POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

3D Gaussian Splatting for UAV-Based Reconstruction of Urban Environments

Supervisors

Prof. Andrea BOTTINO

Prof. Dr. Francesco NEX

Candidate

Carolina ROVEGNO

April 2025

Alle Nonne, il mio tesoro più grande.
Al Nonno, la mia stella.

Acknowledgements

Coming to the end of this long journey, some acknowledgements are necessary.

To my supervisors, Prof. Andrea Bottino and Prof. Francesco Nex for their support and for giving me the opportunity to work on this project at the University of Twente. The months spent in Enschede have been incredible, an experience I will never forget.

To my family, for their unwavering support and for never stopping believing in me, even when everything seemed dark.

To my lifelong friends, Laura, Giulia and Serena, for always standing by my side in every situation.

To the friends I met in Torino, Alessandro, Letizia, Simona, Valentina, for sharing this journey with me and for making these years a little lighter.

Finally, to those who have been with me from the beginning but are no longer here to see me cross this finish line with a laurel crown on my head. This achievement is also for you.

Abstract

UAV-based 3D reconstruction of large-scale environments has been largely adopted in different domains, starting with virtual reality and 3D documentation and moving on to more practical applications such as land analysis, urban area mapping and disaster management. The reconstruction techniques used are mainly based on two different approaches: with passive sensors, relying on image-only methodologies and with active sensors, which consist of laser-based methods (such as LiDAR). Image-based reconstruction was traditionally based on photogrammetric computer vision, but deep learning has recently innovated these techniques. Deep learning-based methods can learn how to represent three-dimensional scenes to generate realistic renderings from a sequence of images. Among these approaches, Neural Radiance Fields (NeRF) and 3D Gaussian Splatting (3DGS) have recently shown promising results in the generation of accurate rendering, but the quality of the generated 3D point cloud is often neglected. The main object of this research project is to assess the 3D Gaussian Splatting algorithm in terms of quality of the generated 3D reconstruction. In particular, both the original implementation and its updated version, called DN-Splatter, are considered during the evaluation process. DN-splatter takes into consideration normal and depth maps extracted from the data to obtain a more accurate representation from a geometrical point of view. The evaluation is conducted on three different UAV-captured datasets of urban environments, two of them including ground-truth LiDAR point clouds. In order to assess the algorithms' performances, both renders and point clouds are evaluated with appropriate metrics. In particular, PSNR, LPIPS, and SSIM are used to evaluate the generated images, thus the quality of the final renders compared to UAV images. On the other hand, the generated point clouds are evaluated from a geometric point of view, by calculating the point-to-point distance from the LiDAR point cloud. Although the quality of the final renders does not show an evident difference between the two algorithms, the results show DN-Splatter outperforms the original 3D Gaussian Splatting in geometric accuracy. In that regard, the rendering is able to "hide" incorrect 3D reconstructions that are more evident in the point clouds comparison.

Table of Contents

List of Tables

List of Figures

Listings

Acronyms

UAVs
Unmanned Aerial Vehicles

SfM
Structure from Motion

MVS
Multi-View Stereo

NeRF
Neural Radiance Fields

3DGS
3D Gaussian Splatting

DNS
DN-Splatter

H3DG
Hessigheim 3D Geometric Benchmark

PSNR
Peak Signal-to-Noise Ratio

SSIM
Structural Similarity Index Measure

LPIPS
Learned Perceptual Image Patch Similarity

Chapter 1

Introduction

1.1 UAVs for 3D Reconstruction

Unmanned Aerial Vehicles (UAVs) are a very useful tool for solving tasks in different scenarios. The ability to be remotely guided or fly autonomously, combined with their small size, light weight and agility makes them the best choice in situations where, for various reasons, it is not possible for humans to reach the scene to be reconstructed. One example is emergency situations or whenever there is a need to capture aerial images for tasks like land analysis. These devices can capture different kinds of data, as they can be equipped not only with RGB and thermal cameras, capable of acquiring high-resolution images, but also sensors that can extract in-depth information of the scene. An example is LiDAR, which will be discussed in the next chapter. The ability to acquire these types of heterogeneous data, either separately or in a hybrid manner, makes them very useful in the field of 3D reconstruction [1] and urban mapping [2], for both active and passive approaches. The reconstruction process can be performed either online or offline, depending on when the data processing is done. Both approaches are based on feature extraction from the scene, followed by mapping and creation of the three-dimensional environment. The offline methodology is based on Structure from Motion (SfM) which is a passive reconstruction technique, that only uses 2D images to create the 3D scene. This approach needs time to be able to process all the data and is often used in photogrammetry or modeling, as well as land analysis and general 3D reconstruction. In contrast, the online method is based on the Simultaneous Localization and Mapping (SLAM) technique. It can also integrate other sensors to acquire information on the device's position and create a map of the surrounding environment. This method is often used for autonomous navigations tasks. In emergency situations it is necessary to use online approaches to obtain results as quickly as possible.

1.2 Motivations and Goals

3D reconstruction of urban environments can be highly useful in different scenarios, from computer vision tasks to virtual reality applications. However, there are some difficulties to consider when recreating a 3D scene. Poor or variable lightning and adverse weather conditions, occlusions and data noise can represent a challenge, especially on large-scale datasets. Additionally, these types of datasets can be problematic in terms of processing time and computational resources, especially when traditional rendering techniques are used, since a dense point cloud is required to achieve accurate results. Also, deep learning techniques may not be as accurate in terms of results if the selected dataset is quite different from the data used in the network training process. 3D Gaussian Splatting [3] tries to partially address these challenges by optimizing the point parameters directly from the input images, without the need for a complex neural network. It implements volumetric rendering, which enables high-quality results even with a sparse point cloud. This allows the processing of a larger amount of data without losing quality reconstruction. However, the algorithm's performance has been so far evaluated on small-scale datasets, which differ in features from the datasets considered in this research project. The main objective is to evaluate the 3D Gaussian Splatting performances in reconstructing large-scale urban environments using high-resolution UAV-images as input data. In particular, the goal is to determine whether the algorithm is capable of accurately reconstructing the geometry of the scene while also generating high-quality real-time renderings.

1.3 Thesis Structure

This research project, with a focus on outdoor environment reconstruction using 3D Gaussian Splatting, is divided into five different chapters:

- **Chapter 2** presents an overview of 3D reconstruction techniques, highlighting the differences between passive and active methodologies. In addition, it also shows the main features of deep-learning-based method which can be both passives, active or even hybrid techniques, based on the scenario and the available data.
- **Chapter 3** focuses on 3D Gaussian Splatting algorithm, documenting its key features, workflow and optimization process which led to a sparse point cloud representation of the scene. Some of the algorithm's limitations are also presented, together with a possible solution: the DN-Splatter algorithm [4].
- **Chapter 4** describes the methodology. Initially, the three datasets selected for the reconstruction process are presented, followed by a description of the

metrics used to evaluate the resulting renders. Next, the chapter focuses on the post-processing of the generated sparse point cloud and the description of the point-to-point distance computation between the output and the LiDAR point cloud.

- **Chapter 5** presents and discusses the results obtained on the three datasets, both visually and geometrically.
- **Chapter 6** concludes the research project summarizing the key concepts and proposing possible insights for future work.

Chapter 2

Background

The process of 3D reconstruction can be accomplished through a variety of methods, ranging from deep learning-based to conventional ones. Depending on the sort of data the algorithm receives as input, these can be divided into sensor-based (active approaches) and image-based (passive approaches). Over the years, numerous studies [5][6][7] have been carried out over time to assess how well certain methods work in resolving particular issues.
An overview of various reconstruction techniques is provided in this chapter.

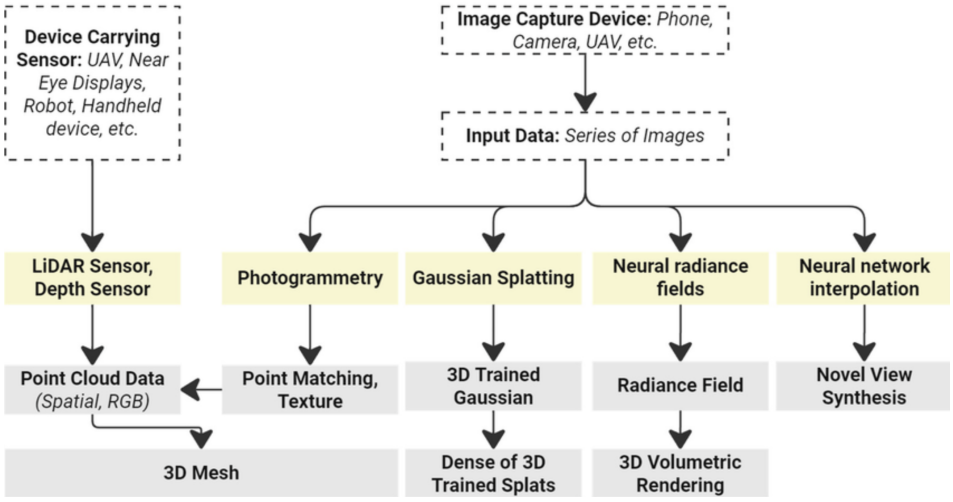

Figure 2.1: Common 3D reconstruction methods for real-world environments. `Source:` [8]

2.1 Active 3D Reconstruction

Active reconstruction approaches use sensors that emit an active signal to learn about the geometry of a scene. These techniques generate depth information dynamically from the signal's interaction with the surroundings rather than simply depending on ambient light or photographs. There are several techniques that are characterized by the use of different signals. By measuring the distance between the sensor and objects, laser pulses can provide depth information about the environment. Other methods employ light signals projected onto the surface of the scene and they analyzed the distortion to calculate depth.
Active reconstruction's independence from lighting conditions and ability to obtain precise depth information even in low-light and low-texture scenarios are its main advantages. The drawback is the need for specialized technology, which inevitably raises the cost of this process compared to methods that only employ RGB images. In addition, the reconstruction process may be complicated by possible noise and interference issues in scenarios with reflecting or clear surfaces.
The LiDAR sensor, which is the specific type of sensor employed to collect the ground-truth data used in the experimental phase of the research project, will be introduced in the next Section.

2.1.1 LiDAR-Based Reconstruction

LiDAR (Light Detection and Ranging) is a remote sensing technology which measures the precise distance between a sensor and nearby objects using laser beams. The basic concept is based on the emission of a laser pulse which is then reflected off the impacted surfaces before returning to the sensor. A finely detailed three-dimensional representation of the scene is produced by calculating the distance between the sensor and the object based on the time it takes for the pulse to return to the receiver. One of LiDAR's primary benefits is its capacity to gather data in challenging locations, low light levels, and adverse weather situations where image-only techniques like photogrammetry might not be as effective. In addition, this technology is really useful to address terrain mapping tasks thanks to its ability to partially penetrate vegetation Its drawbacks, however, are the high expense of specialized technology, the requirement to process vast volumes of data in order to produce precise models, and the incapacity to extract scene-specific texture or color information.
In recent years, LiDAR has been widely used in several fields, including mapping, autonomous driving, archeology, and environmental monitoring [9]. In particular, the 3D reconstruction of outdoor environments has been transformed by its integration with Unmanned Aerial Vehicles (UAV) systems [10][11]. These devices represent the ideal equipment to solve these tasks since they they can rapidly

fly over large areas and gather incredibly detailed information with centimeter accuracy.

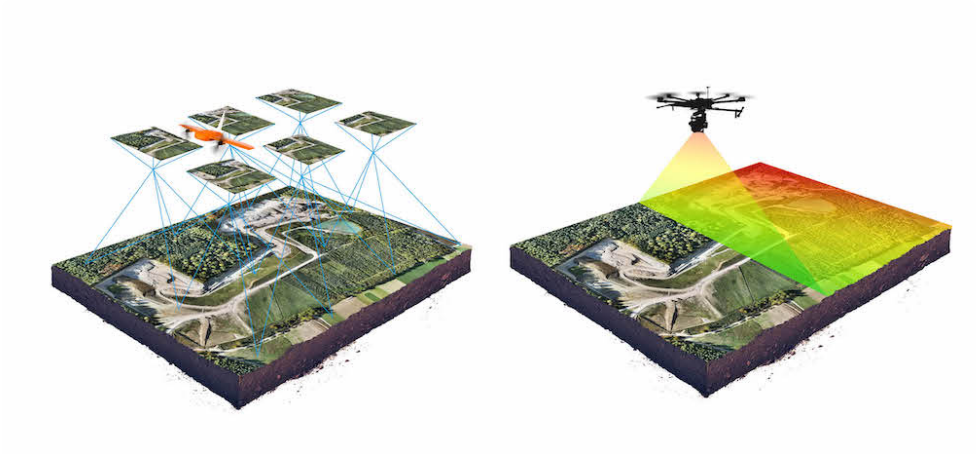

Figure 2.2: Different UAV data collection for urban environment reconstruction tasks: RGB sensor acquisition (passive techniques) on the left, LiDAR sensor acquisition (active techniques) on the right.

2.2 Passive 3D Reconstruction

Passive 3D reconstruction techniques [12][13][14] rely only on 2D images acquired by cameras, without the need for active signals. These methodologies use image analysis to extract geometric information and reconstruct the three-dimensional structure of the scene. Passive techniques have several advantages, including simplicity of data acquisition, since a set of images taken with standard cameras is sufficient. They are also cheaper than active methods and can be applied to a wide range of contexts, such as photogrammetry, archaeology and urban modeling. On the other hand, relying on RGB data only, they are sensitive to all limitations associated with image acquisition, such as variable lighting conditions, data resolution, data noise and possible lack of texture information. In recent years these passive approaches have seen great improvements due to deep learning-based techniques [15]. For the sake of this chapter, a passive technique will be addressed in Section 2.3.1, even though these methods are frequently associated with hybrid approaches, where reconstruction is accomplished by integrating RGB images with data from active sensors.

