UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

Research and Development

BACHELOR THESIS

By

Nguyen Phuc Minh (BA12-118)
Data Science

Title:

3D model reconstruction from multiple view 2D images using deep learning

Supervisor: Dr. Nguyen Hoang Ha - ICT Lab

Hanoi, 2025

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Acronym	Meaning
2D	Two-Dimensional
3D	Three-Dimensional
3DGS	3D Gaussian Splatting
Abs Rel	Absolute relative error
BEiT-L	Bidirectional encoder representation from image transformers
COLMAP	Computer vision and geometry lab multi-view analysis pipeline
DSSIM	Dissimilarity form of structural similarity
EXR	Extended dynamic range
LPIPS	Learned perceptual image patch similarity
MLP	Multi-layer perceptron
MVS	Multi-view stereo
NeRF	Neural Radiance Field
PLY	Polygon file format
PSNR	Peak Signal-to-Noise Ratio
SH	Spherical harmonics
SfM	Structure-from-motion
SSMI	Structural similarity
ViT	Vision Transformer

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Recent advances in 3D reconstruction have shifted the field from classical photogrammetry toward neural rendering, with 3D Gaussian Splatting standing out for its real-time, photo-realistic novel-view synthesis. However, the original 3DGS method struggles in few-shot scenarios where only a small number of input images are available. This project aims to overcome the limitations of some of these scenes by introducing a depth-oriented optimization approach. We leverage a dense outer depth map obtained from a pre-trained monocular depth network and align it to the scene's scale using sparse SfM points. This adjusted depth map serves as a strong geometric cue that is incorporated into the 3DGS pipeline. During splat optimization, we apply a depth loss that pulls the reconstruction towards the depth prior and an edge-aware smoothness loss to adjust the local geometry. Additionally, we modify the training strategy with conservative measures: using lower-order SH for view-dependent color, removing unstable opacity resets, and applying an early stopping criterion to avoid overfitting. This improved method achieves significantly better 3D reconstruction from just five input images, eliminating floating artifacts and preserving scene structure even when the input is extremely sparse.

Keywords: 3D Gaussian Splatting; Few-Shot Learning; Multi-interface Depth Estimation System; Smoothness Regularization; Monocular Depth; Depth Regularization.

I/ INTRODUCTION

1. Context and motivation

1.1. The Evolution from NeRF to Real-Time Rendering

Reconstructing a 3D model from a series of 2D images is one of the most fascinating and foundational problems in computer vision. It is the basis for immersive technologies like AR/VR and critical applications in robotics and autonomous driving. The core challenge is to take a set of pictures of a scene from multiple angles and have a computer automatically build a 3D world that can be explored freely.

While classical techniques like Structure-from-Motion (SfM) and Multi-View Stereo (MVS) were once standard, the recent emergence of Neural Radiance Fields (NeRF) represented a revolution in photorealism. However, NeRF faces significant limitations: its training process is extremely slow and computationally heavy, requiring complex neural networks to process millions of virtual rays. This makes it largely unsuitable for real-time applications and has spurred researchers to seek faster, more efficient alternatives.

1.2. The 3D Gaussian Splatting Breakthrough

A recent breakthrough in this domain is 3D Gaussian Splatting (3DGS), introduced by Kerbl et al., which offers a path to real-time rendering without sacrificing quality. Instead of NeRF's implicit neural representation, 3DGS represents scenes explicitly as a collection of 3D Gaussians. This approach combines the advantages of point-based rendering with differentiable optimization, enabling state-of-the-art visual fidelity while maintaining competitive training times and, most importantly, achieving real-time synthesis of high-quality novel views at 1080p resolution.

2. 3D Gaussian Splatting algorithm in detail

2.1. Scene Representation

The fundamental element of 3DGS is its explicit scene representation using a set of optimizable 3D Gaussians. Each Gaussian is not just a point but a full 3D ellipsoid primitive defined by a set of learnable attributes:

- **Position (μ):** A 3D vector representing the center of the Gaussian in world space.
- **Shape (Covariance Σ):** A 3x3 covariance matrix that defines the shape and orientation of the Gaussian ellipsoid. This is decomposed into a 3D scaling vector and a rotation quaternion, allowing for anisotropic Gaussians that can stretch and flatten to efficiently represent non-uniform surfaces, such as flat planes or thin lines.
- **Color (Spherical Harmonics - SH):** To capture view-dependent effects like specular reflections, color is represented by SH coefficients rather than a single static RGB value.
- **Opacity (α):** A scalar value controlling the transparency of the Gaussian.

2.2. Rendering pipeline

The key to 3DGS's real-time performance is its novel, differentiable tile-based rasterizer, which is highly optimized for modern GPU architectures. The pipeline works as follows:

1. **Sorting:** All Gaussians in the scene are sorted once by depth from far to near.
2. **Projection:** The 3D Gaussians are projected onto the 2D screen space.
3. **Tiling:** The screen is divided into 16x16 pixel tiles. Each Gaussian is then assigned to the tiles it overlaps with.
4. **Alpha Blending:** For each pixel, the final color is computed by iterating through the sorted list of Gaussians for its tile and blending them in order from back to front. This process, a form of alpha compositing, is highly parallelizable and efficient. The system also employs optimizations like "Stop-the-Pop" to reduce rendering artifacts during camera movement.

2.3. Optimization process

The original 3DGS algorithm learns the scene geometry and appearance directly from input images through a carefully designed optimization process.

- **Initialization from SfM:** The process begins with a crucial pre-processing step using a Structure-from-Motion pipeline like COLMAP. This requires a dense set of input images, often 100 or more captured in a 360-degree video, to ensure sufficient parallax for accurate camera pose estimation and sparse point cloud generation. The initial set of Gaussians is then created directly from this sparse point cloud, inheriting its geometric positions.
- **Photometric loss:** The optimization is driven by a photometric loss that compares the rendered images to the ground-truth training views. This loss is a combination of an L1 loss and a structural dissimilarity term (DSSIM), which guides the optimization to match both pixel values and structural details.

- **Adaptive density control:** The set of Gaussians is not static. During training, the algorithm periodically performs adaptive density control. It identifies regions that are either under-reconstructed (large Gaussians) or over-reconstructed (small, dense Gaussians) and applies densification (cloning or splitting Gaussians) and pruning (removing near-transparent Gaussians) to refine the geometric representation.

2.4. Inherent limitations of the original approach

Despite its power, the original 3DGS method has a critical weakness: it performs poorly with only a few input images. Its heavy reliance on a high-quality initial point cloud from SfM means that when input data is sparse (e.g., 5-7 images), the SfM process fails to produce a reliable geometric scaffold due to insufficient parallax. Without this strong initialization, the optimization process, guided only by photometric loss, often leads to geometric collapse and floating artifacts, resulting in a failed reconstruction. Therefore, developing techniques that can reconstruct accurate 3D models from a small number of images is of great practical importance.

3. Objectives - Depth-Regularized 3DGS method

To tackle the above challenge, the objective of this project is to improve the 3D Gaussian Splatting approach for robust 3D model reconstruction in few-shot scenarios. The primary aim is to incorporate geometric priors derived from monochromatic depth estimates into the training process. These depth predictions are aligned with the camera poses generated by COLMAP and integrated into the initialization of the Gaussian arrays, effectively guiding the model with additional 3D structural information.

Another important goal is to fine-tune the loss functions used during training. In addition to the original photometric loss used in standard 3DGS, a depth consistency loss is introduced to enforce consistency between the rendered depth and the depth predicted by the monochromatic depth network. This is important for stabilizing the learning process in low data conditions. Furthermore, smoothness constraints are applied to the Gaussian parameters to encourage surface continuity and prevent abrupt spatial variations.

