Evaluation of Camera Topologies on Output Quality in Gaussian Splatting

Florián Jurík, Marek Vančo, Šimon Tibenský
Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava, Slovakia
jurik.florian@gmail.com, marek_vanco@stuba.sk, simon.tibensky@stuba.sk

Abstract - **Digital life has become an everyday reality, and life without it is almost unimaginable. Digitalization has significantly advanced humanity, improving quality of life, extending lifespans through medical innovations, and offering entertainment such as watching films or playing video games. This paper examines the digitalization of physical objects and environments, using various photo-based algorithms and digital model creation software.**

Keywords **– Gaussian Splatting, PSNR, Camera topology**

I. Introduction

The digital reconstruction of physical environments and objects has a long-standing presence in computer graphics and 3D modeling. In the early days, digital models were manually created using specialized software, which required time, precision, and expert knowledge. Over time, the field has evolved significantly, giving rise to more automated and intelligent methods.

One widely adopted approach is photogrammetry, which generates 3D models from a series of photographs taken around a real-world object or space. This technique relies on identifying and triangulating common features across multiple images to build a spatially accurate representation. While effective, photogrammetry can produce incomplete or distorted results in challenging lighting or texture conditions, often requiring manual correction.

A significant shift occurred with the emergence of Neural Radiance Fields (NeRF) in 2020. NeRF leverages neural networks to create photorealistic 3D scenes from 2D images, capturing complex visual effects such as reflections, shadows, and fine textures with impressive accuracy. This advancement triggered a surge in AI-driven 3D reconstruction research, leading to new tools and techniques [1].

Among the latest developments is Gaussian Splatting, a method that represents 3D scenes using a cloud of Gaussian primitives. Unlike mesh-based or voxel-based systems, Gaussian Splatting offers a real-time, high-fidelity rendering pipeline, balancing quality and performance. It has quickly gained attention for its effectiveness in rendering dynamic and detailed environments.

This paper focuses on comparing various image-based modeling algorithms applied to the reconstruction of a single indoor room. The comparison is performed using the Gaussian Classic software, which provides a practical platform for evaluating the strengths and limitations of each approach. The goal is not only to observe the differences in the generated models but also to assess their efficiency and visual fidelity. A more detailed analysis of the methods and results will be presented in the following chapters.

II. Gaussian Splatting Technique

Gaussian Splatting begins with digital photographs of a scene or object as the primary input. After capturing and uploading the images to a computer, the first essential step is the application of the Structure from Motion (SfM) technique. SfM analyzes the image set, generates a point cloud, and accurately estimates the positions of the cameras from which the photos were taken.

Gaussians are represented as three-dimensional ellipsoids, with each Gaussian described by several parameters:

- Position – the 3D coordinates (x, y, z) of the point in space.
- Covariance – mathematically defines the size, shape, and orientation of the ellipsoid.
- Color – defined using RGB values, determining the appearance of each point.
- Opacity – controls the transparency or opacity of the Gaussian.
- Scale – describes the physical size of the ellipsoid in 3D space.

The total number of Gaussians depends largely on the complexity of the scene or object being modeled, often reaching millions of points in detailed reconstructions. Initially, during the early stages of training, these Gaussians appear as large, rough ellipsoids and do not yet provide a detailed representation of the scene.

To improve model quality, the system enters a training phase based on feedback optimization. During this phase, the current 3D representation is projected back into the 2D image space and compared against the original photographs. The software then identifies discrepancies between the rendered view and the actual photos. Using this feedback, the system adjusts the parameters of the Gaussians, gradually enhancing the detail and realism of the reconstructed scene.

The optimization process involves two key strategies:

- Underrepresented areas – If a region lacks sufficient detail, the software duplicates the corresponding Gaussian and introduces more points to better represent the details.

- Overrepresented areas – If a region is overly dense with Gaussians, the software splits or reduces the influence of those Gaussians to increase accuracy and reduce visual redundancy.

This iterative training process occurs over many cycles. In each iteration, the parameters of the Gaussians are refined to bring the model closer to a photorealistic result. The number of iterations required varies with the complexity of the scene. However, on average, around 30,000 iterations are needed to achieve convergence—the point at which further iterations no longer result in significant visual improvements [2].

