.700	0.900	30.0–90.0	23.3798	0.7603	0.2371
69778	2500	0.400	0.900	50.0–150.0	23.3858	0.7581	0.2465
71137	2500	0.400	0.900	50.0–150.0	23.3968	0.7592	0.2493
73926	2500	0.400	0.900	30.0–90.0	23.4803	0.7630	0.2418
110372	5000	0.400	0.700	30.0–90.0	23.5677	0.7730	0.2093
121268	5000	0.400	0.700	10.0–30.0	23.6687	0.7799	0.1984
125953	5000	0.400	0.900	50.0–150.0	23.7238	0.7755	0.2140
129314	5000	0.400	0.900	50.0–150.0	23.9123	0.7809	0.2081
146662	5000	0.400	0.900	10.0–30.0	23.9234	0.7848	0.1953
177742	7500	0.400	0.900	50.0–150.0	23.9345	0.7854	0.1979
189863	7500	0.400	0.900	30.0–90.0	24.0462	0.7931	0.1821
224673	10000	0.400	0.900	50.0–150.0	24.0664	0.7896	0.1827
261207	10000	0.400	0.900	10.0–30.0	24.1479	0.7987	0.1655
440396	25000	0.400	0.900	50.0–150.0	24.1782	0.7976	0.1680
566515	25000	0.400	0.900	10.0–30.0	24.3129	0.8121	0.1380

Table B.2: Pareto-optimal runs for hkust_campus

Gaussians	Max Gaussians	Opacity	Coverage	RGB Canny	Novel Metrics		
					PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
8086	1000	1.000	0.400	50.0–150.0	22.2227	0.7183	0.4005
8672	1000	1.000	0.400	50.0–150.0	22.3241	0.7196	0.3953
8907	1000	0.700	0.400	50.0–150.0	22.8926	0.7313	0.3889
12806	1000	1.000	0.700	50.0–150.0	22.9221	0.7282	0.3638
17624	1000	0.400	0.700	50.0–150.0	23.4636	0.7462	0.3457
17819	1000	1.000	0.900	50.0–150.0	23.5883	0.7471	0.3206
20329	1000	0.400	0.700	30.0–90.0	23.6243	0.7511	0.3372
21575	2500	0.400	0.400	50.0–150.0	23.6598	0.7600	0.3122
21939	1000	0.400	0.700	10.0–30.0	23.6809	0.7562	0.3265
22864	2500	0.400	0.400	30.0–90.0	23.6895	0.7621	0.3050
23296	2500	0.400	0.400	30.0–90.0	23.7073	0.7625	0.3056
23390	2500	0.700	0.400	30.0–90.0	23.9852	0.7679	0.2917
27896	1000	0.700	0.900	30.0–90.0	24.0826	0.7736	0.2811
29929	1000	0.400	0.900	10.0–30.0	24.1530	0.7793	0.2771
31095	1000	0.400	0.900	50.0–150.0	24.3760	0.7768	0.2818
36362	2500	0.700	0.700	50.0–150.0	24.4087	0.7814	0.2650
39377	2500	0.400	0.700	30.0–90.0	24.4622	0.7836	0.2517
48178	2500	0.400	0.700	30.0–90.0	24.5246	0.7855	0.2443
55659	5000	1.000	0.900	50.0–150.0	24.5479	0.7888	0.2230
56248	5000	0.700	0.700	50.0–150.0	24.7020	0.7940	0.2240
61887	5000	0.700	0.700	50.0–150.0	24.7412	0.7936	0.2276
63083	10000	0.700	0.400	50.0–150.0	24.7504	0.7951	0.2206
63870	2500	0.700	0.900	50.0–150.0	24.9772	0.7973	0.2265
65968	2500	0.700	0.900	10.0–30.0	25.1163	0.8041	0.2004
78837	2500	0.400	0.900	10.0–30.0	25.1253	0.8066	0.1953
128328	5000	0.700	0.900	10.0–30.0	25.2472	0.8089	0.1766
133357	5000	0.700	0.900	30.0–90.0	25.5144	0.8180	0.1699
153139	5000	0.400	0.900	10.0–30.0	25.6580	0.8246	0.1601
225550	7500	0.400	0.900	30.0–90.0	25.7171	0.8233	0.1620
265889	10000	0.400	0.900	10.0–30.0	25.8104	0.8259	0.1462
448803	100000	0.700	0.900	30.0–90.0	25.9756	0.8303	0.1267
846990	100000	0.400	0.900	10.0–30.0	26.1325	0.8347	0.1047
910431	100000	0.400	0.900	10.0–30.0	26.2226	0.8451	0.1060