Finally, the project focuses on modifying the training pipeline to improve convergence behavior under sparse input, including simplifying the appearance model by using lower-order SH, removing the use of blur in training, pruning Gaussians, and adjusting the early stopping criterion. These improvements are all aimed at ensuring that the system can reconstruct plausible 3D scenes from very few images.

4. Method overview

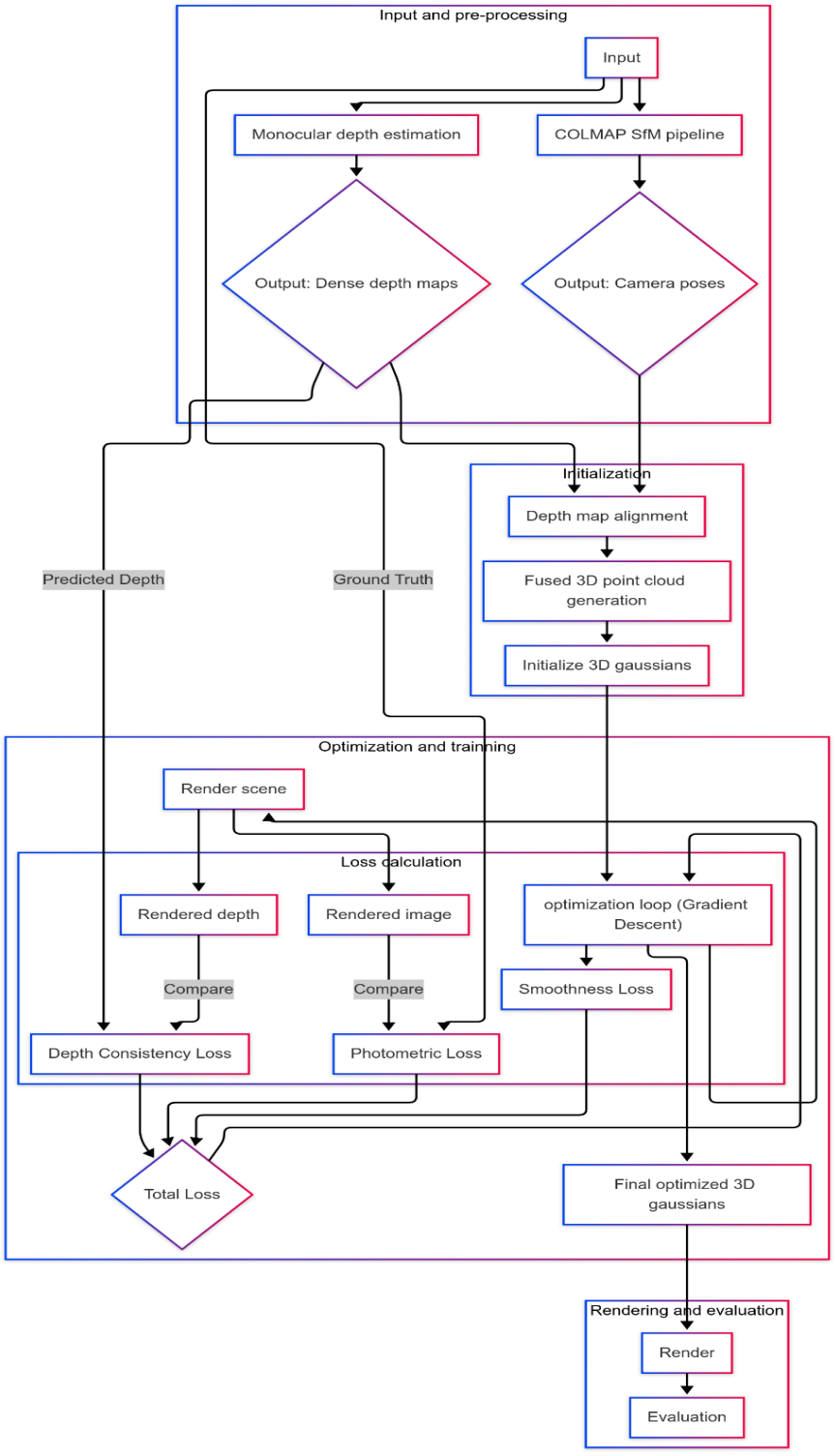


Figure 1: Program structure diagram

The overall pipeline is summarized as follows (see Figure 1):

- **Input data:** A small number of posed 2D images (typically 5-7) are collected. Camera poses are estimated via COLMAP, and a monocular depth estimator is used to predict dense depth maps for each view.
- **Gaussian initialization:** Depth maps are aligned to the SfM coordinate frame. A merged point cloud is created by back-projecting depth pixels into 3D space to form a unified point cloud.
- **Loss formulation:** Photometric Loss (matching rendered view to real image), Depth Consistency Loss (ensures rendered depth aligns with monochrome depth predictions), and Smoothness Loss (promotes smooth and stable geometry evolution).
- **Model optimization:** The parameters of all Gaussians (position, scale, opacity, rotation, color) are optimized via gradient descent.
- **Rendering and evaluation:** Once trained, the optimized Gaussians can render novel views in real time. Performance is evaluated using PSNR, SSIM, and LPIPS metrics, along with visual inspection for realism and artifact detection.

5. Structure of the report

The remainder of this report is structured as follows:

- **Section II - Theoretical background**: Presents the core concepts and principles underpinning 3D Gaussian Splatting and depth-regularized optimization.
- **Section III - Materials and methods**: Details the whole pipeline from input preprocessing to model optimization, including all technical modifications proposed for few-shot settings.
- **Section IV - Results and discussion**: Analyzes both qualitative and quantitative results, compares with the baseline, and includes ablation studies.
- **Section V - Conclusion and future work**: Summarizes the challenges faced and directions for future work.

II/ THEORETICAL BACKGROUND

1. Monocular depth estimation - Anything Depth v2

This Depth-Regularized 3DGS pipeline leverages a monocular depth prediction network (Anything Depth v2) as a guide for scene geometry. This network utilizes a ViT backbone that is pre-trained on large-scale visual data. The figure below illustrates this architecture: an input image is processed by a pre-trained ViT encoder, followed by multi-scale feature fusion blocks and depth output heads. This design enables the model to appear large to comprehend the entire scene and small to refine the details during the decoding step, resulting in sharp depth contours and stable results across a wide range of contexts. The network is pre-trained on a wide variety of data in the form of relative depths, meaning it only needs to keep the depth scale correct, not necessarily know the exact meter. By training on 595,000 synthetic images and 62 million unlabeled images, the model can be applied immediately to real-world images without requiring fine-tuning for each scene. In practice, two modes are available: relative depth models (for arbitrary images) and metric depth models (fine-tuned per domain).