2.2.1 Sparse Reconstruction: Structure from Motion

Sparse Reconstruction is a passive 3D reconstruction technique that aims to obtain a three-dimensional representation of the scene using a limited number of keypoints extracted from RGB images captured from different perspectives. This task is as useful as it is complex; in fact, it is easy to go from a 3D world to a 2D one by capturing a series of images, but the reverse step is not so intuitive. A whole range of factors must be considered.
The Sparse Reconstruction process generally follows a specific pipeline, divided into four main steps:

- **Feature extraction.** This initial stage enables the identification of potential key points in the images. These points must be invariant to geometric and radiometric transformations to ensure robustness even under varying lighting conditions or different camera angles. The most used descriptors that satisfy these conditions are SIFT (Scale Invariant Feature Transform) descriptors [16], which allow robust feature points to be detected at rotation, scale, and illumination. These key points provide key geometric information for the subsequent image matching step.

- **Feature matching** After feature extraction, matches must be established between images to determine which points represent the same portion of the scene in different views. Feature matching is done by comparing the extracted feature descriptors and selecting the best matches using similarity metrics, such as Euclidean distance. This step is crucial, as incorrect matches can compromise the quality of the final reconstruction.

- **Scene reconstruction through triangulation.** The position of cameras and keypoints is estimated in the three-dimensional space and the sparse representation is generated accordingly. This is done through an iterative process where the position of the cameras and images is estimated with respect to the 3D scene initially created from a pair of images.

- **Bundle adjustment.** This is a final optimization stage which reduces the projection error by minimizing the discrepancies between the acquired images and the reconstructed 3D model. This process improves reconstruction accuracy and ensures geometric consistency between different images.

This Section examines the Structure from Motion (SfM) [18][19] technique, which is a popular photogrammetry technique for reconstructing a 3D sparse scene based on this approach. This technique turns out to be widely used in different domains, from computer vision to archaeology, from 3D scanning for the reconstruction of minute objects to the reconstruction of large-scale monuments or scenes. Unlike

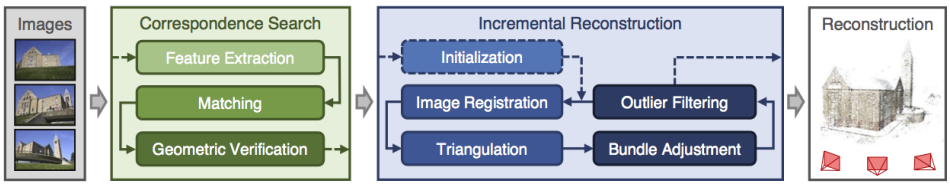

Figure 2.3: Incremental Structure-from-Motion pipeline. **Source:** [17]

traditional photogrammetry, which needs to know the location of certain cameras in advance to accurately recreate the scene, SfM automatically derives this information using the pipeline mentioned above and showed in Fig. 2.3. This way of extracting information makes this approach suitable to be used at consumer level as well, as it achieves respectable results even on data acquired from less high-performance cameras and even smartphones. It can adapt to different levels of input data quality. Despite this, there are also cases where this methodology fails to perform at its best. The main problem lies in the reconstruction of scenes or objects without textures or at least without enough key points to rely on for image overlay. It is much more complex to work with images that represent a very uniform scene with not enough edges or features to align. Moreover, this method produces only a partial representation of the scene, proving insufficient for applications requiring complete and detailed three-dimensional models. For this reason, it is often used as a preliminary step before proceeding with more advanced dense reconstruction techniques.
This approach is also used by the 3D Gaussian Splatting algorithm, using COLMAP software [20][21], to generate the sparse point cloud shown in Figure 2.4 from which the optimization process starts.

Figure 2.4: Structure-from-Motion sparse reconstruction of urban environment.

2.2.2 Dense Reconstruction: Multi-View Stereo

Dense Reconstruction is a 3D reconstruction approach designed to produce a detailed and continuous representation of a three-dimensional scene. Unlike Sparse Reconstruction, which uses a small number of characteristic points, Dense Reconstruction produces a high-resolution point cloud or complex three-dimensional mesh that better captures the geometry of the environment. The Dense Reconstruction process often begins with a preliminary sparse reconstruction step, which offers an initial estimate of camera positions and a rough structure of the scene. Next, the point cloud is densified using complex techniques that take advantage of image correlation. Multi-View Stereo (MVS) [22] is a common approach that employs photos from several viewpoints to accurately rebuild depth and three-dimensional structure. Its main goal is to densify the sparse point cloud generated by SfM in order to obtain a detailed representation of the scene. There are multiple MVS algorithm versions [23], each with unique properties that influence reconstruction quality, processing requirements, and application to a wide range of scenarios, from the most complicated scenery to the reconstruction of minute objects. Point-based techniques work directly on the sparse point cloud acquired from SfM and proceed to densify it by gradually adding points estimated from multiple perspectives using a triangulation procedure. These are highly flexible and computationally efficient they so can be employed in a variety of fields, but they are also vulnerable to errors in matching and generating continuous surfaces. In contrast, voxel-based and patch-based volumetric approaches work on a volumetric representation of the three-dimensional environment, defined by spatially oriented voxels and patches, respectively. They are capable of reconstructing complex scenes in great detail and

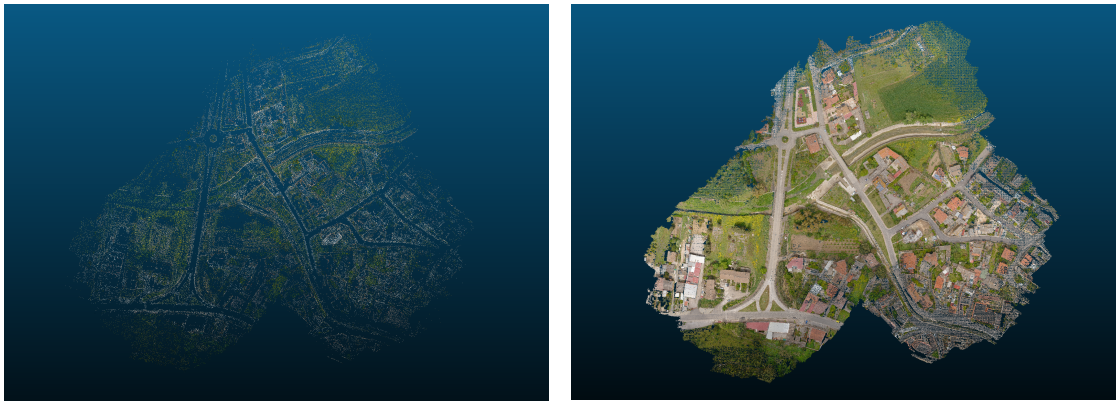

Figure 2.5: Sparse reconstruction (on the left) versus dense reconstruction (on the right) of the same urban environment: comparison of resulting point clouds obtained respectively with SfM and MVS technique.

are resistant to outliers, but they are also computationally expensive and memory demanding. The most frequently used variant of MVS is the depth-based method. This approach involves generating a depth map for each input image, which consists of a grayscale image having each pixel representing the distance between the camera and a point in the scene. Next, these maps are merged to generate a dense point cloud. Finally, after filtering procedures and removing any outliers to increase reconstructive quality, the point cloud is converted to a three-dimensional surface using meshing methods.

2.3 Deep Learning for 3D Reconstruction

Deep learning has revolutionized the field of 3D reconstruction by introducing new approaches [24][25] based on neural networks capable of learning intricate geometric representations from images, point clouds or volumetric data. Traditional systems like Structure from Motion (SfM) and Multi-View Stereo (MVS) use geometric methods to recreate the three-dimensional structure of a scene from images. These techniques, however, necessitate a large number of pictures with significant overlaps, are sensitive to texture and illumination changes, and can produce noisy or inadequate reconstructions in areas with few distinguishing features. With the introduction of models based on convolutional neural networks (CNNs) and more sophisticated architectures like Generative Adversarial Networks (GANs) and Neural Radiance Fields (NeRFs), deep learning has completely changed this process. These techniques overcome many of the limitations of conventional methods by learning how to recreate 3D scenes directly from images without the need for exact geometric information. Specifically, methods like NeRF eliminate the necessity for dense point clouds or explicit meshes by enabling a continuous representation of the world throughout a radiance field. Methods based on deep learning are generally more resilient than conventional techniques, which means they can manage situations with inconsistent lighting or inadequate texture information. This is possible thanks to the large amount of data the models are trained on, which allow them to perform well even in challenging circumstances. The models are also able to create novel views, which means they can be used to extract valuable reconstruction information from perspective not captured by original images. However, some limitations have to be considered when choosing a deep-learning approach to solve the reconstruction task. First of all, these models are computationally expensive, and they require specific hardware to perform correctly. Additionally, they heavily rely on training data. The results could be unpredictable and the performances less accurate when applied to input data with significantly different features from the training set. Lastly, some deep learning based methods ignore the underlying geometric information in favor of a visually

appealing reconstruction. As a result, they frequently highlight the scene's aesthetic value over its geometric accuracy. This might be acceptable in situations where the visual element is more crucial, like virtual reality applications, but it becomes an issue in any scenario where the environment must be precisely recreated, such as indoor or outdoor mapping.

2.3.1 Neural Radiance Field

Neural Radiance Fields (NeRF) is a new deep learning technique for modeling and rebuilding 3D scenes using 2D images introduced in 2020 by Mildenhall et al. [26]. This technique represents a three-dimensional scene as a 5D radiance field, which means that for each point in space, the model considers not only its spatial position (x, y, z) but also the viewing direction (θ, ϕ). The visual appearance of a point in a scene might vary depending on the perspective, especially for materials that reflect light in different ways. The radiance field is represented by a neural network trained to learn the function F(x), which assigns a color value (r, g, b) and volume density σ to each point in 3D space.

$$F(x, y, z, \theta, \phi) \rightarrow (r, g, b, \sigma)$$

The model consist of a two-stage Multi-Layer Perceptron (MLP): the first calculates the density and a vector of intermediate features, and the second uses these features, along with the viewing direction, to determine the final color. Figure 2.6, which is taken from the original paper by Mildenhall et al. , shows the rendering process. The procedure is based on a technique known as volume rendering, which involves casting virtual rays across the scene. Along these, multiple 3D points are sampled at different depths. The information about color and volume density of each point, are then combined to calculate the final intensity of the associated pixel in the reconstructed image. The weighted sum of all sampled points' colors, weighted by their respective opacities, produces the final pixel value in the rendered image. Model optimization involves decreasing the difference between generated and ground-truth images using photometric loss based on the mean square error. Due to its ability to capture fine image details, this method allows for the creation of high-quality, photorealistic three-dimensional scenes. It supports continuous scene representation without the need for explicit meshes, which makes the final results considerably smoother. Additionally, it has the ability interpolate information between distinct views, resulting in new perspectives that did not appear in the training data. In general, it is capable of handling more complex scenes which have particular geometries and non-uniform lighting conditions, resulting in a qualitatively more accurate representation compared to other traditional methods such as MVS. On the downside, this method has some limitations. For starters, it

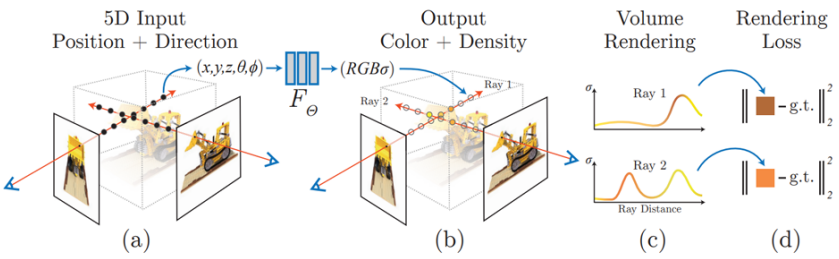

Figure 2.6: NeRF scene representation and rendering procedure.

is a computationally expensive approach that demands significant training time and memory resources. Producing a single image requires sampling thousands of points along a ray, making it an extremely inefficient process because the MLP network must be trained for each scene individually. This can require a large number of training hours in case performance hardware is not available. To solve these challenges, alternatives such as 3D Gaussian Splatting have been designed, which provides a more efficient and faster rendering approach while maintaining the visual quality of NeRF. The 3DS algorithm, which is the main object of this project, will be discussed in Chapter 3.

Chapter 3

3D Gaussian Splatting

3.1 Introduction to the Algorithm

In 2023, Kerbl et al. proposed 3D Gaussian Splatting (3DGS) [3] as a new approach for representing and synthesizing three-dimensional scenes. It is suggested as a more effective substitute for Neural Radiance Fields (NeRF), providing much faster rendering without sacrificing visual quality. In contrast to NeRF, which reproduces the scene using an implicit radiance field learned from a neural network, 3DGS adopts a representation based on 3D Gaussians, which are gaussian distributions placed into three-dimensional space. Each entity is characterized by specific properties such as position, color, opacity, and spatial covariance.
Then, using a sophisticated rasterization technique, these gaussian distributions are projected onto the image plane and merged to create photorealistic images in real time. Among 3D Gaussian Splatting's most innovative characteristics is its ability to dynamically adapt to the scene's complexity and appropriately distribute gaussian points to achieve the ideal balance between performance and quality. These features make 3DGS one of the most promising methods for real-time rendering applications, including visual 3D reconstruction, virtual reality, and immersive environment generation.
This chapter will describe the 3D Gaussian Splatting algorithm, including its primary features, its unique gaussian optimization and the rendering procedure. Some of its drawbacks will be discussed, as well as a potential solution, the DN-Splatter, which is an improved version of the original algorithm that takes into consideration also normal and depth maps extracted from the input data to generate a more geometrically accurate model of the scene.