Table B.3: Pareto-optimal runs for hku_park

Gaussians	Max Gaussians	Opacity	Coverage	RGB Canny	Novel Metrics		
					PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
6637	1000	1.000	0.400	30.0–90.0	19.4951	0.5395	0.4689
6814	1000	1.000	0.400	30.0–90.0	19.6970	0.5576	0.4647
6817	1000	1.000	0.400	50.0–150.0	19.7392	0.5603	0.4611
7158	1000	1.000	0.400	10.0–30.0	19.8082	0.5603	0.4549
7368	1000	0.700	0.400	50.0–150.0	20.0983	0.5768	0.4497
8448	1000	0.400	0.400	50.0–150.0	20.2051	0.5847	0.4354
8919	1000	0.700	0.400	50.0–150.0	20.2593	0.5899	0.4367
9412	1000	0.400	0.400	30.0–90.0	20.2669	0.5870	0.4334
14769	1000	1.000	0.700	30.0–90.0	20.9397	0.6247	0.3547
15273	1000	1.000	0.700	50.0–150.0	21.0885	0.6295	0.3605
15482	1000	1.000	0.900	10.0–30.0	22.1439	0.6730	0.2936
20810	1000	0.700	0.700	10.0–30.0	22.3988	0.6795	0.3013
23580	1000	0.400	0.700	10.0–30.0	22.6429	0.6854	0.2992

(continued on next page)

(continued from previous page)

Gaussians	Max Gaussians	Opacity	Coverage	RGB Canny	Novel Metrics		
					PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
38663	2500	1.000	0.900	10.0–30.0	23.0897	0.7157	0.2050
48675	2500	0.700	0.700	50.0–150.0	23.1591	0.7144	0.2330
55460	2500	0.400	0.700	30.0–90.0	23.2473	0.7202	0.2256
55522	2500	0.400	0.700	10.0–30.0	23.6554	0.7303	0.2132
105187	5000	0.400	0.700	10.0–30.0	24.1234	0.7455	0.1643
145615	10000	0.700	0.700	50.0–150.0	24.3482	0.7439	0.1553
215772	50000	0.400	0.400	50.0–150.0	24.6847	0.7490	0.1430
271587	25000	0.400	0.700	50.0–150.0	25.0173	0.7544	0.1339

Table B.4: Pareto-optimal runs for degenerate.seq

Gaussians	Max Gaussians	Opacity	Coverage	RGB Canny	Novel Metrics		
					PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
17016	1000	1.000	0.400	50.0–150.0	20.6959	0.7216	0.3304
17811	1000	1.000	0.400	50.0–150.0	20.7870	0.7195	0.3298
18974	1000	1.000	0.700	50.0–150.0	21.0079	0.7348	0.3024
20815	1000	0.400	0.400	50.0–150.0	21.0316	0.7306	0.3253
20938	1000	1.000	0.900	50.0–150.0	21.6373	0.7479	0.2678
24262	1000	0.400	0.700	50.0–150.0	21.6417	0.7465	0.2859
24740	1000	0.400	0.700	30.0–90.0	21.6782	0.7515	0.2816
25713	1000	0.700	0.900	30.0–90.0	21.8493	0.7621	0.2571
25992	1000	0.700	0.900	50.0–150.0	21.9798	0.7623	0.2513
29006	1000	0.400	0.900	50.0–150.0	22.0337	0.7608	0.2646
46615	2500	0.700	0.700	50.0–150.0	22.0486	0.7690	0.2249
47745	2500	1.000	0.900	50.0–150.0	22.0565	0.7687	0.2218
50310	2500	0.700	0.700	50.0–150.0	22.0733	0.7659	0.2323
51460	2500	0.400	0.700	50.0–150.0	22.1913	0.7693	0.2260
55189	2500	0.400	0.700	50.0–150.0	22.2421	0.7726	0.2247
57313	2500	0.700	0.900	50.0–150.0	22.2759	0.7783	0.2051
60591	2500	0.700	0.900	50.0–150.0	22.3792	0.7834	0.2038
63514	2500	0.700	0.900	30.0–90.0	22.5149	0.7860	0.1965
70080	2500	0.400	0.900	30.0–90.0	22.6400	0.7890	0.1978
71562	2500	0.400	0.900	50.0–150.0	22.7294	0.7858	0.1974
74397	2500	0.400	0.900	10.0–30.0	22.8480	0.7935	0.1828
117578	25000	0.700	0.700	50.0–150.0	22.9691	0.7958	0.1862
132663	25000	0.400	0.700	50.0–150.0	23.0175	0.7977	0.1839
144252	5000	0.400	0.900	10.0–30.0	23.2367	0.8146	0.1507
145643	5000	0.400	0.900	10.0–30.0	23.2505	0.8118	0.1475
209109	7500	0.400	0.900	10.0–30.0	23.3036	0.8199	0.1382
210445	7500	0.400	0.900	10.0–30.0	23.3113	0.8190	0.1396
248991	25000	0.400	0.900	50.0–150.0	23.4056	0.8123	0.1609
264661	10000	0.400	0.900	10.0–30.0	23.5419	0.8232	0.1289
391576	100000	0.400	0.900	30.0–90.0	23.6265	0.8263	0.1382
556100	25000	0.400	0.900	10.0–30.0	23.7402	0.8318	0.1229
725518	50000	0.400	0.900	10.0–30.0	23.7405	0.8339	0.1171
757890	100000	0.400	0.900	10.0–30.0	23.8347	0.8370	0.1148