Models	Year of publication	Backbone / Params	NYU-D Abs Rel	KITTI Abs Rel
Anything Depth v2	2024	ViT-Large (~330 M)	0.045	0.074
ZoeDepth	2023	BEiT-L with adaptive bins	0.077	0.054
MiDaS	2022	BEiT-L (~345 M)	0.048	0.127
BinsFormer	2023	Swin-Large (~230 M)	0.113	-

Table 1: Comparison of depth prediction methods

The choice of Anything Depth v2 over other depth methods comes from three key reasons. First, the generalizability of the model is well-suited to the few-shot problem: thanks to the huge training set that mixes indoor (NYU-D dataset) and outdoor (KITTI dataset), Anything Depth v2 accurately reconstructs scenes even with just a few frames, greatly reducing the need for long-term optimization in Gaussian Splatting. Second, the computational efficiency is superior: the ViT-Large version of the model achieves Abs Rel 0.045 on NYU-D while running significantly faster than CNN-based competitors, and the authors also provide small variants (25 M parameters) to run

on memory-constrained GPUs. Finally, the multi-scale ViT architecture keeps object boundaries sharp, reduces Gaussian stretching along the blurred edges after fitting, and the project is still actively maintained, unlike some other repos that have stopped updating (ZoeDepth). Therefore, in the context of few frames and limited GPU, Anything Depth V2 offers higher accuracy, better inference speed, sharper architecture, and flexibility (relative/metric, multiple model sizes). These advantages make it suitable for depth prediction, reduce optimization time, and allow the pipeline to run on personal laptops.

To use this depth map as a guide for 3D Gaussian Splatting, we first need to make sure it matches the scale of the 3D scene. The problem is that both the predicted depth map and the original 3D points from SfM only tell us the shape of the scene, not its actual size. We use the sparse 3D points from SfM as anchor points, then adjust (scale and shift) our dense depth map to best match these anchor points. After alignment, the monocular depth prior provides an absolute depth estimate $\widehat{D}(u, v)$ for each pixel (u, v) in each input view, consistent with the 3D coordinate frame of the Gaussian splats. This aligned depth map serves as a supervisory signal for geometry.

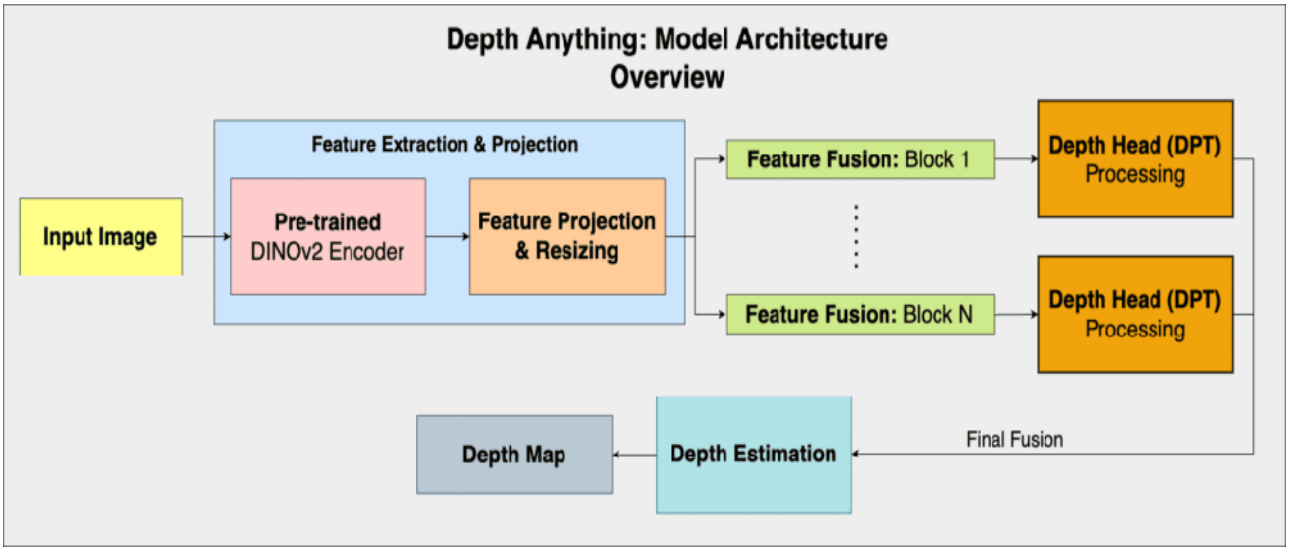


Figure 2: Depth Anything Model Architecture[1]

[1]https://learnopencv.com/depth-anything/#:~:text=In%20the%20illustration%20below%2C%20the,anything%20model%20has%20been%20shown

2. Depth consistency loss

In differentiable Gaussian Splatting, each 3D Gaussian point is not simply a point in space, but a continuous volumetric splat that can spread the color and opacity effects to multiple pixels in the output image. The special feature of this method is its ability to render a depth map directly from a set of Gaussian points, eliminating the need for a depth sensor or peripheral system. This process is similar to volumetric rendering in NeRF: at each pixel, the color streaks generated by the Gaussians are arranged in order from near to far (along the parallax axis), and then the depth of that pixel is calculated by blending the individual depths with the corresponding opacity weights. The formula developed for calculating the rendered depth at pixel p [8] is as follows :

$$D_{rend}(p) = \sum_{i=1}^{N_p} z_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j)$$

where,

- z_i is the depth of the ith Gaussian color streak.
- a'_i is the opacity weight of that color streak. This formula is a form of alpha-compositing, where the final depth is formed by the contributions of multiple layers of transparent streaks on top of each other.
- $\prod_{j=1}^{i-1}(1 - \alpha'_j)$ is the transmittance of all splats in front of the Gaussian.

The entire formula is a form of alpha compositing, which works similarly to Photoshop's splatting technique. For each pixel, the system looks through all the Gaussian layers, where not only the color but also the depth is blended along the ray based on the contribution of each Gaussian layer. This allows the model to reconstruct a 2D depth map from a 3D splat structure.

While depth rendering is a huge step forward, the accuracy of the rendered depth map is entirely dependent on how the Gaussians are optimized. If we rely solely on photometric loss, the Gaussians can still move incorrectly in space as long as the rendered image looks the same, leading to geometric errors like floating artifacts or warping. To ensure that the 3D structure is reconstructed accurately and reasonably, we add a Depth Consistency Loss function, which compares the rendered depth map (D_{rend}) with a reference depth map ($\widehat{D}$) predicted by a monocular neural network from a single image. The difference between these two depth maps is measured by the depth loss function developed in [8]:

$$L_{depth} = \frac{1}{|\Omega|} \Sigma_{(u,v) \in \Omega} \rho(D_{rend}(u,v) - \widehat{D}(u,v))$$

where,

- Ω is the set of pixels with reference depth information $\widehat{D}$.
- ρ is a penalty function (here we use Huber) to reduce the influence of outlier errors.

The Depth Loss function serves as a guide, utilizing knowledge from the single-image depth network as a form of soft ground truth. Instead of requiring the actual depth measured from the sensor, we leverage what the network learns from big data to guide the Gaussian learning process. By iteratively applying this constraint during the optimization process, the reconstructed 3D geometry avoids two common pitfalls: floating artifacts and overfitting. Instead, the model converges to a structure of color splats that is completely consistent with the overall geometry provided by the reference depth map. In essence, this loss function creates a form of intelligent geometric supervision, utilizing the knowledge from the pre-trained monocular model as ground truth, rather than requiring actual measurement data. Experimental results show that this depth-regularized optimization approach yields more precise 3D geometry and greatly enhances image quality from new viewpoints, surpassing the reliance on color alone.