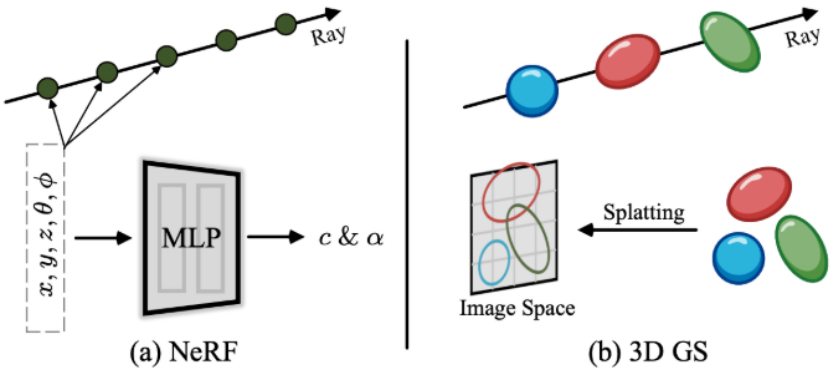

Figure 3.1: NeRF and 3DGS different rendering concepts. NeRF employs an MLP to estimate density and color at each sampled position after periodically sampling each ray from the image to the scene. Gaussian Splatting uses elliptical splats to determine each gaussian's contribution to the light beam. **Source:** [27]

3.2 Algorithm's Pipeline

This Section presents the algorithm pipeline, illustrated in Figure 3.2 from the original paper by Kerbl et al. Several works have since focused on providing a more detailed description of the 3D Gaussian Splatting process from an algorithmic perspective [27][28][29]. The algorithm's pipeline is based on specific stages. Starting with the generation of the sparse point cloud through the use of COLMAP library, we proceed to the rasterization process where the 3D gaussians are mapped into a bidimensional space generating an image. Then, the generated image is then compared to the ground-truth image using a unique loss function and the error is used as a reference to update the gaussian parameters in the optimization step. This type of pipeline allows for a detailed representation of the reconstructed scene, minimizing the time required for the rendering phase. The various pipeline stages will be explained in depth in the following Sections.

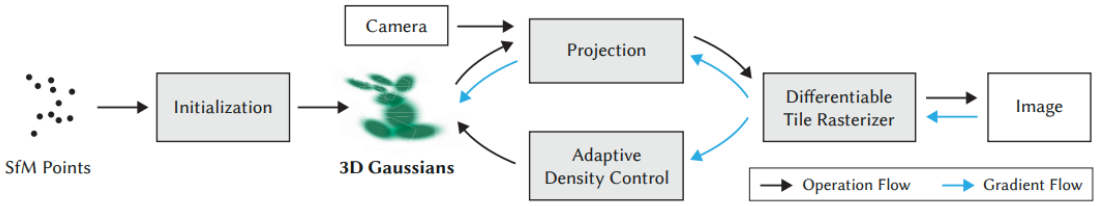

Figure 3.2: 3D Gaussians Splatting's pipeline

3.2.1 Initialization

The initialization process, which is the starting point of the algorithm, consists of the implementation of the SfM technique, previously explained in Section 2.2.1. Starting with the set of two-dimensional images as input, using the COLMAP library, the sparse point cloud representing the scene is generated. Each point is then turned into the center of a three-dimensional anisotropic gaussian, which is defined by four distinct parameters:

- **Position (x, y, z)** , which is defined by three coordinates in the three-dimensional space and represents the location of the gaussian mean (μ);
- **Color**, which is represented by Spherical Harmonics (SH) coefficients. Each gaussian stores a set of SH coefficients for each color channel (R, G, B), allowing the color to shift according to the view direction;
- **Opacity (α)**, which quantify how much a single point is visible in the scene;
- **Covariance (σ)**, represented by a 3x3 matrix, generated from a scale vector $s = (s_x, s_y, s_z)$ and rotation quaternions which define the orientation and the stretch of the gaussian in space.

An anisotropic gaussian, unlike isotropic ones that have a spherical shape, has a complete covariance matrix, so its shape can vary in all directions in space. This allows it to orient and stretch freely to fit more easily the geometry to approximate. The parameters describing each gaussian are crucial to obtain an accurate reconstruction of the scene, but only location and color information can be extracted from the SfM-generated point cloud. The remaining parameters must be estimated during the optimization process.

3.2.2 Rasterization

Following the conversion of every point in the SfM sparse point cloud into a three-dimensional anisotropic gaussian, the points must be mapped into two-dimensional space in order to produce images. The process of converting three-dimensional gaussians in space to two-dimensional gaussians in the image plane is known as rasterization. This is an important phase in the pipeline since it produces an image that can be compared to ground-truth to obtain an algorithm assessment measure. This information will be useful during the optimization phase to modify the gaussian parameters appropriately. The rasterization process begins with a frustum culling operation, which is a filtering operation that eliminates all gaussians that do not fall within the camera's field of view and therefore do not contribute to the generation of the current image. This reduces the number of gaussians to be

optimized and thus makes the process more efficient. Next, it is necessary to act on the three-dimensional gaussians, projecting them onto the two-dimensional image plane. To do this, a transformation is applied to the mean and covariance matrix such that the gaussian is mapped in two dimensions. This is done by considering any perspective deformations and trying to maintain the shape and orientation of the primitive as much as possible. The rasterization process, due to computational efficiency issues, does not take place on the entire image dimension, but is done in parallel on small regions of the image. Specifically, the image is divided into tiles of size 16x16 pixels. Each gaussian is identified by a key, characterized by depth information and the identifier of the tile to which it belongs. Next, the gaussians are sorted according to this key and for each tile, a list is generated which contains all the elements that affect that tile in order of depth. The final step is to combine the opacity and color values of the gaussians in the sorted list. This is then performed for each pixel of the corresponding tile, until the entire image is generated. This procedure, known as alpha-blending, produces natural colors and smooth transitions in the finished image. The algorithm in Figure 3.3, taken from the original publication, explains the fundamental steps in the rasterization process.

Algorithm 2 GPU software rasterization of 3D Gaussians
w, h: width and height of the image to rasterize
M, S: Gaussian means and covariances in world space
C, A: Gaussian colors and opacities
V: view configuration of current camera

```
function RASTERIZE(w, h, M, S, C, A, V)
    CullGaussian(p, V)                          ▷ Frustum Culling
    M', S' ← ScreenspaceGaussians(M, S, V)      ▷ Transform
    T ← CreateTiles(w, h)
    L, K ← DuplicateWithKeys(M', T)             ▷ Indices and Keys
    SortByKeys(K, L)                            ▷ Globally Sort
    R ← IdentifyTileRanges(T, K)
    I ← 0                                       ▷ Init Canvas
    for all Tiles t in I do
        for all Pixels i in t do
            r ← GetTileRange(R, t)
            I[i] ← BlendInOrder(i, L, r, K, M', S', C, A)
        end for
    end for
     return I
end function
```

Figure 3.3: 3D Gaussian Splatting rasterization algorithm

3.2.3 Loss Computation

After the two-dimensional image is obtained by gaussian "splatting" and alpha-blending, it is compared with the ground-truth image to check the quality of the reconstruction. The loss function chosen by this algorithm is characterized by the contribution of an L1 loss, combined with a D-SSIM, multiplied by the parameter λ which is set equal to 2 by the original paper. The function is defined by the following formula:

$$L = (1 - \lambda)L_1 + \lambda L_{D-SSIM} \tag{3.1}$$

Specifically, L1 loss, also called Mean Absolute Error (MAE) basically consists of calculating the absolute difference between the values predicted by the model and actual values. In this case, the pixel-to-pixel differences between the generated image and the ground-truth image are computed. In contrast, Dissimilarity SSIM Loss (D-SSIM) is a function that calculates the index of structural dissimilarity between two images, focusing on structural details and contrast. Minimizing this value maximizes the similarity between the two images considered. This combination allows for a good balance between the numerical precison of L1 loss and the visual quality of D-SSIM.

3.2.4 Optimization

The optimization process is based on an adaptive density control of individual gaussians. This enables for better scene representation while keeping the number of gaussians to a minimum in order to prevent consuming too many computational and time resources at this stage. Although the gaussian parameters are adjusted with each iteration, the adaptive density check is performed once per 100 iterations. Pruning and densification are the two basic procedures done on the set of gaussians. Pruning is the elimination of a gaussian whose opacity value is less than a particular threshold. This means that if a gaussian is too transparent to contribute effectively to the final scene generation, it gets eliminated. This improves computing efficiency while maintaining reconstruction quality. The other requirement for excluding a gaussian is its placement in the scene. For example, if it is positioned in a non-visible zone or in an area with uniform elements, such as the sky, it will not help to define the final scene. Densification, on the contrary, allows to increase the

Figure 3.4: Adaptive density control of 3D gaussians

number of gaussians in certain regions of the scene. The basic idea is shown in Figure 3.4, taken from the original paper by Kerbl et al. This is done following a well-defined approach, depending on the presence of either under-reconstructed or over-reconstructed areas. In the case of under-reconstruction, the gaussians are too small to faithfully represent the desired information. This can generate empty areas, not covered by gaussians. In this case, a cloning of a gaussian present in that area is performed, copying all its parameters and placing it in the direction of the gradient. This allows for more accurate reconstruction and correct filling of empty areas. Conversely, over-reconstruction occurs when a gaussian is too large. This may be necessary for the reconstruction of a certain region of the scene but have excessive size. In this case, a split operation is performed to divide the gaussian into smaller gaussians that more faithfully represent the region of interest. A gaussian that is too large, could cover a vast area of the scene without representing its details and small local variations accurately. Renderings characterized by blurred and unsharp surfaces could then be generated, leading to an approximate and poorly detailed representation. The algorithm in Figure, taken from the original publication, explains the fundamental steps in the optimization and densification process.

Algorithm 1 Optimization and Densification
w, h: width and height of the training images

$M \leftarrow$ SfM Points ▷ Positions
$S, C, A \leftarrow$ InitAttributes() ▷ Covariances, Colors, Opacities
$i \leftarrow 0$ ▷ Iteration Count
while not converged **do**
  $V, \hat{I} \leftarrow$ SampleTrainingView() ▷ Camera V and Image
  $I \leftarrow$ Rasterize(M, S, C, A, V) ▷ Alg. 2
  $L \leftarrow Loss(I, \hat{I})$ ▷ Loss
  $M, S, C, A \leftarrow$ Adam(∇L) ▷ Backprop & Step
  if IsRefinementIteration(i) **then**
    for all Gaussians (μ, Σ, c, α) **in** (M, S, C, A) **do**
      if $\alpha < \epsilon$ or IsTooLarge(μ, Σ) **then** ▷ Pruning
        RemoveGaussian()
      end if
      if $\nabla_p L > \tau_p$ **then** ▷ Densification
        if $\|S\| > \tau_S$ **then** ▷ Over-reconstruction
          SplitGaussian(μ, Σ, c, α)
        else ▷ Under-reconstruction
          CloneGaussian(μ, Σ, c, α)
        end if
      end if
    end for
  end if
  $i \leftarrow i + 1$
end while

Figure 3.5: 3D Gaussian Splatting optimization algorithm.

3.3 Evaluation Metrics

In order to evaluate the performance of the 3D Gaussian Splatting algorithm a number of metrics, such as PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index Measure) and LPIPS (Learned Perceptual Image Patch Similarity), must be taken into account [30]. These objectively assess the quality of the reconstruction from a photometric point of view by comparing the generated renders with the ground-truth images given as input to the training pipeline. Specifically, while PSNR evaluates an error at the level of individual pixels, SSIM and LPIPS are metrics that evaluate structural and perceptual features within the images. These three evaluation metrics are presented in detail in the following Sections.

3.3.1 PSNR

PSNR is a metric that is usually employed in image compression contexts and establishes the quality of a reconstructed image, in relation to the original one. Specifically, it is a signal-to-noise ratio that works at the level of individual pixels, so it does not take into account local structure or complex image features. It is based on the use of the (MSE), which allows calculation of the mean square difference between corresponding pixels in the two images. The MSE is defined by the following formula:

$$MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (I(i,j) - \hat{I}(i,j))^2 \tag{3.2}$$

Where $I(i,j)$ and $\hat{I}(i,j)$ are the pixel values of the original image and the reconstructed image, respectively, m and n are the dimensions of the image, so mxn is the total number of pixels. The PSNR uses MSE as the noise metric of the generated image, compared with L representing the maximum value in pixel intensity, which is 255 in the case of 8-bit images per channel. The PSNR formula is defined as follows:

$$PSNR = 10 \cdot \log_{10} \left(\frac{L^2}{MSE} \right) \tag{3.3}$$

It is evident that the MSE value calculated on the generated image heavily affects the final PSNR score. In particular, high MSE, which means that the reconstruction is very different from the ground truth, results in low PSNR. Conversely, when the reconstruction is more faithful to the original, so the MSE is lower, it will result in a PSNR metric with higher values. In general, good PSNR scores are found in the 20-40 dB range. Lower values suggest a poor model reconstruction capability.
An important factor to consider is that this metric, relying solely on the MSE, works

on the intensity difference between pixels of the reconstructed image and pixels of the original image. Therefore, it does not consider local structural components, such as textures, edges, and fine details, which instead play a key role in human perception of the image. This could lead perceptually very different images to have comparable PSNR values. For this reason, this metric is never used individually, but in combination with other metrics that calculate the degree of similarity between images on a perceptual level. In this case, SSIM and LPIPS are used.

3.3.2 SSIM

Structural Similarity Index Measure (SSIM) [31] is a metric to assess the similarity between the reconstructed image and the source image. Unlike PSNR, it is not based solely on the difference between pixels but is sensitive to factors such as luminance, contrast, and image structure that play a key role in human perception. This puts emphasis on the whole image by allowing minor local changes to go unnoticed. SSIM is based on three main components:

- **Luminance (μ)** is a measure of the average light intensity in a given area, which emphasizes the average of the pixel values of the images. It plays an important role in human perception: two images with the same structural characteristics but different luminance are perceived as qualitatively different.
- **Contrast (σ)**is the variation of the pixel values, around the luminance (average) value. Thus, it represents the variance of the pixels. This parameter helps to understand, from a perceptual point of view, how sharp or blurry an image appears;
- **Structure (σ_{x_y})**refers to how the pixels are correlated within the image.

These three factors' contributions are combined to provide the descriptive formula of SSIM:

$$SSIM(I, \hat{I}) = \frac{(2\mu_I\mu_{\hat{I}} + C_1)(2\sigma_{I\hat{I}} + C_2)}{(\mu_I^2 + \mu_{\hat{I}}^2 + C_1)(\sigma_I^2 + \sigma_{\hat{I}}^2 + C_2)} \tag{3.4}$$

This measure can assume values between [-1, 1], where 1 denotes a perfect similarity between the two images, 0 denotes no similarity and hence no correlation between the images, and negative values denote a great dissimilarity.

