Point2Gaussian: Point-Cloud-to-Gaussian Conversion for Efficient 3D Scene Rendering

Supplementary Material

A. Additional Implementation Details

A.1. Details of Virtual Camera Placement

In Sec. 3.2 of the main text, we described our method for placing virtual cameras on two concentric spheres. The radii of these spheres are determined by the diagonal length of the axis-aligned bounding box of the point cloud part $\mathcal{P}_i$, denoted as $\mathrm{diag}(\mathcal{P}_i)$. Here, we provide the detailed mathematical formulation for this calculation.

First, the point cloud part $\mathcal{P}_i$ consists of N_i points. Each point p_n is represented by its 3D coordinates:

$$p_n := (x_n, y_n, z_n) \in \mathbb{R}^3 \quad (n = 1, 2, \ldots, N_i). \tag{2}$$

Next, we compute the side lengths of the axis-aligned bounding box that encloses $\mathcal{P}_i$. These lengths, L_x, L_y, L_z, are calculated as the range of the coordinates along each axis:

$$L_x := \max\{x_1, \ldots, x_{N_i}\} - \min\{x_1, \ldots, x_{N_i}\}, \tag{3}$$
$$L_y := \max\{y_1, \ldots, y_{N_i}\} - \min\{y_1, \ldots, y_{N_i}\}, \tag{4}$$
$$L_z := \max\{z_1, \ldots, z_{N_i}\} - \min\{z_1, \ldots, z_{N_i}\}. \tag{5}$$

Finally, the diagonal length of this bounding box, $\mathrm{diag}(\mathcal{P}_i)$, is computed using the Pythagorean theorem in three dimensions:

$$\mathrm{diag}(\mathcal{P}_i) = \sqrt{L_x^2 + L_y^2 + L_z^2}. \tag{6}$$

This value provides a robust measure of the overall size of the point cloud part, which we then use to adaptively set the camera distances as described in the main text.

B. Supplementary Experimental Analysis

B.1. Detailed Geometry Accuracy Analysis

In the main text (Table 2), we reported the Mean Absolute Error (MAE) for our method and several baselines. To offer a more granular comparative view, Figure 8 presents the cumulative distribution function (CDF) of the absolute depth error.

As illustrated, the performance curve of our method closely tracks that of the state-of-the-art method, Pulsar, and both methods outperform other baselines across the entire error spectrum. Specifically, for our method, 81.7% of pixels exhibit an error below 0.01 m, and 96.5% are below 0.05 m.

This result confirms that the low MAE is not an artifact of averaging but stems from the consistently high accuracy across the vast majority of pixels. It further substantiates our claim that the Point2Gaussian conversion robustly preserves the geometric integrity of the original scene.

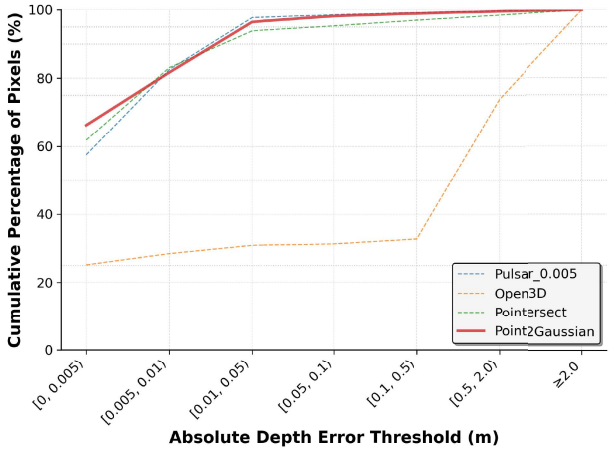

Figure 8. CDF of Absolute Depth Error.

B.2. Additional Analysis on the Indoor LiDAR-RGBD Dataset

As stated in the main text, a formal quantitative evaluation on the Indoor LiDAR-RGBD Dataset [20] is not feasible due to the lack of perfectly aligned ground-truth camera poses. However, to provide further numerical insight that complements the qualitative comparisons, we computed visual quality metrics for reference purposes.

Reference Metrics Calculation. We computed PSNR, SSIM, and LPIPS between rendered images and the RGB images from the dataset's video stream (captured with an Asus Xtion Live camera, using every 100th frame). It must be strongly emphasized that, due to the aforementioned camera pose inaccuracies, these metrics do not represent an absolute measure of quality. Instead, they serve as a numerical proxy to help interpret the visual artifacts and differences seen in our qualitative results (Figure 9). For instance, a significantly lower SSIM score can numerically reflect severe structural artifacts that are visually apparent.