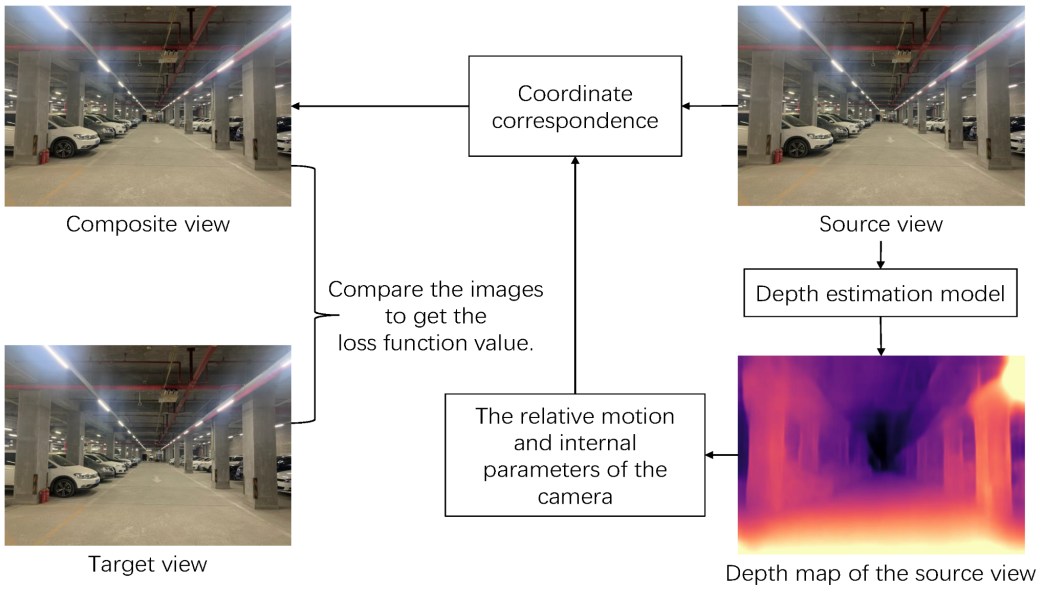


Figure 3: Depth consistency loss[2]

[2] https://www.mdpi.com/2079-9292/12/11/2390

3. Smoothness loss

In addition to the depth penalty, the model also uses another important component, the smoothness loss. This function plays a particularly important role in stabilizing the geometric optimization process, especially in regions of the image where there is no reliable depth information or it is noisy. The core idea of this mechanism is inherited from optical flow algorithms and single-image depth estimation, where object planes usually vary smoothly in 3D space – except at real edges where there are physical discontinuities.

Without any smoothness constraints applied, the model tends to generate Gaussians that float in space, which only match the surface color but are completely geometrically inaccurate. On the other hand, if a uniform smoothing coefficient is applied to the entire image, the model will be penalized even in regions with valid depth changes, such as the edge of an object. This blurs the real edges, resulting in the loss of important geometric detail.

Therefore, the smoothing function is designed to be edge-aware. Specifically, the penalty is adjusted based on the local contrast of the original image (the magnitude of the image gradient). For regions with low gradients, such as a flat wall or a smooth background, the model will penalize any depth changes strongly, forcing the 3D surface to be smooth. Conversely, for regions with sharp or high gradient edges, such as the ears of a fox or the edge of a motorcycle, the loss function will be slightly reduced, allowing Gaussians to make sudden depth jumps without being penalized. The general formula developed in [8] for the smoothness loss function is as follows:

$$L_{smooth} = \Sigma_{(u,v)}(\left|\partial_x D_{(u,v)}\right| e^{-\gamma|\partial_x I_{(u,v)}|} + \left|\partial_y D_{(u,v)}\right| e^{-\gamma|\partial_y I_{(u,v)}|})$$

where,

- $\left|\partial_{x\ or\ y} D_{(u,v)}\right|$ is the gradient of the depth map. It measures the abrupt change in depth between neighboring pixels. The larger the value, the more jagged the surface.

- $e^{-\gamma|\partial_{x\ or\ y} I_{(u,v)}|}$ is the edge detection component, with the input image I.

This gives us an idea of how this mechanism works

- At a flat image region (low $|\partial I|$): The weight $e^{-\gamma|\partial I|}$ will approach 1, causing the loss function to penalize any changes in depth very heavily. This forces the 3D surface to be very smooth.
- At a sharp edge (high $|\partial I|$): The weight $e^{-\gamma|\partial I|}$ will become very small, significantly reducing the penalty. This enables the model to generate abrupt depth jumps that are consistent with the object's contour.

Edge-aware surface smoothing not only prevents Gaussian points from floating in space but also acts as an important geometric constraint during the optimization process. When combined with a

depth loss, it forces the model to simultaneously satisfy two conditions: minimize deviations from the predicted depth map and maintain geometric continuity. As a result, the reconstructed 3D structure not only closely follows the input data but also ensures physical realism. This is especially effective in low-shot conditions, where many regions of the object are not fully observed from multiple viewpoints. The combination of the smoothing loss and the depth loss based on data predicted from a single-shot neural network forms an effective geometric monitoring system. The system enables accurate 3D surface mapping of Gaussian points without requiring actual depth maps from dedicated sensors. This is a core improvement that enables the method to outperform the original 3DGS method, which can only infer scene geometry based on photometric information.

III/ MATERIALS AND METHODS

This chapter translates the theoretical concepts into a concrete, reproducible workflow. The pipeline is divided into four stages: Data acquisition and Pre-processing, Gaussian initialization, Depth-aware training and Rendering. Each stage will be detailed below:

1. Data acquisition and Pre-processing

The original 3DGS pipeline was designed for capturing rich data. The input is typically around 100+ frames taken from a 360° video around an object. Since every surface point is observed under significant baseline changes, COLMAP reconstructs a dense, accurate point cloud; the Gaussians inherit these points and converge quickly using only photometric monitoring. Specifically, converting a set of input images into SfM data ready for training is done using COLMAP to perform a complex series of tasks. This process includes extracting features from each image, searching for matching pairs of images, reconstructing a sparse 3D model, and finally calibrating the parameters of each image to match the ideal pinhole camera model. The diversity of the viewpoints of this large image set ensures a sufficiently large variation in baseline (distance between camera positions), an important factor for COLMAP to be able to accurately triangulate and reconstruct the geometry.

In contrast, this low-shot variant requires only five to seven images facing the object. Like the original, the process still starts by running COLMAP on a small set of input images to obtain a camera pose and some rough 3D points. However, when the camera moves too little, parallax is almost non-existent, leaving large parts of the object occluded. This causes the COLMAP algorithm to struggle, producing only sparse, porous 3D models or even failing. If the Gaussian model relies solely on these few 3D points, it lacks the necessary geometric constraints and is prone to overfitting: it may learn and perfectly reproduce 5-7 training images but produce distorted, warped results when asked to render from a new perspective.

To address this issue, two additional components have been added to the pipeline. The first is the prediction of depth maps for each image. After COLMAP is run, the script will use a monocular depth estimation AI model, specifically Anything Depth v2, to generate a dense depth map for each input image. Each pixel in the image now has additional information about the estimated distance to the camera, acting as a geometric skeleton, helping to infer the overall shape of the scene even in areas that COLMAP cannot reproduce. The second is the alignment and depth storage to tie the depth prediction to the actual geometry of the scene. Since AI-predicted depth maps may not be accurate in absolute scale, this alignment step is needed to adjust the scale and position of these depth maps so that they match the coordinate system and real 3D points that COLMAP has reconstructed.

This additional information is then exploited fully in the training phase. Specifically, the trainer uses the aligned depth maps as an additional geometric constraint. A depth loss function is computed to induce the Gaussians to move toward the predicted surface.