3.3.3 LPIPS

Learned Perceptual Image Patch Similarity (LPIPS) [32] is an evaluation metric that computes the similarity between two images from a perceptual perspective. To do this, unlike SSIM, it leans on features extracted from convolutional neural

networks which are pre-trained to model human perception of image quality. When it is necessary to compare two images, these are passed into a neural network, usually VGG or AlexNet. The feature maps in each layer of the network are then extracted and compared to each other by calculating the difference between them. Comparing the maps across multiple layers of the network allows local as well as global details of the images to be captured. At this point, the difference is multiplied by a weight that is optimized in training to best reflect human perception. The formula for LPIPS is given by:

$$LPIPS(I, \hat{I}) = \sum_{l} w_l \| f_l(I) - f_l(\hat{I}) \|_2^2 \tag{3.5}$$

where w_l are the weights learned from the network, and $f_l(I)$ and $f_l(\hat{I})$ are the feature maps recovered at level l of the network. The ultimate value of the LPIPS measure typically falls between 0 and 1, where 0 denotes perfect similarity, meaning that there is no difference between the feature maps. Higher values, on the other hand, indicate dissimilarity between the images.

3.4 Addressing Limitations

Through the pipeline previously described, the 3D Gaussian Splatting technique allows for the generation of high-quality real-time renderings using just photos which represent the scene from different perspectives. Despite its many advantages, it has several drawbacks. Scalability comes first. It turns out to be a very good algorithm for reconstructing small-scale scenes, but the problem becomes more complicated when large-scale scenes must be represented. In that case, a substantially larger number of three-dimensional gaussians will be required to describe the scene. Each gaussian, with its parameters, will have to undergo several iterations of optimization, which lead to a very large number of gradient descent operations and, consequently, also to a higher consumption of memory and hardware resources. In addition, the optimization process characterized by pruning and densification, could generate unwanted visual artifacts. Too much pruning may delete information necessary for an accurate reconstruction of the scene, just as poor densification may result in gaussians that are too large or insufficient for describing a specific region. Another very important limitation, especially in the task analyzed by this project where scene geometry plays a key role, is the fact that the reconstructed scene is nothing more than an approximation of the original geometry. Indeed, the algorithm is designed to obtain real-time renderings of the reconstructed scene, but since these are generated through the use of probability distributions, even if approximated through the optimization process, there is no guarantee that the geometric information is faithfully maintained.

To overcome these limitations, recent studies have proposed improvements such as integrating depth and normal maps to the original algorithm. This is the example of the DN-Splatter, proposed by Turkulainen et al [4] in 2025, which will be described in the following Section.

3.5 DN-Splatter: Depth and Normal Supervision

The DN-Splatter algorithm was born with the idea of addressing some of the limitations of 3D Gaussian Splatting, improving the quality of the final reconstruction, not only in terms of rendering, but also from a geometric point of view. Through the use of depth maps, both monocular and obtained through special sensors, and normal maps it seeks to achieve better geometric consistency of the reconstructed scene. This information is also integrated into the optimization process through the use of a special loss function that takes into account not only the photometric factor, but also information about normals and depth.

3.5.1 Normals Estimation

The first innovation introduced by DN-Splatter compared with the original algorithm is the estimation of the normals of the gaussians representing the scene. The normals are calculated using the geometric information of each gaussian, specifically their scale vector $s = (s_x, s_y, s_z)$. This, together with the mean and quaternion of rotation, determines the shape and orientation of the primitive in space. The normal of each gaussian can then be approximated as the direction of the smallest axis of the scale vector. During the optimization process, it is imposed that the gaussians become as flattened as possible along one axis, so that they resemble disks as closely as possible and best represent surfaces. This implies that one of the three scaling axes, will be significantly smaller than the other two. To ensure that this happens, a loss function is used on the scale, defined in this way:

$$L_{\text{scale}} = \sum_i \| \arg\min(s_i) \|_1 \tag{3.6}$$

An additional check is performed during optimization to determine the direction of the normals. As a result of the gaussian parameter update, these could be oriented in the opposite way. The scalar product for each point is then calculated, and if it is negative, indicating that the normal points in the direction of the camera, it is inverted. Furthermore, depth maps are calculated for each image using pre-trained networks such as Omnidata [33] and DSINE [34], beginning with the input images, to ensure that the gaussians are oriented supervisedly. By comparing these ground-truth normal maps to the normals generated by the gaussians, the parameters can be optimized to minimize the difference using a loss function. During this stage,

the L_{normal}, a normal-related loss function, is calculated. This is defined by the contributions of two distinct loss functions and has the following formula:

$$L_{\text{normal}} = L_{\hat{N}} + L_{\text{smooth}} \tag{3.7}$$

Where,

- $L_{\hat{N}}$ is an L1 loss, or Mean Absolute Error (MAE), which determines the distance between normals estimated by gaussian scaling vector ($L_{\hat{N}}$) and ground-truth normals (N) by reducing the noise of the estimates. Its formula is given by:

$$L_{\hat{N}} = \frac{1}{|\hat{N}|} \sum \|\hat{N} - N\|_1 \tag{3.8}$$

- $L_{\textbf{smooth}}$ is a Total Variation (TV) loss function, which is a kind of L1 acting on the differences between neighboring pixels. This dictates that normals change smoothly between adjacent pixels, avoiding abrupt transitions that may indicate errors in reconstruction.

$$L_{\text{smooth}} = \sum_{i,j} \left(|\nabla_i \hat{N}_{i,j}| + |\nabla_j \hat{N}_{i,j}| \right) \tag{3.9}$$

3.5.2 Depth Maps Generation

In the context of DN-Splatter, depth maps play a crucial role in helping the model reconstruct accurate three-dimensional scenes. DN-Splatter can rely on two main sources of depth maps:

- **Monocular-depth:** given input images, maps are estimated from a pre-trained network;
- **Sensor-depth:**they are acquired directly from sensors on the scene, such as LiDAR or RGB-D sensors.

In the case where the dataset under consideration does not contain depth information, pre-trained models, e.g., the ZoeDepth network [35], are used to generate depth maps from the input RGB images. Since the maps generated may have scaling problems with respect to the reconstructed model, it is necessary to align these maps with the depth information extracted from the projection of the SfM points sparse point cloud on the camera.
In the case where scene depth information captured by special sensors is available instead, it is possible to use these maps for more effective training, since they are not predicted by neural networks, but computed directly on the information extracted from the scene to be reconstructed. A depth loss function based on the

gradients of the RGB images is introduced whose goal is to adapt dynamically to the different regions of the scene and their complexity, so as to give more weight to the smoother areas and reduce its influence in the areas with more detail. This is very useful given the poor ability to reconstruct areas without textures by using images alone. In this situation photometric loss may not be as efficient; therefore, depth loss helps in regularization. The function is given by the following formula:

$$L_{\hat{D}} = g_{\text{rgb}} \cdot \frac{1}{|\hat{D}|} \sum \log(1 + \|\hat{D} - D\|_1) \tag{3.10}$$

3.5.3 Loss Functions

The loss function in the DN-Splatter technique differs from the one used in the original 3D Gaussian Splatting version. Its goal is to regularize depth and normals of the scene so that not only the photometric component is considered, but also the geometric one, improving the quality of the reconstruction. Specifically, the loss function is a combination of several components and is defined as follows:

$$L = L_C + \lambda_d L_D + L_{\text{scale}} + (\lambda_n L_N + \lambda_s L_{\text{smooth}}) \tag{3.11}$$

Where,

- $L_{\mathbf{C}}$ (3.1) represents the photometric loss used in the original implementation of 3D Gaussian Splatting , characterized by the combination of L1 loss and LD-SSIM;
- $L_{\mathbf{D}}$ (3.10) represents depth loss function;
- $L_{\mathbf{scale}}$ (3.6) acts directly on the gaussians, forcing them to "flatten" to better represent the scene surfaces;
- $L_{\mathbf{N}}$ (3.8) represents loss on normals;
- $L_{\mathbf{smooth}}$ (3.9) is a smoothness regularization on the normals.

The different λ parameters, on the other hand, are directly selectable in the training phase as needed and control the influence of individual losses on the final optimization. In the reference paper the values are as follows: $\lambda_d = 0.2$, $\lambda_n = 0.1$, $\lambda_s = 0.1$

Chapter 4

Methodology

4.1 Selected Datasets

In order to evaluate the performance of the 3D Gaussian Splatting algorithm for the reconstruction of urban environments, starting from UAV-based imagery, three datasets with different features were used.

Pescara del Tronto. This dataset consists of 125 images at 4000x3000 resolution picturing the Italian town of Pescara del Tronto following the earthquake in August 2016. These images represent a scene characterized by a central urban area surrounded by dense vegetation. Despite the presence of buildings, the number of well-defined edges is limited due to their post-earthquake state. The abundance of debris makes it challenging to identify well-defined key points for the generation of the SfM sparse point cloud. In addition, this dataset does not have ground-truth obtained through LiDAR sensor; therefore, it will not be possible to have a geometric quality metric in this case, but only an assessment of visual quality obtained through renders. Figure 4.1 shows some images of the dataset in question.

Figure 4.1: Example of images in the PDT Dataset.

UseGeo [36][37]. This dataset contains a collection of data including RGB images,

relative depth maps, camera poses and point clouds, both photogrammetric (obtained through MVS) and LiDAR. In total, it contains 829 images at a resolution of (7952x5278), divided into three separate datasets. For the purpose of this project, dataset 1 characterized by 224 images is considered, whose resolution was reduced by half (3976x2639) to ensure a limited training time, to avoid running into gpu saturation problems and to have a better comparison with the previous dataset. The flight specifications are shown in table 4.1.
The scene could be described as divided into two different parts, an urban area characterized by well-defined edges and a surrounding green area. The vegetation, although present, is less invasive compared to the previous dataset. In this case, ground-truth obtained by LiDAR sensor is present, so it is possible to use this information for supervise the training with depth information and evaluate geometric accuracy of reconstruction by difference between generated point cloud and LiDAR-point cloud. In particular, an area rich in buildings will be compared with one predominantly characterized by vegetation.
Examples of images in this dataset are shown in Figure 4.2.

Figure 4.2: Example of images in the UseGeo Dataset.

Hessigheim 3D Geometric Benchmark (H3DG) [38]. This dataset was created in the context of a joint project between Institute for Photogrammetry (Ifp, University of Stuttgart) and the German federal agency of hydrography. The data represent a region of the country of Hessigheim, Germany and are divided into two main blocks: North and South. Specifically, information regarding drone trajectory, exterior information, raw rgb images with a resolution of (14204x10625), and LiDAR mesurment are available. In the context of this project, a subset of RGB images representing a region of the southern block is considered. For computational issues and comparison purposes with other datasets, the source images were rescaled by a factor of 0.28, obtaining a resolution equal to (3977x2982). The area represented is predominantly residential, characterized by the presence of numerous buildings, streets and some garden sections. This allows for a high number of details and key points useful for accurate reconstruction of the sparse point cloud. Unlike the other two datasets, there are no areas characterized by

dense vegetation, at least in the region considered for the project.
In general, the captured images are rich in detail, which can be very useful in the reconstruction phase for a more accurate result. On the other hand, it can also be problematic from a computational point of view: a larger number of gaussians will be needed for scene reconstruction, requiring much more computational resources to carry out the optimization phase.
Examples of images from the H3DG dataset are shown in Figure 4.3.

Figure 4.3: Example of images in the H3DG Dataset.

Dataset	**Sensor**	**Camera**	**Average flight height**	**GDS (Ground sampling distance)**	**LiDAR point cloud density**
UseGeo	RIEGL miniVUX-3UAV scanner	SONY ILCE-7RM3	80 m	2 cm	50 points/m^2
H3DG	RIEGL VUX1LR scanner	Two oblique Sony Alpha 6000	50 m	2-3 cm	400 points/m^2

Table 4.1: UseGeo and H3DG flight specifications.

4.2 Computational Resources

Given the high demand for computational resources needed for this project, a cluster service provided by Politecnico di Torino [39] was employed. Specifically, the Legion cluster was used, characterized by the following technical specifications:

- **Architecture:** Cluster Linux Infiniband-EDR MIMD Distributed Shared-Memory

- **Node interconnection:** Infiniband EDR 100 Gb/s
- **Network Service:** Ethernet 1 Gb/s
- **CPU model:** 2x Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores
- **GPU Node:** 24x nVidia Tesla V100 SXM2 - 32 GB - 5120 cuda cores
- **Performance:** 90 TFLOPS (July 2020)
- **Computational cores:** 1824
- **Number of nodes:** 57
- **Total RAM Memory:** 22 TB DDR4 REGISTERED ECC
- **OS:** CentOS 7.6 - OpenHPC 1.3.8.1
- **Scheduler:** SLURM 18.08

Despite the great potential of this cluster, which still supported an appropriate level of model training, the algorithm codes taken into examination do not appear to be optimized for the use of multiple GPUs in parallel. This has made it necessary to make certain arrangements during the data pre-processing stage, especially since handling hundreds of high-resolution photos is highly demanding. These regulations can be executed even without a powerful cluster, making it possible to apply the methods mentioned above to complete the reconstruction process.

4.3 Software Setup

In order to successfully develop and run this project, it has been necessary to set up a dedicated work environment capable of supporting the organization of the code, the management of external libraries, and the processing of a large amount of data. This Section presents the setup employed, including the development environments and tools for managing computational jobs.

4.3.1 Microsoft Visual Studio

Microsoft Visual Studio was used as the integrated development environment (IDE) for debugging and code management. Its integration with versioning systems, e.g., Git, proved very useful in maintaining an organized workflow. In addition, the use of its particular extension, Desktop Development with C++, provided a robust environment for development. Although the code base is written in the Python language, this extension is necessary because many scientific computing libraries, e.g., PyTorch, OpenCV, TensorFlow contain implementations written in the C++ language for reasons of computational efficiency.