Comparison Runs

Table C.1: Comparison of Gaussian-LIC2 vs PM Runs - hku_campus_seq

Gaussians		PSNR (dB) $\uparrow$		SSIM $\uparrow$		LPIPS $\downarrow$	
G-LIC2	PM	G-LIC2	PM	G-LIC2	PM	G-LIC2	PM
36k	29k	22.5333	22.7107	0.7416	0.7227	0.2668	0.3330
60k	59k	22.9150	23.2408	0.7542	0.7546	0.2403	0.2499
102k	73k	23.2767	23.4803	0.7688	0.7630	0.2172	0.2418
117k	110k	23.3653	23.5677	0.7736	0.7730	0.2063	0.2093
162k	146k	23.6177	23.9234	0.7811	0.7848	0.1933	0.1953
175k	146k	23.6693	23.9234	0.7847	0.7848	0.1905	0.1953
186k	177k	23.7148	23.9345	0.7858	0.7854	0.1881	0.1979
242k	224k	23.7993	24.0664	0.7899	0.7896	0.1824	0.1827
258k	224k	23.9903	24.0664	0.7945	0.7896	0.1812	0.1827
280k	261k	23.9283	24.1479	0.7943	0.7987	0.1787	0.1655

Table C.2: Comparison of Gaussian-LIC2 vs PM Runs - degenerate_seq

Gaussians		PSNR (dB) $\uparrow$		SSIM $\uparrow$		LPIPS $\downarrow$	
G-LIC2	PM	G-LIC2	PM	G-LIC2	PM	G-LIC2	PM
32k	29k	21.6747	22.0337	0.7666	0.7608	0.2227	0.2646
65k	63k	22.2825	22.5149	0.7880	0.7860	0.1845	0.1965
93k	74k	22.4789	22.8480	0.7925	0.7935	0.1761	0.1828
124k	117k	22.5761	22.9691	0.7979	0.7958	0.1654	0.1862
155k	145k	22.8361	23.2505	0.8064	0.8118	0.1569	0.1475
188k	145k	22.8019	23.2505	0.8085	0.8118	0.1573	0.1475
212k	210k	23.0881	23.3113	0.8132	0.8190	0.1517	0.1396
241k	210k	23.0349	23.3113	0.8154	0.8190	0.1444	0.1396
263k	248k	23.2311	23.4056	0.8160	0.8123	0.1463	0.1609
296k	264k	23.2432	23.5419	0.8197	0.8232	0.1429	0.1289

Table C.3: Comparison of Gaussian-LIC2 vs PM Runs - hkust_campus

Gaussians		PSNR (dB) $\uparrow$		SSIM $\uparrow$		LPIPS $\downarrow$	
G-LIC2	PM	G-LIC2	PM	G-LIC2	PM	G-LIC2	PM
11k	8k	23.6622	22.8926	0.7692	0.7313	0.2797	0.3889
23k	23k	24.3896	23.7073	0.7977	0.7625	0.2205	0.3056
43k	39k	24.7147	24.4622	0.8088	0.7836	0.1846	0.2517
63k	63k	25.1331	24.7504	0.8198	0.7951	0.1659	0.2206
85k	78k	24.9208	25.1253	0.8177	0.8066	0.1593	0.1953
105k	78k	25.0930	25.1253	0.8238	0.8066	0.1496	0.1953
119k	78k	25.4010	25.1253	0.8270	0.8066	0.1509	0.1953
135k	133k	25.4940	25.5144	0.8251	0.8180	0.1428	0.1699
158k	153k	25.3912	25.6580	0.8273	0.8246	0.1404	0.1601
172k	153k	25.3745	25.6580	0.8317	0.8246	0.1373	0.1601
185k	153k	25.5731	25.6580	0.8336	0.8246	0.1358	0.1601

Table C.4: Comparison of Gaussian-LIC2 vs PM Runs - hku_park

Gaussians		PSNR (dB) $\uparrow$		SSIM $\uparrow$		LPIPS $\downarrow$	
G-LIC2	PM	G-LIC2	PM	G-LIC2	PM	G-LIC2	PM
20k	15k	22.0250	22.1439	0.6867	0.6730	0.2602	0.2936
40k	38k	23.0794	23.0897	0.7239	0.7157	0.2052	0.2050
80k	55k	22.6698	23.6554	0.7178	0.7303	0.1899	0.2132
118k	105k	23.6441	24.1234	0.7395	0.7455	0.1641	0.1643
153k	145k	23.5392	24.3482	0.7424	0.7439	0.1658	0.1553
186k	145k	24.3656	24.3482	0.7538	0.7439	0.1511	0.1553
220k	215k	23.3305	24.6847	0.7338	0.7490	0.1690	0.1430
253k	215k	23.9877	24.6847	0.7431	0.7490	0.1525	0.1430
283k	271k	24.2656	25.0173	0.7548	0.7544	0.1518	0.1339
313k	271k	23.1228	25.0173	0.7304	0.7544	0.1660	0.1339
341k	271k	24.5027	25.0173	0.7542	0.7544	0.1459	0.1339