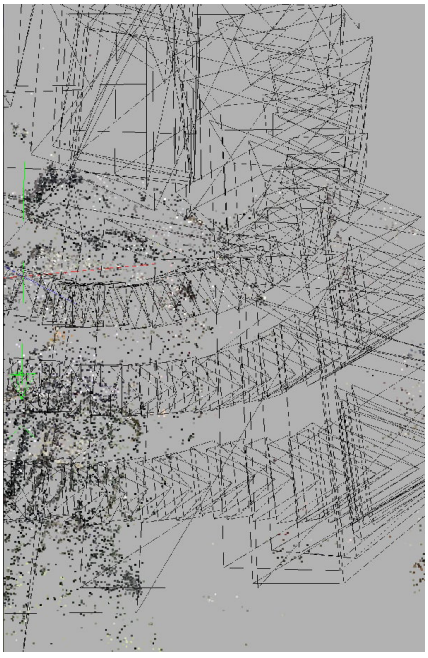
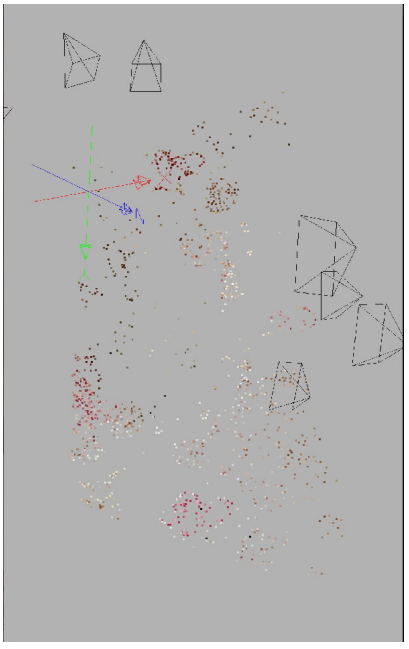
	Original 3DGS	Depth-Regularized 3DGS (Ours)
Capture protocol	A large number of still images surrounding the object, taken from the 360-degree video, are sampled continuously to ensure full spherical coverage.	From 5+ key frames capturing the sides of the object
Point-cloud density from SfM	Dense: thousands of matched key-points per frame, reliable triangulation, rich initial geometry.	Sparse: hundreds of matching points per image make the point cloud fragmented; many surfaces are not observed.
Pre-processing	Frames down-resized to 1k–2k px, undistorted, often auto-masked to remove turntable background.	Same resizing, add monocular-depth inference and scale fitting.
Need for extra priors	None: geometry is recovered by multi-view parallax alone.	Critical: depth prior & smoothness compensate for missing parallax.
Example (The camera angles are black pyramids)		

Table 2: Comparing data preparation and preprocessing

2. Gaussian initialization

After preprocessing, we have three important things: the size-aligned images, the camera pose from COLMAP, and the predicted depth map that has been refined to the COLMAP coordinate system. From each image, we back-project each pixel with its depth into 3D space to get a dense point cloud of about 1.2 million points/image. Combining all the images, merging duplicate points (< 3 mm apart) and removing stray points, we get a clean cloud that covers the planes suggested by the depth. This is a big difference from the original 3DGS, which relied entirely on the SfM discrete point cloud; now we have a relatively reliable geometry from the start.

Next, each 3D point in the refinement cloud is used to initialize a Gaussian color trace with specific properties: the center (μ_j) is the coordinate of the 3D point, the covariance matrix (Σ_j) is set to be isotropic with a radius proportional to the depth, and the initial blur (α_j) is set to a very low level of 0.01 to allow them to evolve gradually during training. The colors are directly sampled from the corresponding RGB image, and the SH degree is set to 0, ensuring that the initial colors remain unchanged with the viewing angle, thereby avoiding the situation where the model cheats on colors when the data is too small. The result of this stage is a cloud of about 3–4 million Gaussian points, forming the initial dataset ready for training.

Figure 4: Visualizing the 3D Gaussians of an object[3]

[3] https://towardsdatascience.com/wp-content/uploads/2024/03/1biRa_Ur9zPam7pbnWSWuxQ-1458x1536.png

3. Depth-Aware training

After the Gaussian swarm is initialized, the main optimization phase is performed in a fully differential manner to update all parameters of each splat. The whole process is driven by a differentiable renderer and a multi-component loss function.

3.1 Differentiable Rasterisation

For machine learning to work, we need a rasterizer that not only draws a 2D image but also backpropagates the gradients from that image back to the parameters of a 3D Gaussian point. This system inherits Kerbl's tile-based splatting rendering architecture, dividing the screen into a 16x16-pixel grid. For each Gaussian point, it computes a 2D projection and determines which tiles it intersects, then arranges all these splats once in a row from front to back within each tile. In the forward pass, colors are blended sequentially to create the final image. In the backward pass, gradients are propagated in reverse order to update the parameters.

To provide important geometric information for this process, two key mechanisms were added. First, a depth buffer is maintained in parallel with the color buffer. This buffer accumulates weighted depth values at each pixel, allowing the system to build an accurate depth map and preserve the correct geometric context as the gradient propagates. Second, to ensure consistent training speed, an optimization technique called per-tile culling is used. This technique automatically ignores color tiles that contribute too little ($< 10^{-5}$) to a region of the image, or even ignores the entire tile if it is completely occluded (when opacity is close to 1). This significantly reduces the computational load without affecting the final result.

3.2 Loss Functions

The learning process is guided by an overall objective function, which is a weighted combination of three different loss components, aiming to balance between color reproduction, geometry, and surface smoothness. For each observed image i, the model is optimized using the objective function developed in [3]:

$$L^i = \left|\left|I^i_{rend} - I^i_{gt}\right|\right|_1 + 0.2DSSIM + 0.1\left|\left|D^i_{rend} - \widehat{D}^i\right|\right| + 0.01\sum_{p,q}\rho(D^i_{rend}(p) - D^i_{rend}(q))e^{-\gamma\left|\left|\triangle I^i_{gt}\right|\right|}$$

The first component, Photometric loss, is responsible for image fidelity. By combining the L1 absolute error and the structural dissimilarity index (DSSIM), it forces the rendered image to match the real image in both color and overall structure. Next, and most importantly, the Depth loss, which directly penalizes the discrepancy between the rendered depth map and the reference depth, thereby creating a pressure that forces the Gaussian points to move to the correct geometric position in 3D space.

Finally, the Smoothness loss, with a weight of 0.01, serves as a mechanism for surface refinement. It encourages smooth reconstructed surfaces by penalizing abrupt changes in depth between neighboring pixels. To avoid blurring the sharp edges of the object, this component is designed to be edge-aware with a factor of $\gamma = 10$. Furthermore, the use of a Huber kernel (ρ) smooths out the effects of outliers, making the optimization more stable. It is worth noting that pixels without valid reference depth data are excluded from both the depth loss and smoothness components.

In summary, these three components perform three tasks: forcing color and texture matching, anchoring the rendered depth to the reference map, and maintaining smoothness of the surface except at real edges in the image.

3.3 Optimiser and schedule

Finally, to generate an accurate 3D scene, all the parameters of each Gaussian point are optimized using the Adam algorithm. However, instead of using a single learning rate, each group of parameters will have its own carefully tuned learning rate. Specifically, the position of the point is optimized with a learning rate of $1.6e^{-4}$, while the opacity is assigned a very high learning rate of $5e^{-2}$ to fill in the empty areas in the scene quickly. Similarly, the size ($5e^{-3}$), rotation ($1e^{-3}$), and color ($2.5e^{-3}$) also have separate learning rates to fine-tune each aspect effectively. To help the model converge more effectively in the end, the overall learning rate will be reduced exponentially after every 4000 iterations.

The training process is not simply about adjusting parameters. After the first 500 iterations, a densification phase begins. In this phase, the system automatically duplicates Gaussian points in areas that need more detail, based on the position gradient. This process is repeated every 100 iterations up to the 15,000 iteration mark, effectively building and refining the geometry of the scene.