Observations. The calculated reference metrics are presented in Table 5. The values align with our qualitative findings: Our method, Point2Gaussian, shows noticeably better metrics compared to the baselines. This numerically corresponds to the visually more coherent and artifact-free images it produces, as shown in Figure 9. Conversely, the

Table 5. Reference metrics and performance statistics for different rendering methods on the indoor dataset. Note: Due to imprecise camera poses, image metrics (PSNR, SSIM, LPIPS) are for comparative reference only and do not reflect absolute ground-truth accuracy.

Rendering Method	PSNR↑	SSIM↑	LPIPS↓	DataSize↓	Total Primitives↓	FPS↑
Pulsar_0.001	11.55	0.197	0.673	1.58GB	3.33×10^7	2.59
Open3D	8.85	0.071	0.774			7.87
Point2Gaussian	**13.27**	**0.544**	**0.564**	**251MB**	$\mathbf{3.87 \times 10^6}$	**189.17**

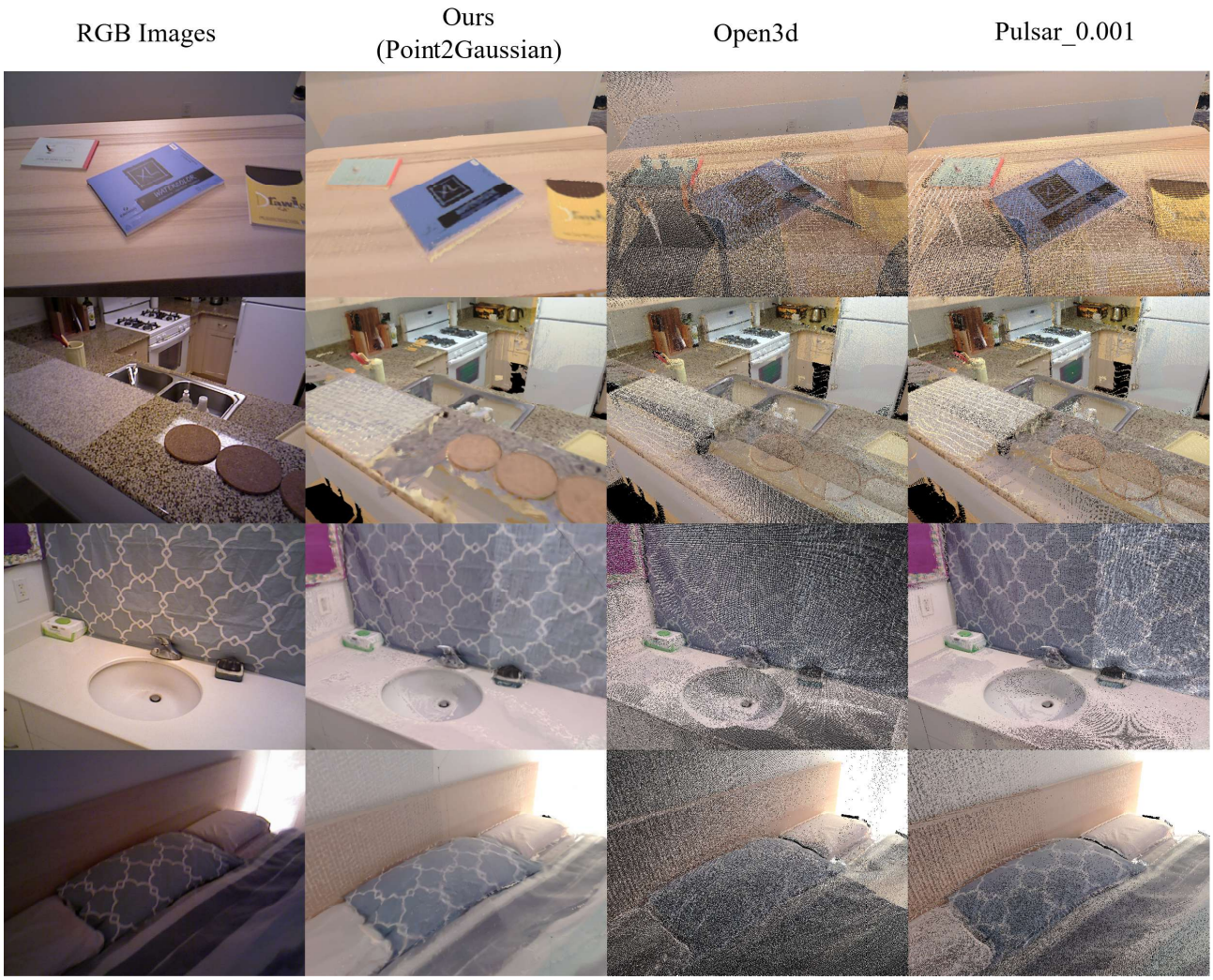

Figure 9. Additional qualitative comparison of rendering results across different rendering methods on Indoor LiDAR-RGBD Dataset.

very low scores for Open3D are consistent with the sparse and disconnected appearance in its rendered images. These numbers, while imperfect, support the conclusion from our qualitative analysis that Point2Gaussian generates results more similar to the real images. The table also includes data size and rendering speed, further highlighting our method's efficiency.