III. Camera Layout Algorithms, Dataset

Accurate and efficient camera placement is crucial for high-quality 3D reconstruction using image-based methods. We explore five distinct camera placement strategies employed in the study: the Hemisphere Algorithm, Random Placement, Original Algorithm, an algorithm made by a colleague.

The Hemisphere Algorithm involves positioning cameras uniformly along a hemispherical surface within the room. This setup ensures comprehensive coverage from above, capturing the scene from a wide range of angles. It's particularly effective for scenes where overhead views provide significant structural information.

Original Algorithm proposes an algorithm that efficiently and progressively suggests the next best camera placement to maximize reconstruction quality. They introduce two key metrics, computed on a discretized 3D grid within a user-defined bounding box:

1) Observation Frequency: How often a point in the volume is observed by the current set of cameras.

2) Angular Uniformity: For each point, this measures how uniformly the viewing directions of observing cameras are distributed, compared to an ideal uniform angular distribution. It's calculated using the Total Variation distance between the empirical and uniform distributions.

A reconstruction quality score is defined by combining these two metrics for all points in the discretized volume. The goal is to maximize this score.

The greedy algorithm helps to bring it together. It samples a set of candidate camera poses in free space. It evaluates which candidate camera, if added to the existing set, would yield the largest increase in the reconstruction quality score. The "best" camera is chosen, an image is captured, and the dataset is updated. This process repeats until a camera budget is met [3].

In the Random Placement strategy, cameras are distributed randomly throughout the room. This method introduces variability in viewpoints, which can be beneficial for capturing diverse perspectives. However, the lack of structured placement may lead to uneven coverage and potential gaps in the reconstructed model.

The Tibonachi algorithm is based on the Original Algorithm and adds the following improvements to its operation:

- Coroutine support to spread computations across multiple frames for smooth real-time performance
- Configurable additional observation points to further boost spatial coverage.
- Partial grid sampling to reduce computational overhead
- Strict enforcement of minimum separation between points

These enhancements yield faster convergence, more balanced view distributions, and improved 3D reconstruction quality.

In the study, a total of 36 datasets were utilized to evaluate the performance of different camera placement algorithms in 3D scene reconstruction. The datasets were distributed evenly across the five algorithms, with each algorithm tested using nine distinct datasets. These datasets were categorized based on the number of input photographs used for reconstruction. The image count increased progressively, starting from 20 photos and increasing by increments of 10, up to 100 photos. This systematic grouping—ranging from sparse to dense image coverage—allowed for consistent performance comparison and detailed analysis of how each algorithm responds to varying levels of visual information.

All datasets were generated using virtual camera placements within a digitally created 3D room designed in a computer environment. This approach allowed precise control over camera positions and scene conditions, ensuring consistency across all tests. The synthetic nature of the room ensured reproducibility and eliminated real-world noise, making it ideal for isolating and analyzing the performance of the camera placement algorithms.

IV. Prerequisites

The developers of Gaussian Splatting have provided an application that greatly simplifies the process of generating 3D models. However, using this tool requires several pre-installed components and meeting specific hardware and software requirements.

Hardware Requirements are a GPU with CUDA support and a compute capability of 7.0 or higher ; at least 24 GB of VRAM.

Software Requirements are a C++ compiler compatible with PyTorch extensions; CUDA SDK 11, required for building PyTorch extensions; compatibility between the C++ compiler and the installed CUDA SDK; Conda, which is not mandatory but highly recommended as it simplifies the environment setup.

The application runs entirely through a command-line interface (CLI). However, once a model has been generated, it can be viewed in a dedicated viewer built on the SIBR framework. The latest major update to the software was released in October 2024, and the full application is available for download on GitHub for free [4].

V. Workflow

After the all the necessary components are successfully installed for Gaussian Splatting and the photographs have already been transferred to the computer, the model making process can begin.

Inside the gaussian-splatting directory—which contains several files and subfolders—only certain items are relevant for this workflow. Inside the data directory, create a new folder and within it, create a subfolder named input. Place all the captured photographs into this input subfolder.