The entire process is set to run for a maximum of 30000 iterations, but an early-stopping mechanism is applied to increase efficiency. Suppose the depth error on the test dataset does not improve after 5 consecutive evaluations. In that case, the training process is automatically terminated, and the best version is saved, thereby avoiding wasted resources and overfitting.

A special feature of this method is that it does not involve pruning or blurring the Gaussian points during the training process. This allows the points to converge naturally, creating sharp or smooth surfaces based on the consistency of both color and depth signals in the input data.

4. Rendering

Once training is complete, the optimized 3D Gaussian point cloud is ready for final rendering and post-processing. For novel-view synthesis, we retain and inherit the entire high-performance rendering architecture in the original 3DGS. This method relies on a tile-based rasterizer. Instead of processing each pixel individually, the screen is divided into 16x16-pixel tiles. All Gaussian points in the scene are sorted once by depth from far to near. Each 3D Gaussian is then projected onto 2D space and splatted onto the tiles it intersects. Finally, the color of each pixel in each tile is computed by alpha blending the sorted color streaks in order from front to back.

To achieve fast rendering and real-time performance, 3DGS has applied many optimization techniques and advanced features:

- Popping artifacts: To make the image move smoothly when changing the perspective, a technique called "Stop-the-Pop" is used. It will perform a super-fast local reordering step inside smaller groups of pixels (4x4). This improvement not only makes the image more consistent, but also allows for a reduction of the number of Gaussian points needed by up to half while maintaining the same quality.
- Acceleration for VR/AR: For demanding applications such as virtual reality, there is a render-only mode. This mode compresses color and transparency data into a more compact 8-bit format, reducing memory bandwidth by about 30%. This is key to achieving frame rates above 90 Hz, providing a smooth, stutter-free VR experience.
- Flexible output: Rendered images and depth data can be exported to high-quality file formats such as EXR/PLY or streamed to interactive viewing applications[4]. This allows users to control the viewing angle, change resolution and other settings in real time.

Thanks to these optimizations, the pipeline can maintain VR frame rates (> 90Hz) for MipNeRF-360 scenes at 1080p resolution. The rendering algorithm already has very high performance and is stable, so we did not need to fine-tune it further during runtime.

[4] https://github.com/graphdeco-inria/gaussian-splatting/blob/main/SIBR_viewers

IV/ RESULTS AND DISCUSSIONS

1. Results

We evaluated 3DGS by depth based on two small self-portrait scenes. All scenes were reconstructed with 3DGS baseline and Depth-Regularised 3DGS trained for 30k iterations:

Dataset	Method	Number of images	Resolution	PSNR	SSIM	LPIPS
Fox	3DGS	7	540x960	8.1726	0.4582	0.5300
	Ours	7		18.4024	0.7038	0.4442
Bike	3DGS	228	1080x1920	26.7454	0.8879	0.2876
	3DGS	29		9.6198	0.3307	0.5694
	Ours	29		18.1259	0.6481	0.4391

Table 3: Evaluation table

- Fox: +10.23 dB PSNR, +0.246 SSIM, –0.085 LPIPS over 3DGS.
- Bike: +8.51 dB PSNR, +0.27 SSIM, –0.13 LPIPS over 3DGS with the same image budget.
- Compared with the heavy 228-view baseline, our 29-view model retains ≈ 67%PSNR and 73% SSIM while using only 12% of the data.

2. Visual comparison

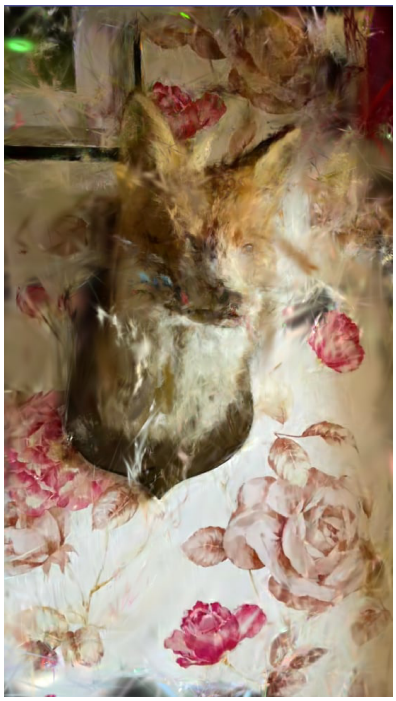
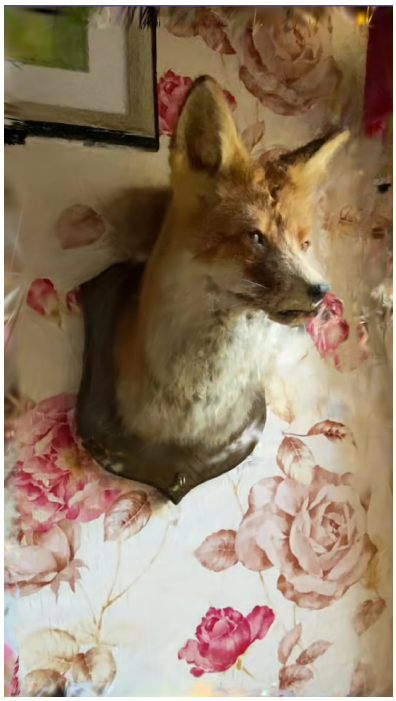
		
Ground truth	3DGS (7 images)	Ours (7 images)
Reference quality	Pure photometric optimisation cannot locate Gaussians correctly with so little parallax; geometry collapses into noisy sprites.	Depth prior anchors Gaussians on the correct depth plane; edge-aware smoothness completely removes runny hairs and color defects, restoring sharp wallpaper texture.

Table 4: Fox dataset results

			
Ground truth	3DGS (228 images)	3DGS (29 images)	Ours (29 images)
Reference quality	With dense parallax the baseline almost matches ground truth: the bike’s frame is straight, the mirror and exhaust pipe keep their sheen, and only a few isolated speckles remain.	The seat collapses, fairings warp, specular highlights smear, and the scene is littered with translucent blobs. The optimization process loses reliable information about depth and the quality of the reconstruction degrades severely.	The bike's structure has been properly restored. Focus on restoring the rigid shape of the bike, preserving sharp reflections, and reducing the floating image noise.

Table 5: Bike dataset results

3. Failure cases

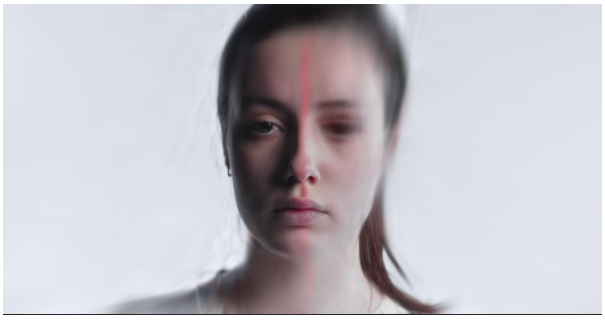


Figure 5: Failure case 1

Figure 6: Failure case 2

In additional experiments, the method fails dramatically in two scenes, as illustrated above, revealing the current limitations of the system. In case 1, the camera scans almost entirely frontally and does not capture horizontal angle differences, causing the depth network to infer that the nose and cheekbones are at different depths; during optimization, the Gaussians on the right cannot find a stable position and slide along the ray, stretching into a blurry strip, while repeating the skin color to create a red streak running down the bridge of the nose. Ironically, the edge smoothing constraint mechanism, designed to improve quality, inadvertently propagated this error across the entire face, blurring essential details such as the eyes and eyebrows.