C. Additional Ablation Study

C.1. Ablation on Point Cloud Splitting Strategy

In addition to the primary ablation studies presented in the main text, we also evaluated the foundational step of our pipeline: the strategy for partitioning the initial scene point cloud. A proper splitting method is crucial, as it directly

Table 6. Ablation study on the point cloud splitting strategy.

Splitting Strategy	PSNR↑	SSIM↑	LPIPS↓	DataSize↓
Clustering-based Splitting	**22.75**	**0.81**	**0.21**	316MB
Grid-based Splitting	21.85	0.76	0.29	**209MB**

Figure 10. Qualitative comparison of point cloud splitting strategies.

impacts how parts are isolated for conversion, affecting occlusion and the final reconstruction quality.

We compared two strategies:

- **Clustering-based Splitting:** We partition the original point cloud using a clustering method, analogous to instance segmentation. This allows each object-like cluster to be processed individually, minimizing occlusion from other objects during the rendering of training views.
- **Grid-based Splitting:** As a baseline, we also evaluated a simple grid-based approach. The scene is divided into uniform cells (5m edge length in our experiments), and points within each cell are converted independently.

Table 6 compares the two strategies using image-quality metrics and file size. The clustering approach consistently delivers superior reconstruction quality across all metrics. This is because clustering intelligently separates objects, preventing details of one object from being occluded by another during the view-generation process. In contrast, the grid-based method is prone to such occlusion, causing some structures to be missed during conversion, which, while reducing the final file size, significantly degrades quality.

This numerical difference is visually explained by the qualitative results shown in Figure 10. The Grid-based method (left) results in a blurry model with significant detail loss. This is caused by internal occlusion, where large surfaces like the ground occlude important object details within the same arbitrary grid cell during training view generation. The Clustering-based method (right) avoids this issue by processing coherent objects individually, thus preserving fine details and achieving a much sharper reconstruction.

This analysis validates our choice of using a clustering-based method for point cloud partitioning as the default strategy throughout all other experiments.

D. Additional Discussion

D.1. Comparison with feed-forward methods

While feed-forward models offer rapid inference, and some methods can also convert point clouds to Gaussians, these methods share three critical limitations for industrial applications compared to our approach:

Primitive Efficiency: Feed-forward methods typically enforce a 1:1 input-to-primitive mapping (or even generate more points via splitting). For massive LiDAR scans, this results in an explosion of Gaussian primitives, significantly increasing storage and computational demands. In contrast, our method performs significant primitive decimation, representing the scene with dozens of times fewer Gaussians than the input points, as shown in Table 1. This is the key to achieving our compact storage and real-time rendering.

Zero-Shot Generalization: Learning-based regressors often suffer from domain gaps between different datasets (e.g., trained on ShapeNet objects, failing on outdoor scenes). Our method requires no additional training data for different scenes or categories, ensuring consistent performance across arbitrary unseen datasets.

Input Limits: Due to heavy memory overheads, feed-forward methods are typically limited to 100K points per pass. Processing our substation dataset (3.18×10^8 points) forces a fatal compromise: either (1) Extreme Fragmentation: Splitting the scene into thousands of tiny patches (vs. our 60 parts) disrupts the local geometric context and creates boundary artifacts; or (2) Aggressive Downsampling: Reducing point density to fit memory limits directly results in the loss of critical fine-grained information.

D.2. Process time of offline process

For the 4.4GB substation dataset, total processing on a single NVIDIA RTX 4000 Ada GPU is 285h (277h Pointersect + 7.7h 3DGS). We emphasize this is a one-time offline investment yielding a permanent asset for $>$80 FPS real-time inspection. Our priority is high-fidelity visualization rather than immediate data delivery. While accelerating the renderer itself was out of scope, the bottleneck (Pointersect) is embarrassingly parallel. In practical deployment, distributing the workload across N GPUs linearly reduces rendering time by a factor of 1/N. This ensures scalability for industrial applications where fidelity outweighs setup time.