4.3.2 Anaconda Environment

Anaconda is an open-source distribution of Python that allows simplified management of libraries and development environments. In the context of this project, a reduced and lighter version of it, Miniconda, was used directly on Legion clusters. This was used for the creation of an isolated and configurable environment that allowed simple and controlled management of the libraries and dependencies needed to run the 3D Gaussian Splatting code, minimizing any incompatibilities between packages. The environment is generated with the command shown in Listing 4.1, eventually followed by a list of packages that must be present within it. The environment must then be activated and, if necessary, it is possible to continue with the installation of other packages.

```
#create miniconda environment
conda create -n[NameEnvironment] python=3.7
source ~/.bashrc
conda activate [NameEnvironment]

#install addictional packages
conda install -n [NameEnvironment] [package1] ... [packageN]
```

Listing 4.1: Commands to create a Miniconda environment

4.3.3 Singularity

The DN-Splatter algorithm is not executed through the Miniconda Environment system like 3D Gaussian Splatting, but it exploits another approach: a Singularity container [40]. This is a containerization system that is often used in HPC systems because it allows complete isolation of the system, like a small virtual machine. It therefore turns out to be a very secure approach. This solution proved necessary because of the incompatibility of the Open3D library, which is necessary for the code to work properly, with the CentOS 7.6 operating system present on the cluster. The container was created locally using the installation instructions on the DN-Splatter code GitHub page and then uploaded to cluster where it was run as needed. The singularity container is manageable through a series of commands. First, the choice must be made whether to start from a pre-existing container, which can be downloaded via `singularity pull` command, or to create one from scratch. It is important to note that creating a new container requires `sudo (superuser do)` permissions. In this case, the container used was generated locally via the `sudo singularity build` command from a pre-existing container, based on Ubuntu 22.04, equipped with CUDA 11.8 and cuDNN 8, optimized to run on machines with NVIDIA GPUs.

4.3.4 Slurm Scheduling

Simple Linux Utility for Resource Management (SLURM) [41] is a resource management and job scheduling system used in HPC systems. This allows user jobs to be executed and distributed, via queues, in order to optimize the resources required and reduce execution time. Through submission sbatch scripts, it is possible to request the resources needed for a given job, such as number of CPUs, GPUs, memory, and execution time. One of the advantages of using the execution queue is that the job, once scheduled, runs even without an active connection to the cluster, which makes it very useful in the case of very long optimization times. In addition, through a priority management mechanism based on several factors such as queue time and job size in terms of required resources, it is always guaranteed that the scheduled job manages to start in a reasonably short time. In the context of this research project, his method was used to perform the scene reconstruction using both algorithms, 3D Gaussian Splatting and DN-Splatter.
An example of a sbatch script for training using 3D Gaussian Splatting is presented below in Listing 4.2.

```
#SLURM sbatch script for job submission
#!/bin/bash
#SBATCH --job-name=h3dg-base
#SBATCH --mail-type=ALL
#SBATCH --mail-user=s306124@studenti.polito.it
#SBATCH --partition=cuda
#SBATCH --time=5:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=[output-path/output_%x_%j.log]
#SBATCH --error=[output-path/error_%x_%j.log]
#SBATCH --mem=32GB
#SBATCH --gres=gpu:1

#Configure cuda memory allocation:
export PYTORCH_CUDA_ALLOC_CONF=max_split_size_mb:32

#activate conda environment:
source /.bashrc
conda activate gaussian_splatting

#run train script
python train.py
   --source_path data/H3DG-3977x2982
   --model_path output/H3DG-3977x2982/base
   --eval
   --resolution 1
   --data_device cpu
   --test_iterations 30000
```

```
30      --save_iterations 30000
31      --checkpoint_iterations 7000 14000 21000
```

Listing 4.2: SLURM sbatch script for job submission

4.4 Running with Limited Resources

This section presents strategies to be made in case the available computational resources are not sufficient for proper training of the dataset under consideration. 3D reconstruction is a computationally intensive task, requiring the processing of a large amount of data, often at high resolution, and there is not always access to specialized hardware. However, several arrangements can be applied to reduce the computational load and allow the algorithm to operate properly even on less performing hardware. These include downsampling strategies for images or point clouds, specific code optimizations, and the use of HPC server systems.

4.4.1 Image Rescaling

Although it may seem insignificant, lowering the resolution of the RGB images that are being fed into the algorithm is one of the first adjustments that can be made. Excessively high resolution will necessitate a significantly greater number of gaussians, whose optimization stage will certainly affect the GPU state and may result in a CUDA OOM (Out Of Memory) problem. This expedient is far more effective than lowering the quantity of photos provided as input to the system in terms of final reconstruction's quality. Because the reconstruction performed by 3D Gaussian Splatting is passive, i.e., based solely on data extracted from RGB images, the more information you have about the scene from different perspectives, the more details you will capture for the final result.
In the context of this study, image resolution was reduced in both UseGeo and H3DG datasets. Specifically, UseGeo was tested using all 224 source images at full resolution (7952x5278), but the optimization failed due to a CUDA OOM problem. Therefore, it was decided to cut the resolution in half to ensure that the scene could be recreated without any particular issues and with good reconstruction quality, as will be explained in detail in the Results Section (5). As for the H3DG dataset, which is characterized by images at a very high resolution (14204x10625), given the problems encountered in the training of the previous dataset, it was decided to maintain a resolution comparable to the latter, so a downscale to the resolution of (3977x2982) was performed.

4.4.2 Cells Division

In the hypothetical case where there is a need to reconstruct a large-scale scene, a simple downsampling of the images may not be sufficient to be able to finish the gaussian optimization phase without problems. It may happen that, despite the reduction in quality, if the number of images needed to describe the scene turns out to be very large, the available resources may still not be sufficient for accurate reconstruction. To solve this problem, a division of the scene into cells can be made, which will then be optimized separately. At the end of the training pipeline, the resulting point clouds can be merged to obtain a point cloud representing the entire scene, if necessary. During the testing phase, two possible modes of division were considered: one manual and the other automatic, via python scripts.
Manual division is conceptually very simple. Its foundation is the manual division of the photos that will subsequently be sent to the system as input. The number of folders created corresponds to the number of areas into which the source scene is divided. The RGB photos that represent each region are grouped together within each folder. Then, each folder will be treated as a separate dataset and then optimized separately. This method is certainly very simple and intuitive; however, it is not very scalable. In the case of a very large scene, going through a large number of images and dividing them into the relevant folders is definitely not a well-optimized job.
Automatic division by script, on the other hand, does not rely on dividing images, but on dividing the sparse point cloud representative of the entire scene. In order to do this, however, it is necessary to have information about the poses of the camera that acquired each image, in order to be able to determine which images contribute to the generation of a given region of the point cloud. The idea is simple: starting with the sparse representation generated using COLMAP, determine the coordinates of the bounding box that includes the point cloud. At this point, as needed, a division of this area into nxm cells is made. Through the pose information and the coordinates of each cell, it is possible to determine which images contribute to the generation of each cell. Separate folders can then be created, one for each cell, containing the RGB images, the image poses, and the sparse point cloud "clipped" from the sparse point cloud of representing the entire scene. An additional control on the number of images is also inserted, so that each subdataset has a maximum number of images and does not risk encountering OOM problems. This is done in the following way: when the number of images exceeds the predefined constant, the point cloud is further divided into the four parts. This leads to the condition described above, where there are several folders, each representing different regions of the scene to be reconstructed, which can be separately optimized as separate datasets.
Figure 4.4 shows in detail the workflow of automatic division. This method turns

out to be very fast, as it is automatic and especially scalable to large scenes. Moreover, it allows a representation of the final scene to be obtained despite having limited computational capabilities.

Algorithm 1 Point Cloud Division

P: Sparse point cloud
C: Camera poses
ROW: Number of rows in the grid
COL: Number of columns in the grid
IMG: Threshold for maximum images per cell

```
function SplitPointCloud(P, C, ROW, COL, IMG)
    G ← DividePointCloud(P, ROW, COL)                ▷ Split into grid cells
    T ← 0                                            ▷ Initialize total points counter
    for all Cells g in G do
        D_g ← CreateDirectory(g)
        I_g ← FindImagesForCell(g, C)                ▷ Find associated images
        if |I_g| > IMG then
            G' ← SubdivideCell(g, 2, 2)              ▷ Further split the cell
            for all Sub-cells g' in G' do
                D_g' ← CreateDirectory(g')
                I_g' ← FindImagesForCell(g', C)
                SavePoses(I_g', C, D_g')
                CopyImages(I_g', D_g')
                SavePointCloud(g', D_g')
            end for
        else
            SavePoses(I_g, C, D_g)
            CopyImages(I_g, D_g)
            SavePointCloud(g, D_g)
        end if
        T ← T + |g|                                  ▷ Update total points count
    end for
    PrintTotalPoints(T)
end function
```

Figure 4.4: Point cloud division algorithm

4.4.3 Other Suggestions

Speaking specifically of 3D Gaussian Splatting, other possible optimizations to reduce resource consumption can be applied directly in the training phase. By acting directly on the command line arguments in the train.py file, several parameters can be specified that act directly on the number of gaussians generated, their optimization and, consequently, the resources required to finish the process without running into issues. Firstly, a very important argument is `data_device`. Its default value is `cuda`, but as also specified by the authors, in case of datasets characterized by a large number of high-resolution images, it is appropriate to set this value to `cpu`. This is done as a matter of memory management and computational resources during training. Since the memory of the GPU (VRAM) is limited compared to the CPU memory, in the case of very heavy datasets, it is counterproductive to use it to load input data, model variables, and operations necessary for optimization. In the case of high-resolution datasets, it is very easy to saturate VRAM and run into CUDA OOM problems. For this reason, the tendency is to use `data_device` cpu in order to reduce VRAM consumption and leave the GPU "free" to perform

the more complex and intensive computations such as gaussian optimization and rendering. However, this necessarily results in a slowdown of the training phase, since data must be transferred from the cpu to the gpu when needed. As mentioned earlier, a large number of gaussians necessarily requires a large number of operations during optimization, which then becomes computationally challenging. Therefore, it is possible to adjust the parameters that handle the threshold values for the densification operations in order to limit, as much as possible, the total number of gaussians. This of course comes at the expense of reconstruction quality. It would be advisable to find the right compromise between a reasonable amount of gaussians and good reconstruction quality. Some parameters that can be adjusted are:

- **`densify_from_iter`** and **`densify_until_iter`**, which set the interval of densification during training. With a narrower interval, fewer densifications will be performed, so fewer gaussians will be obtained, but also a lower quality of the scene;

- **`densification_interval`** sets the frequency with which the densification process is applied. The default value is 100, which means that this operation is performed every 100 iterations of optimization;

- **`densify_grad_threshold`** sets a minimum limit for the gradient of the 2D position, below which points are not densified. If the gradient is below this threshold, it means that between successive iterations, that particular point has varied little, so it may not be densified. This allows only the points that are actually changing to be densified, optimizing resources. By increasing this value, this also excludes from densification points that have more significant changes between successive iterations, limiting the number of gaussians but also risking reducing reconstruction quality.

4.5 Data Pre-Process

Once the scene to be reconstructed has been chosen, it is not enough to have the RGB images available to start the training process. Certain steps, which will be described in this Section, need to be taken to prepare the dataset for training. The generation of the sparse point cloud and its alignment to the camera poses is a common operation for both algorithms; the generation of normals and depth maps, on the other hand, is related only to DN-Splatter, since 3D Gaussian Splatting does not consider this information during the training phase.

4.5.1 SfM-sparse Alignment with Camera Poses

The first step for proper three-dimensional reconstruction of the scene, whether using 3D Gaussian Splatting or DN-Splatter as an algorithm, is to generate the scattered point cloud through the use of the COLMAP library, as explained in detail in Chapter 3.
Once the original code has been cloned from GitHub, it is necessary to enter the corresponding dn-splatter folder and run the command shown in listing 4.3. This calls the `convert_colmap.py` function, which executes the following commands in order:

- **`colmap feature_extractor`**, which extract keypoints from RGB images in the `<data_root/images>` directory;
- **`colmap exhaustive_matcher`**, which performs matching between features of different images;
- **`colmap mapper`** which performs the bundle adjustment operation.

```
python dn_splatter/scripts/convert_colmap.py
    --image-path [data_root/images]
    --use-gpu

#This function provides the following directories in <data_root>
<data_root>
|---images
|   |---<image 0>
|   |---<image 1>
|   |---...
|---colmap
    |---database.db
    |---sparse
        |---0
            |---cameras.bin
            |---images.bin
            |---points3D.bin
```

Listing 4.3: Convert images to COLMAP format

The `convert_colmap` function, as can be seen from the structure of `<data_root>` directory shown in Listing 4.3, generates four different files. First, a database which stores all the information useful for reconstruction, such as image metadata, extracted features, image matches, and preliminary 3D points. Inside the `<data_root/colmap/sparse/0>` directory, on the other hand, binary information related to sparse reconstruction is stored. Specifically:

- **`cameras.bin`**contains information about the camera used for the reconstruction, thus its intrinsic parameters (model, identifier, focal length, image size).

- **images.bin** contains information about the images and their estimated poses from the reconstruction; it contains the id of the images and the camera that acquired them, position and orientation of the camera, and finally list of associations between two-dimensional features and three-dimensional points.
- **points3D.bin** contains the sparse point cloud and its features such as RGB information and spatial coordinates.