In the pickup truck image, the problem is horizontal: the camera pans parallel to the truck's body on a flat desert floor, resulting in almost no height information (parallax along the y-axis). The monocular depth thus aligns the hood, wheels, and sand floor to the same plane; When optimized, the opacity reset and the large initial Gaussian radius cause the points to be spread into an elongated blur, distorting the car body and obliterating the tires and tail section.

Both errors demonstrate the system’s strong dependence on the depth prior. If that guess is inaccurate, which is common when viewing an object from one direction or if it's highly reflective, the system is led astray. Incorrect depth information creates artificial geometry, which results in blurred features. Instead of helping, the flawed estimate acts like a faulty mold, forcing the final 3D model into an unnatural shape and causing it to look stretched and warped.

4. Practical application

To validate the practical application, we performed an additional experiment by converting the Gaussian representation into standard 3D formats such as point clouds and meshes. These formats not only allow for visual representation but also facilitate integration into existing 3D workflows, such as advanced editing, measurement, or rendering. Using the 3DGS-to-PC[5] tool, we evaluated the quality of 3D models generated from both full and low-image data.

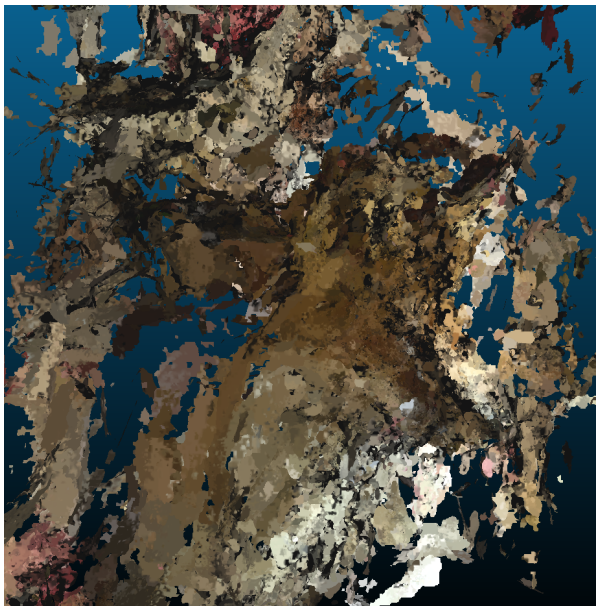
	
Point cloud	Mesh

Table 6: Practical application of the fox dataset

The Fox dataset shows a noisy and fragmented point cloud. With a limited number of views, the optimized Gaussians are not dense enough to cover the object surface continuously. This results in a sparse point cloud, which makes it difficult to reproduce a smooth and accurate mesh surface. The generated point cloud is of very low quality, which is evident in its fragmented, non-uniform, and noisy characteristics. Instead of forming a dense layer of points covering the surface of the fox, the points are sparsely distributed, creating large voids where the surface should be continuous. The point density is also uneven, with small clusters in the snout and ear areas, while other areas are almost empty. The poor quality of the input point cloud directly leads to the failure of the polygon mesh generation process. The image on the right shows that the result is not a monolithic 3D model, but a heavily fragmented surface, consisting of many disjointed and unconnected polygonal patches.

[5] https://github.com/Lewis-Stuart-11/3DGS-to-PC?tab=readme-ov-file

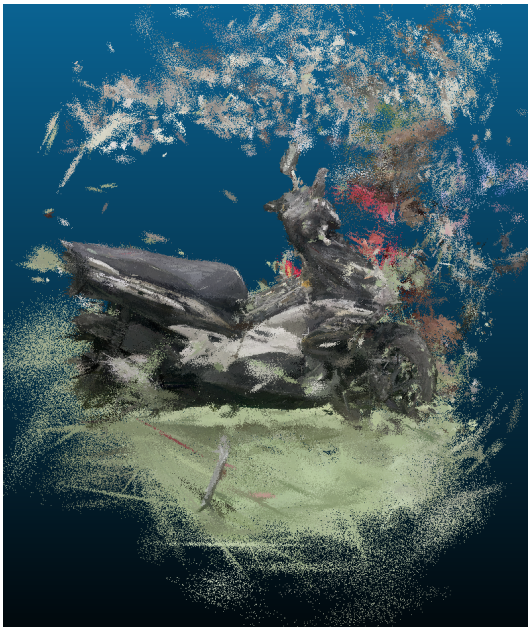
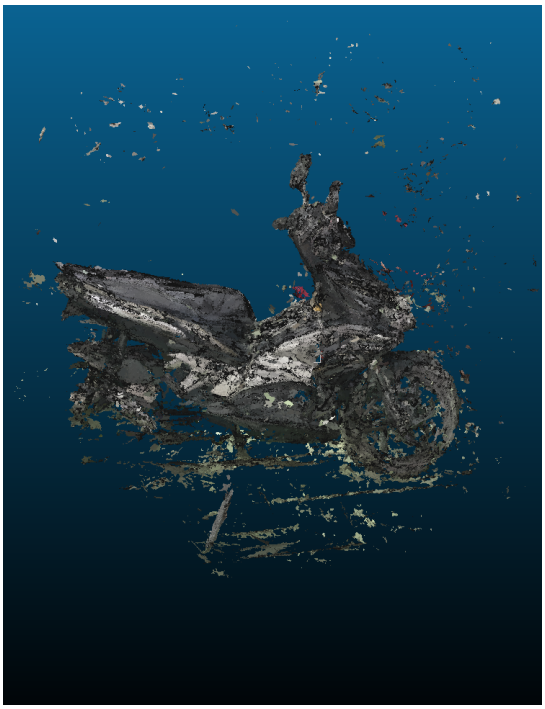
	3DGS(228 images)	Ours (29 images)
Point cloud		
Mesh		

Table 7: Practical application of the bike dataset

The dataset using 228 images can be considered a benchmark, demonstrating good quality when there is enough data. The generated point cloud is of high quality, as shown by the dense, homogeneous point density and seamless coverage of the vehicle surface. The overall shape of the vehicle is reproduced with high accuracy, and there are almost no noisy points. This shows that with enough camera angles, the original algorithm is able to confidently locate Gaussians, creating a solid geometric foundation. From the high-quality point cloud, the mesh generation process produces a nearly complete, coherent and sharp 3D model. Complex details such as wheels, saddles and bodywork are clearly represented. The mesh surface is smooth, with few errors, proving that this is a high-quality 3D asset, ready for practical applications.

The right column shows both the successes and limitations of the method under data-scarce conditions. The greatest success is that the method has preserved the overall structure of the vehicle, avoiding complete geometric collapse. However, compared to the 228-image version, the quality has significantly deteriorated. The point cloud is significantly sparser, creating visible gaps on the surface of the saddle and tires. A more serious problem is the appearance of a large number of floating noise points above and around the vehicle, with unstable colors, indicating the uncertainty of the algorithm when locating Gaussians in areas with insufficient information. The defects of the point cloud have directly affected the quality of the mesh model. Although the main frame of the vehicle is still intact, the mesh surface is fragmented and incomplete. Many important areas are riddled with large holes. The floating noise points have been interpreted by the mesh generation algorithm as small geometric fragments, hovering around the main object, seriously reducing the quality and aesthetics of the model.

5. Discussion

Our study demonstrates that incorporating depth information into 3DGS optimization process yields significant improvements over the original method, particularly in data-limited contexts.