Run Anaconda Prompt and enter the following commands:

1) conda activate gaussian_splatting - This command activates the gaussian_splatting Conda environment, ensuring that Python and all required dependencies are available.

2) cd gaussian-splatting - changes the working directory to the root folder of the Gaussian Splatting project.

3) python convert.py -s <path> - processes the input photographs and converts them into a format suitable for model generation. The duration of this step depends on the number and quality of the photos but typically takes less than 5 minutes.

4) python train.py -s <path> -m <path> --data_device cpu --iterations 30000 - This command starts the training process and performs multiple operations simultaneously:

- ***-s <path> -*** specifies the input folder with photographs
- ***-m <path> -*** sets the output folder where the final model will be saved. If not specified, the model will be saved under a randomly generated name in the output folder
- ***--data_device cpu -*** forces the training to run on the CPU (optional if a GPU is available)
- ***--iterations 30000 -*** defines the number of training iterations

The total training time depends on the number of iterations and the size of the input data. For 30,000 iterations, the process typically takes about 8 minutes. If a higher number of iterations is used, the program automatically creates checkpoints during training—saving intermediate versions of the model every 25% of the total iterations. For instance, with 30,000 iterations, a checkpoint at 7,000 iterations will also be available.

VI. Evaluation

There are multiple ways to evaluate the quality of a 3D model. While visual inspection can often reveal which model appears more realistic, a professional and objective comparison requires the use of standardized evaluation metrics. These metrics can be computed once the model generation process is complete.

One of the fundamental metrics related to accuracy is the Mean Squared Error (MSE). Although MSE is not typically reported as a final evaluation value, it serves as the basis for several other important quality indicators. An MSE value of zero indicates perfect prediction accuracy, while higher values reflect a greater discrepancy between predicted and reference data. MSE is calculated using the following formula:

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n} \quad (1)$$

Where *yi* is true value, *ŷi* is the predicted value, and *n* is the number of data points. Squaring the differences ensures that all errors are positive, making the MSE value always non-negative [5].

A widely used metric is the Peak Signal-to-Noise Ratio (PSNR), which assesses the fidelity of an image by comparing the original input image to the reconstructed or processed version. PSNR measures the level of noise or error between the two images and is expressed in decibels (dB). A higher PSNR value indicates that the processed image is more similar to the original and thus of higher quality. Importantly, the original and processed images must be of the same resolution for PSNR to be computed [6]. The PSNR formula is as follows:

$$PSNR = 10\, log_{10} \frac{(L-1)^2}{\frac{1}{N^2}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1}(\hat{I}(r,c) - I(r,c))^2} \quad (2)$$

Where L is the maximum possible pixel value (for 8-bit images, L=256), N x N is the total number of pixels in the image, $I(r, c)$ is the original image pixel value at row r and column c, $\hat{I}(r, c)$ is the corresponding pixel value in the processed image.

The following tables are organized by the algorithms. Each test was run with 30,000 iterations. In the row beneath the title, can be found the number of images used in each dataset. The next row shows the number of cameras accepted by the program—these are the cameras that were successfully connected and used to generate the digital model. The final row presents the evaluation results of each model, represented by the corresponding PSNR values. In some places in the table, dashes appear. These indicate that the program was unable to align the images, and therefore could not generate a digital model from them.

TABLE I. Evaluation of the Datasets

Hemisphere									
Input images	***20 pic.***	***30 pic.***	***40 pic.***	***50 pic.***	***60 pic.***	***70 pic.***	***80 pic.***	***90 pic.***	***100 pic***
Detected cameras	18	30	40	50	60	70	80	90	100
PSNR [dB]	25.152	27.019	28.888	28.575	29.653	30.868	30.636	33.377	33.287
Original Algorithm									
Input images	***20 pic.***	***30 pic.***	***40 pic.***	***50 pic.***	***60 pic.***	***70 pic.***	***80 pic.***	***90 pic.***	***100 pic***
Detected cameras	12	27	34	43	43	62	59	76	85
PSNR[dB]	23.702	26.398	27.588	28.284	29.365	29.728	30.181	30.460	30.595