An extra step needs to be taken to ensure that the newly created sparse point cloud is in line with the camera positions if a dataset contains information about the camera poses in addition to the RGB photos that represent the scene. The coordinates of the cameras in this case are in ENU (East-North-Up) format and, in order to be displayed correctly by COLMAP, must have the following format: `image_name.jpg` X Y Z. The files containing the camera poses of the datasets under consideration, have a different format from the required one. These include, in addition to the camera position coordinates, information about the camera rotation angles (omega[deg], phi[deg], kappa[deg]), focal length, main point coordinates, and distortion coefficients. A conversion of this file was therefore necessary in order to transform it into one compatible with COLMAP requirements. To perform the alignment, simply run the command shown in Listing 4.4. In the case of 3D Gaussian Splatting, at this point the dataset can be considered complete, therefore it is ready to start the training phase. Regarding DN-Splatter, additional steps need to be taken, which will be explained in the next Section.

```
colmap model_aligner
   --input_path /path/to/model
   --output_path /path/to/geo-registered-model
   --ref_images_path /path/to/text-file
   --ref_is_gps 0
   --alignment_type custom
   --alignment_max_error 3.0
```

Listing 4.4: COLMAP model aligner

4.5.2 Compute Normal and Depth Maps

In order to perform scene reconstruction using DN-Splatter, it is necessary to have folders containing normal and depth map information. Being able to supervise the optimization step with a ground-truth is essential.
Regarding normals, these are estimated through the use of the DSINE model, which is pre-trained to generate, from each RGB image, the corresponding map of normals. This is done through the following command:

```
#Generate monocular normal estimates for input images
python dn_splatter/scripts/normals_from_pretrain.py
```

```
    --data-dir [data_root]
    --model-type dsine
```

Listing 4.5: Nomal maps generation

These maps are necessary in the training phase to have additional information about the surface to be reconstructed, which is more accurate than the normals estimated directly from gaussians.
The next step is to generate the depth maps. This step may not be necessary if the dataset already has depth maps obtained by sensors during the data acquisition phase. In this case, it is possible to use these sensor-based maps as supervision during training, which are certainly more accurate than the monocular maps extracted from RGB images by neural networks. In the event that a dataset does not have this information, the generation of monocular depths becomes critical. This is done through the command shown in Listing 4.6. Specifically, depth maps calculated from the sparse point cloud are first generated and saved in the `<data_sfm_depths>` directory. Next, monocular depths are computed using the ZoeDepth network, which takes RGB images as input and, for each image, generates the corresponding depth map. As a final step, since the monocular depths often have scaling problems, these are aligned with the sfm_depths in order to ensure geometric consistency between the scene to be reconstructed and the corresponding depth maps and to obtain a more accurate reconstruction.

```
#Converts colmap SfM points to scale aligned mono-depth estimates
python dn_splatter/scripts/align_depth.py
   --data [data_root]
   --sparse-path [data_root/colmap/sparse/0]
   --mono-depth-network zoe
   --align-method grad_descent

#This function provides the following directories in <data_root>
<data_root>
|---image_path
|   |---<image 0>
|   |---<image 1>
|   |---...
|---colmap
    |---database.db
    |---sparse
        |---0
            |---cameras.bin
            |---images.bin
            |---points3D.bin
|---sfm_depths
|   |---<sfm_depth 0>
|   |---<sfm_depth 1>
|   |---...
```

```
|---mono_depth
|   |---<mono_depth 0>.png
|   |---<mono_depth 0>_aligned.npy
|   |---...
```

Listing 4.6: Monocular depths estimation

4.6 Analysis of Training Hyperparameters

As previously mentioned, the DN-Splatter algorithm is a variation of the 3D Gaussian Splatting technique. However, its implementation is based on Nerfstudio's [42][43] version of 3D Gaussian Splatting, Splatfacto, rather than the original code. Because of this, there are a lot more parameters that may be changed during the training phase compared to the original 3DGS code. In particular, it is possible to act on a very large number of parameters related to the optimization of gaussians, colors, opacity, and cameras, but also parameters that handle machine configuration, logging of results, and the viewer for real-time visualization of the training process. In this Section, some parameters related to the optimization pipeline (pipeline.model) are presented in particular. For simplicity of explanation, they can be divided into two categories: parameters related to gaussian number management and parameters related to various loss functions. Regarding the first category, the following parameters are given:

- **`pipeline.model.refine-every`** defines the frequency at which the gaussian refinement process takes place. The default value is 100 iterations. It corresponds to the `densification_interval` parameter used in the implementation of 3D Gaussian Splatting and explained in Section 4.4.3.

- **`pipeline.model.cull-alpha-thresh`** represents the opacity threshold below which gaussians are removed, since they are considered insignificant for the final representation. It is necessary to manage the value well since a threshold too high eliminates a large number of gaussians, losing quality. Conversely, a threshold too low keeps gaussians that do not make a significant contribution to the scene, burdening calculations and memory unnecessarily.

- **`pipeline.model.cull-scale-thresh`** represents the scale threshold, below which gaussians are removed. The concept is similar to the previous parameter, the only difference being that the threshold works on the size of the gaussians and not their opacity.

- **`pipeline.model.cull-screen-size`** defines the minimum threshold (pixel) size of the gaussian on the screen. If the projected gaussian occupies fewer pixels than the threshold, it is deleted.

- **pipeline.model.split-screen-size** represents the minimum threshold below which the gaussian is no longer reduced into smaller gaussians.

Speaking about loss functions, the choice of what kind of function to utilize in order to oversee gaussians and normals is left up to the individual. To balance their contributions during the optimization stage, it is also possible to directly alter the weights, which are the lambda parameters that multiply the loss functions. The following are directives that allow the modification of lambda parameters and the choice of loss functions:

- **pipeline.model.depth-loss-type** permits the selection of the loss type to be applied to depth data. The available losses are: MSE, L1, LogL1, HuberL1, TV, EdgeAwareLogL1, EdgeAwareTV.
- **pipeline.model.smooth-loss-type** permits the selection of the smooth loss type to be applied to normals data. Again, as in the previous case, the losses available are MSE, L1, LogL1, HuberL1, TV, EdgeAwareLogL1, EdgeAwareTV.
- **pipeline.model.ssim-lambda**
- **pipeline.model.sensor-depth-lambda**
- **pipeline.model.mono-depth-lambda**
- **pipeline.model.smooth-loss-lambda**
- **pipeline.model.normal-lambda**
- **pipeline.model.sparse-lambda**

In general, varying these parameters provide different reconstructions not only in quality but also in computational efficiency. It is important to note, however, that an incorrect choice of hyperparameters could generate scenes of inferior quality or even create problems of memory saturation due to an excessive number of gaussians to be optimized. Therefore, it is necessary to find the right balance and modify the parameters judiciously.

4.7 Evaluated Training Configurations

This Section presents the different configurations that were tested, broken down by algorithm. The results of applying these configurations to the three datasets considered will be presented in Chapter 5.

4.7.1 3D Gaussian Splatting

Regarding the 3D Gaussian Splatting algorithm, two configurations were tested.

Base configuration. Characterized by the default values proposed in the original implementation of the algorithm. The only differences from this version appear to be in the `data_device` and `resolution` parameters. The former, which by default contains the value `gpu`, has been assigned the value `cpu`, for the reason explained in Section 4.4.3. When it comes to resolution, the original code stipulates that resolutions greater than 1600 pixels, automatically undergo a reduction to 1600 pixels to avoid getting computationally intensive calculations. In this case, the resolution value has been set to 1, which implies that the original resolution of the input images is used. If set otherwise, the resolution undergoes a downscale by a factor of 1/n, where n is the value passed to the `resolution` parameter. Specifically, the values of the parameters chosen are given below:

```
#Base configuration - 3D Gaussian Splatting
python train.py --source_path [data_root]
   --model_path [output_folder]
   --eval
   --resolution 1
   --data_device cpu
   --sh_degree 3
   --iterations 30000
   --feature_lr 0.0025
   --opacity_lr 0.05
   --scaling_lr 0.005
   --position_lr_max_steps 30000
   --position_lr_init 0.00016
   --position_lr_final 0.0000016
   --densify_from_iter 500
   --densify_until_iter 15000
   --densify_grad_threshold 0.0002
   --densification_interval 100
   --opacity_reset_interval 3000
   --lambda_dssim 0.2
```

Listing 4.7: Base configuration - 3D Gaussian Splatting

High quality configuration. In this arrangement, the parameters that specifically deal with gaussian optimization are changed. The interval and frequency at which densification operations occur are increased by altering the threshold values using the `densify_until_iter` and `densification_interval` arguments. Lowering the settings of `feature_lr`, `opacity_lr`, and `scaling_lr` results in a somewhat slower but more reliable training process. In order to enable the model to concentrate more on the finer points of the reconstruction, the value of `lambda_dssim` is finally adjusted, dropping it slightly from the suggested value.

The values of the parameters used in this configuration are listed below.

```
#High Quality configuration - 3D Gaussian Splatting
python train.py --source_path [data_root]
   --model_path [output_folder]
   --eval
   --resolution 1
   --data_device cpu
   --sh_degree 3
   --iterations 30000
   --feature_lr 0.0020
   --opacity_lr 0.03
   --scaling_lr 0.003
   --position_lr_max_steps 30000
   --position_lr_init 0.00016
   --position_lr_final 0.0000016
   --densify_from_iter 500
   --densify_until_iter 20000
   --densify_grad_threshold 0.0002
   --densification_interval 50
   --opacity_reset_interval 3000
   --lambda_dssim 0.1
```

Listing 4.8: High Quality configuration - 3D Gaussian Splatting

4.7.2 DN-Splatter Algorithm

Concerning the testing phase performed by DN-Splatter algorithm, three different configurations were analyzed, which will be presented below. As the list of possible hyperarameters is very long, for the sake of simplicity and clarity, only the parameters that were changed for each configuration will be reported. Parameters not explicitly discussed are intended to be default values according to the original implementation. The training command starts by executing the singularity container (file with the .sif extension), which contains all the libraries and dependencies necessary for proper code execution. Then the various options related to the dn-splatter module are defined. All four configurations use CoolerMapDataParser (coolermap command) as their dataperser, which is a version that extends ColmapDataParser from Nerfstudio. Its purpose is to process the data obtained from COLMAP, considering the additional information given by the depth and normal maps.

Base. This configuration features all the established defaults proposed by the original implementation. In particular, normals generated by the zoe model and monocular depth maps aligned with those extracted from the SfM reconstruction are used as ground-truth. Two loss functions are then specified, one for depth optimization and one for normals optimization. HuberL1 used for depth information

helps to improve its accuracy by trying to keep the scene noise low and reducing the impact that outliers might have on the optimization. In contrast, TV is a loss function that aims to make the normals estimated from gaussians more uniform, thus trying to achieve more homogeneous surfaces in the final reconstruction. It basically uses the difference between adjacent pixels to penalize local variations. This configuration was also tested, in the case of the UseGeo dataset, by using depth maps extracted directly at the scene during data acquisition as ground truth for supervising depth information.

```
#Base configuration - DN-Splatter
singularity exec --nv /home/crovegno/dns_nerf_1.0.3.sif
ns-train dn-splatter
    --experiment-name h3dg-base
    --data [data_path]
    --output-dir [output_path]
    --vis viewer+tensorboard
    --viewer.quit-on-train-completion True
    --pipeline.model.depth-loss-type HuberL1
    --pipeline.model.smooth-loss-type TV
    --pipeline.model.use-depth-smooth-loss True
    --pipeline.model.use-normal-tv-loss True
    --pipeline.model.mono-depth-lambda 0.2
    --pipeline.model.smooth-loss-lambda 0.1
    --pipeline.model.normal-lambda 0.1
    --pipeline.model.cull-alpha-thresh 0.1
    --pipeline.model.densify-grad-thresh 0.0002
    --pipeline.model.densify-size-thresh 0.01
    --optimizers.means.optimizer.lr 0.00016
coolermap
    --colmap-path  [colmap_path]
    --depths-path [mono_depths_path]
    --load-depths True
    --normal-format opencv
    --normals-from pretrained
```

Listing 4.9: Base configuration - DN-Splatter

Edge-based. The following configuration was tested with the goal of putting the focus of training on edges optimization, since these are very important in the case of urban datasets. In this situation, several values of hyperparameters were changed, starting with the loss functions. In particular, the loss choices try to minimize the difference between estimated information and ground-truths while trying to keep a focus on edges and image details. The EdgeAwareLogL1 works on the depth estimation error, focusing on the edges of the image, using a combination of L1 and logarithm to be less sensitive to large errors. The EdgeAwareTV on the other hand tries to improve estimates on normals, focusing on changes in normals along significant edges of the scene.

```
#Edge-based configuration - DN-Splatter
singularity exec --nv /home/crovegno/dns_nerf_1.0.3.sif
ns-train dn-splatter
    --experiment-name h3dg-edges
    --data [data_path]
    --output-dir [output_path]
    --vis viewer+tensorboard
    --viewer.quit-on-train-completion True
    --pipeline.model.depth-loss-type EdgeAwareLogL1
    --pipeline.model.smooth-loss-type EdgeAwareTV
    --pipeline.model.use-depth-smooth-loss True
    --pipeline.model.use-normal-tv-loss True
    --pipeline.model.mono-depth-lambda 0.2
    --pipeline.model.smooth-loss-lambda 0.1
    --pipeline.model.normal-lambda 0.1
    --pipeline.model.cull-alpha-thresh 0.1
    --pipeline.model.densify-grad-thresh 0.0002
    --pipeline.model.densify-size-thresh 0.01
    --optimizers.means.optimizer.lr 0.00016
coolermap
    --colmap-path  [colmap_path]
    --depths-path [mono_depths_path]
    --load-depths True
    --normal-format opencv
    --normals-from pretrained
```

Listing 4.10: Edeg-based configuration - DN-Splatter

Tuning. In this configuration, the goal was to play with the different parameters, trying to obtain a higher quality reconstruction without affecting too much the total number of gaussians and thus the computational resources required. More importance is given to the contribution of depth, increasing the lambda parameter of the monocular depths with the goal of better aligning the estimated surfaces with the real ones. The contribution of ssim is also increased in order to improve the perceptual quality of the scene. The opacity threshold below which gaussians are discarded is lowered; thus, a higher number of gaussians will contribute to the final reconstruction.

```
#Tuning configuration - DN-Splatter
singularity exec --nv /home/crovegno/dns_nerf_1.0.3.sif
ns-train dn-splatter
    --experiment-name h3dg-tuning
    --data [data_path]
    --output-dir [output_path]
    --vis viewer+tensorboard
    --viewer.quit-on-train-completion True
    --pipeline.model.depth-loss-type EdgeAwareLogL1
    --pipeline.model.smooth-loss-type EdgeAwareTV
    --pipeline.model.use-depth-smooth-loss True
```

```
    --pipeline.model.use-normal-tv-loss True
    --pipeline.model.mono-depth-lambda 0.3
    --pipeline.model.ssim-lambda 0.3
    --pipeline.model.smooth-loss-lambda 0.2
    --pipeline.model.normal-lambda 0.1
    --pipeline.model.cull-alpha-thresh 0.05
    --pipeline.model.densify-grad-thresh 0.0002
    --pipeline.model.densify-size-thresh 0.01
    --optimizers.means.optimizer.lr 0.00012
coolermap
    --colmap-path  [colmap_path]
    --depths-path [mono_depths_path]
    --load-depths True
    --normal-format opencv
    --normals-from pretrained
```