Quantitatively, the results demonstrate the remarkable effectiveness of this method. On the Fox dataset, where we have only seven images, the difference is huge. Our model outperforms the original by up to 10.23 dB. In simple terms, this is the difference between a broken image and a clean image. On the Bike dataset, we use 29 images and still convincingly beat the original 3DGS model. Most notably, with only about 12% of the data, we achieve nearly 70% of the performance of the 3DGS model trained on 228 images. These numbers confirm the ability to reproduce images with high fidelity from a small amount of data.

Visually, the method's advantage stems from its geometric stabilization mechanism, which completely resolves the issues of structural collapse and positional ambiguity in the original 3DGS model when data is lacking. The standard model, when faced with a lack of parallax information, often encounters ambiguity in determining the position of Gaussians along the view ray. This allows the Gaussians to drift almost freely in 3D space, optimizing photometric loss, which inevitably leads to geometric collapse and creates flat, noisy objects. The combined use of the depth regularization constraint with the edge-aware smoothing mechanism, as mentioned in the previous section, forces the model to learn a consistent geometry while automatically focusing reconstruction resources on the main object where the depth information is most reliable. Since depth estimation networks tend to be more accurate for foreground objects, the depth constraint will be strongest for the central object. Conversely, in background regions where depth information may be less reliable, reconstruction will rely more on the already limited color signal, resulting in some noise remaining in the background. This explains why the texture of the motorcycle in the Bike dataset is almost perfectly preserved, while the background is less focused and may have some noise remaining.

Despite its effectiveness, our method exhibits an inherent limitation: its heavy dependence on the accuracy of the initial depth information. As experiments have shown, in monotone (low parallax) perspective scenarios, an incorrect depth estimate can lead to serious geometric errors. These errors are then amplified during optimization, causing artifacts such as distortion, stretching, and blurring of details. In essence, incorrect depth information acts as a faulty mold, forcing the final 3D model to conform to an unnatural and inaccurate structure. Another fundamental limitation exists with the original 3DGS algorithm: it is explicitly designed for object-centric scenes and consequently fails in large-scale, unbounded environments like panoramic videos. This is because its core architecture is built on several assumptions that do not hold true in the panorama context. First, the algorithm assumes a set of inward-facing cameras observing a limited space. But fundamentally, its rasterization process is designed for conventional pinhole camera projections. It

assumes that a 3D Gaussian projected onto a 2D image plane will form a 2D ellipse. However, as recent research has shown[6], this core assumption breaks down completely for panoramas. Splatting a 3D Gaussian onto the curved surface of a spherical image (represented in equirectangular format) results in a distorted shape that the algorithm's flat 2D Gaussians cannot accurately model. Furthermore, these choices affect the 3D space parameterization, which uses Cartesian coordinates that are not suitable for representing the background at infinity. Therefore, because of fundamental incompatibilities in both the camera geometry assumptions, the projection mathematics, and the space parameterization, the original 3DGS algorithm cannot be directly applied to panoramic scenes.

Additionally, the Poisson Surface Reconstruction algorithm used in the 3DGS-to-PC tool works best when there is a sufficiently dense and clean point cloud to interpolate a closed surface. When the input point cloud is too sparse and porous, the algorithm fails to bridge large gaps. Instead, it only generates small surface fragments where there are point clusters, resulting in a jagged, fragmented, and porous mesh that does not represent the true shape of the object. The result is a surface that looks like a shard of glass, completely unusable for practical applications such as editing or animation. Validating the application based on converting the Gaussian representation into standard 3D formats such as point clouds and polygon meshes also highlights the weakness of this program. With enough data, the original model produces a dense point cloud, which forms the basis for a near-perfect 3D mesh. However, in the low-image scenario, while the overall structure is preserved, the geometric quality of the point cloud deteriorates significantly, becoming sparser and noisier.

[6] https://arxiv.org/abs/2402.00763

V/ CONCLUSION AND FUTURE WORK

This research presents a depth-constrained optimization method for 3D Gaussian Splatting that significantly improves the quality of 3D reconstruction from a limited number of images. By integrating depth information from a single-image network and applying an edge-preserving smoothing mechanism, our method overcomes the core challenges faced by standard 3DGS in the low-image case. We demonstrate significantly higher fidelity on standard datasets where conventional methods would yield very poor results or even fail. Visually, our approach preserves geometric structure and details much better than the original model, successfully reconstructing structures that would otherwise collapse when data is scarce.

Based on the analyzed limitations, our long-term goal is to comprehensively improve the practical applicability of the method, focusing on improving the quality of the mesh conversion so that the output is smooth, clear, and easily integrated into standard 3D workflows, as well as scaling the processing to go beyond just object-centric datasets. To achieve this, we will focus on the following key research directions:

- Research to improve existing conversion algorithms such as 3DGS-to-PC, or to find alternative methods that can better handle sparse and noisy point clouds. Techniques such as Screened Poisson surface reconstruction or neural network-based implicit surface representations are expected to yield smoother and more coherent results.
- Optimize the Gaussians themselves to be more mesh-friendly, by adding geometric constraints during training to encourage a dense and uniform distribution.
- Extend the algorithm to be able to reproduce large-scale scenes and 360-degree panoramic videos. The main approach is to develop an architecture that uses separate Gaussian sets or different parameterization methods to effectively manage foreground and background objects[7].

Successfully recreating these complex scenes would make the algorithm a universal tool, capable of digitizing everything from a small object to an entire city block. In short, the project has successfully implemented a 3D reconstruction method from a 2D dataset, solving the strict data quantity requirements of the original algorithm and giving positive initial results. Future developments will be the next necessary steps for the algorithm to become more complete and useful in real life.

[7] https://github.com/inuex35/360-gaussian-splatting?tab=readme-ov-file

REFERENCES

[1] Chung J, Oh J, Lee KM (2024) Depth-Regularized Optimization for 3D Gaussian Splatting in Few-Shot Images. Proceedings of CVPR Workshops (3DMV, CVPRW 2024). Available: https://openaccess.thecvf.com/content/CVPR2024W/3DMV/html/Chung_Depth-Regularized_Optimization_for_3D_Gaussian_Splatting_in_Few-Shot_Images_CVPRW_2024_paper.html

[2] Kerbl B, Kopanas G, Leimkühler T, Drettakis G (2023) 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Transactions on Graphics. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[3] Kumar R, Vats V (2024) Few-shot Novel View Synthesis using Depth Aware 3D Gaussian Splatting. arXiv:2410.11080. Available: https://arxiv.org/abs/2410.11080

[4] Lewis A G Stuart, Michael P Pound (2025) 3DGS-to-PC: Convert a 3D Gaussian Splatting Scene into a Dense Point Cloud or Mesh. arXiv:2501.07478. Available: https://arxiv.org/abs/2501.07478

[5] Xu H, Peng S, Wang F, Blum H, et al. (2024) DepthSplat: Connecting Gaussian Splatting and Depth. arXiv:2410.13862. Available: https://arxiv.org/abs/2410.13862

[6] Yang L, Kang B, Huang Z, Zhao Z, Xu X, Feng J, Zhao H (2024) Depth Anything V2. arXiv:2406.09414. Available: https://arxiv.org/abs/2406.09414

[7] Yin R, Yugay V, Li Y, Karaoglu S, Gevers T (2024) FewViewGS: Gaussian Splatting with Few View Matching and Multi-stage Training, arXiv:2411.02229. Available: https://arxiv.org/abs/2411.02229

[8] Zhu Z, Fan Z, Jiang Y, Wang Z (2023) FSGS: Real-Time Few-shot View Synthesis using Gaussian Splatting. arXiv:2312. Available: https://arxiv.org/abs/2312.00451