Random									
Input images	***20 pic.***	***30 pic.***	***40 pic.***	***50 pic.***	***60 pic.***	***70 pic.***	***80 pic.***	***90 pic.***	***100 pic***
Detected cameras	-	-	-	-	14	21	54	63	53
PSNR [dB]	-	-	-	-	23.430	26.681	26.032	24.659	27.177
Tibonachi									
Input images	***20 pic.***	***30 pic.***	***40 pic.***	***50 pic.***	***60 pic.***	***70 pic.***	***80 pic.***	***90 pic.***	***100 pic***
Detected cameras	16	21	24	31	36	47	51	58	69
PSNR [dB]	24.991	25.396	26.739	27.811	29.487	28.015	27.580	27.022	29.298

VII. Observations

In nearly all algorithms, increasing the number of input images generally leads to a higher PSNR, suggesting improved model quality with more data.

The Hemisphere dataset consistently achieves the highest PSNR values across nearly all picture counts. It also exhibits a very high camera acceptance rate, often utilizing 100% or close to 100% of the input pictures

The Original Algorithm generally performs better than Random and Tibonachi, but consistently lags the Hemisphere dataset in PSNR. Its camera acceptance rate is moderate, lower than Hemisphere.

The Random algorithm performs significantly worse than the others, particularly in the lower image-count datasets where it fails to connect any cameras at all (20–50 images). Even in the higher datasets, its PSNR values remain consistently lower, reflecting the importance of structured camera placement in achieving high model quality.

The Tibonachi algorithm shows very strong early performance, outperforming even Hemisphere at 20, 30, and 40 images. Its performance increases steadily and peaks at 29.487 (60 images), after which it stabilizes with mild fluctuations. This makes it a viable choice when dealing with fewer images, due to its efficient camera distribution.

There is a clear correlation between the number of cameras successfully used and the PSNR results. Algorithms like Hemisphere and Original maintain a 1:1 camera-to-image ratio, or close to it, while Random suffers from lower camera utilization, impacting its output quality.

VIII. Future Work

In the future, this study could be expanded by including additional camera placement algorithms that were not covered in this paper. The scope of the research can also be broadened in other directions. For instance, beyond Gaussian Splatting, similar 3D reconstruction technologies could be incorporated and compared. Another possibility is to test alternative software solutions specifically designed for generating Gaussian Splatting models, which may offer different features, performance, or quality. These extensions could provide a more comprehensive understanding of the strengths and limitations of various approaches in photogrammetric 3D modeling.

Acknowledgment

This paper was supported by DISIC (09I05-03-V2), NEXT (ERASMUS-EDU-2023-CBHE-STRAND-2), EULiST (ERASMUS), CYB-FUT (ERASMUS+), InteRViR (VEGA 1/0605/23).

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, 'NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis', Aug. 03, 2020, arXiv: arXiv:2003.08934. doi: 10.48550/arXiv.2003.08934.

[2] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, '3D Gaussian Splatting for Real-Time Radiance Field Rendering', ACM Trans. Graph., vol. 42, no. 4, pp. 1–14, Aug. 2023, doi: 10.1145/3592433.

[3] (PDF) Improving NeRF Quality by Progressive Camera Placement for Unrestricted Navigation in Complex Environments'. Accessed: May 16, 2025. [Online]. Available: https://www.researchgate.net/publication/373641957_Improving_NeRF_Quality_by_Progressive_Camera_Placement_for_Unrestricted_Navigation_in_Complex_Environments

[4] 'graphdeco-inria/gaussian-splatting: Original reference implementation of "3D Gaussian Splatting for Real-Time Radiance Field Rendering"'. Accessed: May 16, 2025. [Online]. Available: https://github.com/graphdeco-inria/gaussian-splatting

[5] J. Frost, 'Mean Squared Error (MSE)', Statistics By Jim. Accessed: May 16, 2025. [Online]. Available: https://statisticsbyjim.com/regression/mean-squared-error-mse/

[6] 'What is peak signal-to-noise ratio in image processing', HowDev. Accessed: May 16, 2025. [Online]. Available: https://how.dev/answers/what-is-peak-signal-to-noise-ratio-in-image-processing