Listing 4.11: Tuning configuration - DN-Splatter

This configuration proved to be problematic when training the H3DG dataset. Since the scene to be represented is rich in detail, a much higher number of gaussians is generated than in the other datasets, as will be shown in the Results Section. This, combined with the fact that this configuration tries to increase the number of gaussians in order to obtain a higher quality representation, caused CUDA OOM problems. It was therefore necessary, exclusively for this dataset, to think of another training configuration, with the opposite goal of decreasing the number of generated gaussians. The final configuration is shown below:

```
#Tuning configuration for H3DG dataset - DN-Splatter
singularity exec --nv /home/crovegno/dns_nerf_1.0.3.sif
ns-train dn-splatter
    --experiment-name h3dg-tuning
    --data [data_path]
    --output-dir [output_path]
    --vis viewer+tensorboard
    --viewer.quit-on-train-completion True
    --pipeline.model.depth-loss-type EdgeAwareLogL1
    --pipeline.model.smooth-loss-type EdgeAwareTV
    --pipeline.model.use-depth-smooth-loss True
    --pipeline.model.use-normal-tv-loss True
    --pipeline.model.mono-depth-lambda 0.2
    --pipeline.model.ssim-lambda 0.2
    --pipeline.model.smooth-loss-lambda 0.1
    --pipeline.model.normal-lambda 0.05
    --pipeline.model.cull-alpha-thresh 0.13
    --pipeline.model.densify-grad-thresh 0.0005
    --pipeline.model.densify-size-thresh 0.02
    --optimizers.means.optimizer.lr 0.00014
coolermap
    --colmap-path  [colmap_path]
```

```
    --depths-path [mono_depths_path]
    --load-depths True
    --normal-format opencv
    --normals-from pretrained
```

Listing 4.12: Tuning configuration for H3DG dataset - DN-Splatter

4.8 Renders and Point Cloud Extraction

Following the training procedure, the next stage is based on the outcomes' extraction and subsequent analysis, which are essential steps to assess the accuracy of the configurations that were just tested. After training, the DN-Splatter algorithm creates an output directory with the following definition:

```
#DN-Splatter output directory
<output_path>
|---<experiment_name>
    |---<dn-splatter>
        |---<timestramp>
            |---<nerfstudio_models>
            |   |---step-000029999.ckpt
            |---config.yml
            |---dataparser_transforms.json
            |---events.out.tfevents
```

Listing 4.13: DN-Splatter output directory

where,

- **`config.yml`** contains information regarding the configuration of the model and all its parameters, including the paths to all folders necessary to resume training from a given checkpoint, if necessary;
- **`dataparser_transform.json`** contains a transformation matrix describing the orientation of the scene in the camera system and a scaling factor.
- **`events.out.tfevents`** is the file that contains the Tensorboard logs that allow us to check in real time how the training is progressing by analyzing the performance of loss and evaluation metrics.

The photometric quality of the reconstruction can be seen directly through the support of Tensorboard, if enabled during the training phase. The same applies to the final scene reconstruction: this can be viewed either in real time or at the end of the optimization phase via the viewer made available by nerfstudio. In case it is necessary to download the renders at the end of the training, they can be generated and exported manually using the appropriate command:

```
#Renders extraction and metrics computation
ns-eval
    --load-config [config_path]
    --output-path [output_path]
    --render-output-path [renders_path]
```

Listing 4.14: Renders extraction and metrics computation

The same applies to the final point cloud. In case it is necessary to perform post-processing operations on it or to compare it with a LiDAR reference to obtain a geometric measure of the reconstruction, it should be exported via the corresponding command:

```
#Point cloud export
ns-export gaussian-splat
    --load-config [config_path]
    --output-dir [output_path]
```

Listing 4.15: Point cloud export

4.9 Point cloud Post-Process

Frequently it happens that the point cloud resulting from the training phase of the algorithm is not the final result desired, but rather becomes the input for other operations. Therefore, it is often necessary to submit this data to a post-processing phase that prepares it for this purpose.
In the context of this project, the resulting point cloud must be compared with the point cloud obtained by LiDAR sensor to calculate the geometric accuracy of the reconstruction. This is done through the use of CloudCompare software, which allows point cloud processing in an easy but effective way. The point clouds obtained as a result of the training, before being imported to CloudCompare for post-processing operations, undergo a conversion performed through 3dgsconverter [44]. This operation is not strictly necessary, since the point cloud is still read correctly by the software, but since the software does not support gaussian visualization, without the conversion much information related to colors, described by SH coefficients, and opacity values is lost.
Four postprocessing operations performed on the resulting point clouds are presented in this Section.

4.9.1 Alignment

The manual point cloud alignment operation is not always necessary, but there are cases when it proves to be really important. An example might be the case where one does not have information about camera poses available for a point cloud

alignment using `colmap model_aligner`. Or situations where not enough hardware resources are available for training the complete dataset and it is necessary to apply the cell partitioning operations as described in Section 4.4.2. In this case, the only way to realign the different cells is to perform a manual alignment to recreate the point cloud representing the entire scene. In the context of this project, this operation was performed in particular in the case of the Pescara del Tronto dataset. In fact, in addition to a total optimization of the dataset in its entirety by using resources offered by the cluster, the solution of dividing it into cells for training with limited resources was also tested. Figure 4.5 shows the three individually

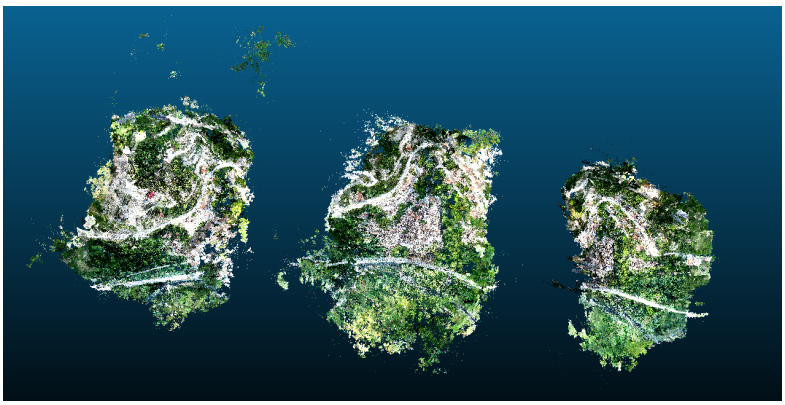

Figure 4.5: Pescara del Tronto dataset divided into three cells, optimized separately, before the alignment phase using Align Tool.

trained cells of the dataset under consideration, prior to manual alignment. The cell alignment was done using CloudCompare and consists of two basic steps: first a approximate alignment is done using Align Tool, then refined using Iterative Closest Point (ICP).
The initial alignment using Align Tool allows for a rough alignment between the two point clouds by manually selecting at least three points common to the two point clouds. Later, this alignment can be improved using ICP. This allows, through a series of iterations, to minimize the distance between the two point clouds by finding the best combination of rotation and translation operations. Once the alignment of the point clouds is done, a merge can be performed so that they are joined and the point cloud representing the entire scene is obtained.

4.9.2 Outliers Removal

Another very important operation is the management of outliers. These are characterized by points that deviate from the main point cloud, compromising the result of any subsequent operations such as LiDAR ground-truth distance

calculation. They can result from errors during the processing phase, data noise or lack of a significant number of data points to represent a certain region of the scene. In particular, this occurs near the outside margins of the point cloud, where fewer images are taken from different angles, giving the model less information to use for precise scene reconstruction.
Figure 4.6 shows the point cloud for the PDT dataset, obtained by DN-Splatter algorithm. As can be seen, the areas circled in red represent outliers, groups of points that deviate from the distribution of interest and are likely to create problems in subsequent operations. A manual cleanup was then done, using CloudCompare's Segment Tool, to remove these areas and obtain a more compact point cloud. Figure 4.7 shows the final result.

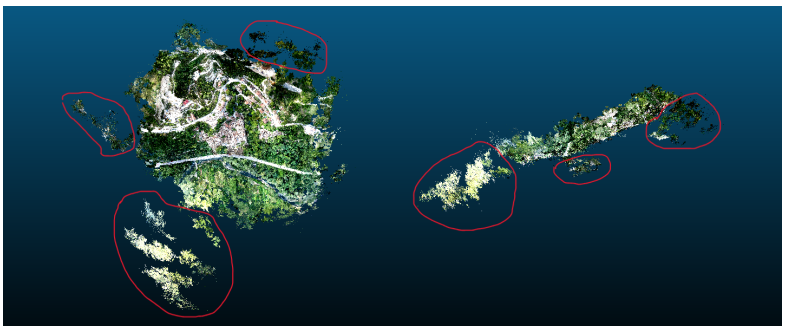

Figure 4.6: DN-Splatter point cloud, displayed in CloudCompare. On the left, the top view of the scene is shown, on the right the side view of the same scene. Outliers, circled in red, are highlighted.

Figure 4.7: DN-Splatter point cloud, following the clean-up operation using Segmentation Tool.

4.9.3 Downsampling

Downsampling is an operation that allows the reduction of the number of points that belong to a point cloud, in order to make processing easier, but without losing too much in terms of quality. This, can prove very useful when working for example with LiDAR data, characterized by millions of points. In this context, in order to have an easier data handling during the post-processing phase, a downsampling was performed on the point cloud obtained from the LiDAR sensor of the H3DG dataset, which was very heavy. This downsampling was necessary because the point cloud obtained from the reconstruction algorithm is much less detailed than the LiDAR ground-truth. As a result, the two relevant data points can be compared more fairly. Using a technique based on spatial sampling, which measures the distance between individual points, the downsampling was performed using CloudCompare's Subsampling tool. By setting a value of 0.02 m for example, one point is retained every 2 cm. This method, compared to random sampling, maintains a more uniform distribution of points, reducing density without losing too much structural detail.

4.9.4 Point-to-Point Distance

The last post-processing operation applied to the point cloud resulting from the reconstruction algorithm is the calculation of the point-to-point distance with the LiDAR point cloud, representing the grond-truth. This step is necessary because the ultimate goal of this project is precisely to evaluate the ability of the algorithm to generate an accurate reconstruction, not only photometrically but also geometrically. The distance is calculated by taking the ground-truth point cloud as a reference and setting the reconstructed point cloud as the target. Then, using a nearest neighbor search algorithm, the nearest points belonging to the target cloud are searched for each point in the LiDAR. The Euclidean distance between these points is then computed. At the end of the process, the result can be visualized in the form of a color scale on the target cloud, where the color changes indicate the difference in distance. In this case, since the outer areas of the reconstructed point cloud, due to the limited number of data covering those areas, are poorly defined, only the central part is considered for distance calculation.

Chapter 5

Results and Discussions

This Section will present and discuss the results obtained during the course of this thesis project, through the algorithms and approaches described in the previous chapters. In particular, the analysis will be of two types, photometric and geometric. The former focuses on the visual evaluation of reconstructed scenes, through the analysis of evaluation metrics; the latter emphasizes the geometric accuracy of the point clouds reconstructed by the models, in relation to the LiDAR point cloud obtained from the sensor during data acquisition.

5.1 Photometric Analysis

When reconstructing a real scene, photometric analysis of the generated environment represents a fundamental aspect of the whole process. This is because the human brain is naturally inclined to recognize real images; therefore, it is particularly sensitive to noise or artifacts in reconstruction. Inconsistent colors, poorly defined textures or errors in lighting have a very strong visual impact, immediately giving the impression of a generated image, different from the real one.
In order to evaluate the photometric accuracy of the reconstruction, the evaluation metrics proposed by the original implementation of the 3D Gaussian Splatting algorithm have been considered: PSNR, SSIM, LPIPS. Table 5.1 shows the results obtained on the three datasets considered, trained by 3D Gaussian Splatting, in the configurations presented in Section 4.7.1. From these data, it can be noted immediately that, unlike what might be expected, a higher number of gaussians does not necessarily lead to an increase in reconstruction quality. In fact, the High-Quality configuration, despite having a much higher number of gaussians than the basic configuration, as can be seen from the G-Count field in the table, achieves almost comparable results in the evaluation metrics. In contrast, a more substantial difference can be seen by comparing the performance of the algorithm

Dataset	Config.	PSNR	SSIM	LPIPS	G-Count
PDT	Base	**24,9772**	**0,7311**	**0,3181**	8.099.123
	Hq	24,5958	0,7066	0,3316	7.561.579
D1	Base	**26,8579**	**0,8198**	**0,2498**	12.490.019
	Hq	26,4887	0,7998	0,2916	11.080.751
C1-H3DG	Base	**25,9121**	**0,8521**	**0,2984**	13.412.243
	Hq	25,6817	0,8336	0,3002	12.611.021
C2-H3DG	Base	**26,2762**	**0,8909**	**0,2704**	13.104.205
	Hq	26,0723	0,8738	0,2981	12.206.544

Table 5.1: 3D Gaussian Splatting photometric results

on the three datasets examined. In particular, the dataset representing Pescara del Tronto turns out to have the lowest values. This is certainly related to the features of the images that compose it. In fact, the scene is characterized by a vast green area, which is almost devoid of key points with which to perform an accurate sparse reconstruction. In addition, the central region of the scene, despite being characterized by an urban area, is also particularly complex to reconstruct because of the condition of the buildings following the earthquake. Rubble and debris create a highly variable area that is difficult to reproduce correctly during the gaussian optimization phase. This generates poor PSNR, SSIM and LPIPS values, describing the fact that the generated renders are deficient both at pixels level and at perceptual level.
The other two datasets analyzed, UseGeo and H3DG, as can be seen from the values shown in Table 5.1, are more accurately represented. This again, as explained above for the PDT dataset, depends on the properties of the ground-truth images

Figure 5.1: Some extracts from the reconstruction of the Pescara del Tronto dataset.

that compose it. Both datasets are characterized by urban landscapes, rich in detail and well-defined edges. This aspect greatly helps the process of finding and matching features to create the sparse point cloud from which the optimization process starts. Consequently, it proves easier to optimize gaussians for a more accurate representation of the scene, confirmed by the better values of evaluation metrics.

Dataset	Cell	PSNR	SSIM	LPIPS	G-Count
PDT	C1	22,8397	0,6204	0,4534	5.298.320
	C2	**23,4572**	**0,6796**	**0,3841**	4.919.338
	C3	23,4422	0,6712	0,3875	6.353.678
D1	C1	**26,8480**	0,7965	**0,2422**	3.910.713
	C2	23,4682	0,7430	0,2686	5.536.776
	C3	22,3885	0,7063	0,3459	2.410.307
	C4	24,6784	**0,8053**	0,2470	4.102.176
	C5	25,0813	0,7629	0,3032	2.364.862
	C6	26,0062	0,7927	0,2622	4.976.818
	C7	23,0083	0,7684	0,2445	5.654.048

Table 5.2: 3D Gaussian Splatting photometric results for individual cell using base configuration

Taking a closer look, it can be seen that different regions of the same scene produce renders of different quality. Table 5.2 shows the results obtained on the PDT and UseGeo datasets, divided into cells as described in Section 4.4.2, , to try coping with the problem of limited computational resources. The obtained metrics show how the quality of the generated renders varies significantly depending on the conformation of the represented territory. For example, in the UseGeo dataset, the most distant values are represented by cell number 3, characterized almost exclusively by vegetation, and cell number 1, rich in detail due to the presence of buildings and roads with well-defined edges. A representation of the two cells can be seen in Figure 5.2.
Regarding the DN-Splatter algorithm, the tests conducted on the three datasets under consideration, using the configurations described in Section 4.7.2, are shown in Table 5.3. Again, similar to previous results, there is not much difference among different configurations for the same dataset. In general, it can be seen that the basic configuration achieves better results in terms of evaluation metrics. It is also interesting to note that, in contrast to the approach based on the implementation of 3D Gaussian Splatting, the number of gaussians used for reconstruction is significantly lower, while still maintaining the photometric quality and, in some cases, improving it. As in the previous case, the best performance is on the UseGeo

Figure 5.2: On the left, the reconstruction of C3,largely covered in vegetation; on the right, the reconstruction of C1 characterized by a high number of details.

Dataset	Config.	PSNR	SSIM	LPIPS	G-Count
PDT	Base	**27,8767**	**0,8454**	**0,2338**	3.703.461
	Edges	25,3759	0,7673	0,3847	2.396.815
	Tuning	26,4337	0,8037	0,3102	4.191.022
D1	Base	27,8504	0,7650	0,2787	3.771.038
	B-Sensor	**27,9094**	0,7646	0,2768	3.790.209
	Edges	27,2715	0,7458	0,2962	3.455.903
	Tuning	27,7093	**0,7825**	**0,2487**	6.677.079
C1-H3DG	Base	**22,6333**	**0,6648**	**0,3607**	9.366.555
	Edges	22,1787	0,6325	0,3958	7.263.430
	Tuning	21,5394	0,6048	0,4328	5.619.977
C2-H3DG	Base	**23,2432**	**0,7195**	**0,3007**	9.019.811
	Edges	22,6004	0,6729	0,3515	7.263.430
	Tuning	22,2445	0,6669	0,3653	5.619.977

Table 5.3: DN-Splatter photometric results

dataset. On this very dataset, a test was also performed using depth maps extracted from LiDAR data as ground-truth for depth information. However, at least at the photometric level, there is not much difference from the basic configuration. The H3DG dataset, on the other hand, gets lower quality results. This is probably due to the strong contrast given by the light/shadow areas. These sudden changes in illumination probably create artifacts during optimization, affecting the overall quality of the reconstruction.
Another very important aspect to consider is that both models work from input images, so they can only use the information they have available as ground-truth data for the optimization phase. In the case of data acquired by UAV from above,

Figure 5.3: Some extracts from the reconstruction of the H3DG dataset.

they do not have side perspectives of the buildings or regions to be reconstructed. This generates a final reconstruction that is very detailed, from a photometric point of view, in certain areas, but equally deficient in others. As can be seen from Figure 5.4, the output is very good from an aerial view, but extremely undefined from a perspective which is different from the input images. The model itself is capable of reconstructing new views not present in the input data by interpolating among those available, but in the case of aerial images this is extremely complicated. In fact, the images on which the original algorithm was tested are all characterized by one key aspect: they are images that " rotate" around the subject, capturing it from every perspective. In this way, a good overall result can be achieved. Aerial images, on the other hand, capture the environment only from above, limiting the model's ability to correctly reconstruct different perspectives.
Concluding this photometric analysis, it can be said that, in general, the quality of the results is not excellent, but still satisfactory, and thus the model is able

Figure 5.4: On the left, a side view of the reconstructed environment; on the right, an aerial view of the same region.

to represent even large-scale aerial scenes in a photometrically accurate manner. However, attention should be paid to the region being reconstructed, the quality of the images being acquired, and keep in mind that since this is aerial data, viewing the reconstruction from different perspectives could be challenging.

5.2 Geometric Analysis

The goal of the 3D Gaussian Splatting algorithm studied in this thesis project is to recreate urban environments using only RGB images as starting data. As was described in the previous Section, it has been shown to work quite well even on images different from those for which it was designed, such as aerial images that allow reconstructing large-scale environments from a different perspective. Unfortunately, however, the reconstruction in this case has a purely aesthetic purpose, therefore the geometric accuracy of the generated point cloud is not taken into account. This may be acceptable in situations where the main focus falls on visual appearance, such as reconstructing environments for virtual reality applications. However, in contexts that require some geometric accuracy, such as mapping urban environments, this aspect must also be taken into account.
This Section therefore evaluates geometrically the point clouds generated by both algorithms in the different configurations tested. This is useful not only to evaluate the geometric accuracy of 3D Gaussian Splatting, but also to test whether DN-Splatter actually succeeds in solving this problem by generating more accurate clouds. Geometric accuracy was evaluated by calculating the distance between the point cloud obtained by LiDAR sensor, for datasets that have such information, and the point cloud generated by the reference algorithm. Specifically, results will be presented using two types of values: the average distance (expressed in cm) between the LiDAR points and the points in the generated cloud, and the percentage of points in the generated cloud that are below a reference threshold, both in terms of global distance and in terms of distance over the plane. For the calculation of C2C (Cloud to Cloud) distance using the CloudCompare software, a maximum threshold distance between points of 1m was used. This threshold value is very high, but it was necessary for proper distance evaluation given the low accuracy of the generated point clouds.
The point cloud generated by the 3D Gaussian Splatting algorithm turns out to be rich in outliers, which cannot always be removed by clean-up operation before distance calculation. Especially, these clusters of points develop in height, going to affect the global distance calculation. For this reason, the distance calculated in the XY plane is also considered, which ignores the Z component, thus the elevation difference. In fact, since the scene is taken exclusively from aerial images, this data is not as important as in other contexts. The geometric results calculated on the

Dataset	Config.	3DGS - C2C Absolute Distance - Global							
		5 cm	**10 cm**	**20 cm**	**50 cm**	**80 cm**	**#gauss.**	**Mean**	**Std.Dev.**
D1	Base	20,81%	32,74%	45,69%	60,03%	63,83%	12.499.019	46,2 cm	42,4 cm
	HQ	19,34%	30,52%	42,43%	55,24%	58,81%	11.080.751	50,4 cm	43,3 cm
D1 Central	Base	20,39%	35,08%	51,48%	68,64%	-	5.251.163	25,0 cm	19,5 cm
	HQ	19,26%	33,41%	49,99%	68,24%	-	4.391.684	25,5 cm	19,4 cm
C1 H3DG	Base	15,43%	26,12%	40,76%	62,04%	69,81%	13.412.243	46,4 cm	41,2 cm
	HQ	14,29%	25,81%	39,18%	60,02%	66,91%	12.611.021	48,3 cm	43,1 cm
C2 H3DG	Base	12,56%	22,58%	36,87%	59,04%	65,90%	13.104.205	48,6 cm	39,6 cm
	HQ	11,64%	21,12%	34,92%	58,35%	62,13%	12.206.544	51,2 cm	40,3 cm

Table 5.4: The percentage of gaussians in the point cloud produced by 3DGS that are below a specific distance threshold in comparison to LiDAR is displayed in the table for each tested configuration. The mean and standard deviation global distance values, both in centimetres, are also displayed.

Dataset	Config.	3DGS - C2C Absolute Distance - Planar							
		5 cm	**10 cm**	**20 cm**	**50 cm**	**80 cm**	**#gauss.**	**Mean**	**Std.Dev.**
D1	Base	38,41%	57,52%	77,73%	97,25%	99,75%	12.499.019	13,0 cm	14,1 cm
	HQ	38,97%	58,44%	78,18%	97,00%	99,70%	11.080.751	12,9 cm	14,4 cm
D1 Central	Base	41,85%	66,04%	88,64%	100%	-	5.251.163	9,18 cm	8,61 cm
	HQ	40,53%	64,63%	87,83%	100%	-	4.391.684	9,47 cm	8,79 cm
C1 H3DG	Base	44,19%	65,51%	87,12%	99,03%	97,97%	13.412.243	10,4 cm	11,7 cm
	HQ	43,92%	65,03%	86,70%	97,31%	99,72%	12.611.021	10,6 cm	11,9 cm
C2 H3DG	Base	42,10%	64,04%	84,57%	98,13%	99,83%	13.104.205	10,8 cm	12,2 cm
	HQ	41,23%	62,81%	84,10%	97,75%	99,61%	12.206.544	11,3 cm	12,4 cm

Table 5.5: The percentage of gaussians in the point cloud produced by 3DGS that are below a specific distance threshold in comparison to LiDAR is displayed in the table for each tested configuration. The mean and standard deviation planar distance values, both in centimetres, are also displayed.

point clouds generated by the 3D Gaussian Splatting algorithm are presented in Table 5.4 and Table 5.5.
The generated point clouds, as can be seen, are not very accurate, with a very low percentage of points being below 10 cm in distance, especially for global distance. On the other hand, excluding the Z component and calculating the distance over the plane, the values are better, with more than 95 percent of points being at a maximum distance of 50 cm. In the particular case of the UseGeo dataset, given

the large difference from the LiDAR reference, a central region of the scene was extracted that appeared to be the most accurate. This was evaluated separately by setting a maximum distance threshold lower than the total point cloud of 50 cm. It can be seen, however, that the results, at least as far as global distance is concerned, are comparable with those extracted from the total point cloud. Speaking instead of distance on the plane, the values are slightly better, with 100% of the points below 50 cm distance. The average distance values, expressed in centimeters, and the standard deviation values for each point cloud evaluated, both for global and planar distances, are also shown in Table 5.4 and Table 5.5 .
It can therefore be stated that the geometric accuracy in the 3D Gaussian Splatting algorithm could be improved, given the results on the different datasets. Figure 5.5 shows the point clouds representing the distances, global and planar, calculated with respect to the ground-truth LiDAR cloud and particularly the point cloud representing the central cell of the UseGeo dataset, in its basic configuration.

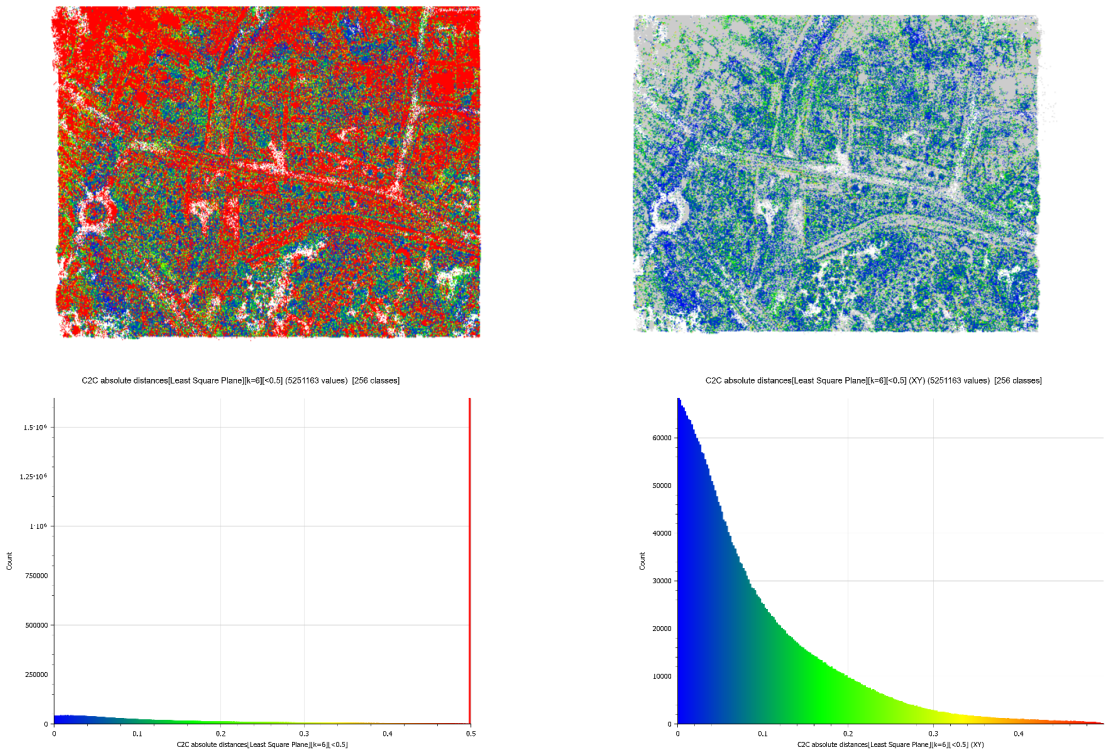

Figure 5.5: On the left, the point cloud representing the global distance between the LiDAR reference and the generated point cloud, with the relative distribution of points on the different distance values; on the right, the same representation but related to the planar distance.

The DN-Splatter algorithm works on this limitation of the original implementation by introducing the possibility of computing normal and depth maps to improve the geometric accuracy of the generated point cloud. The results are generally better than those obtained using the original 3D Gaussian Splatting implementation, shown in Table 5.4 and Table 5.5. Again, global and planar distance values are shown, with an improvement in the case of planar distance. Table 5.6 and Table 5.7 show the distance-related results obtained by comparing the LiDAR point cloud with the point cloud generated by DN-Splatter. The values for the different configurations tested have a rather similar trend for the same dataset. However, it can be seen that the UseGeo dataset has better results than the H3DG dataset, on both blocks considered (C1 and C2). This is due to the fact that the point cloud generated for this dataset is not particularly accurate, especially the one representing block C1, which has worse distance values compared to block C2.
In particular, as can be seen from Figure 5.6, the point clouds appear to have several outliers, especially in height at the roofs of buildings. This could be due to the strong changes in brightness given by the light/dark areas, which are particularly pronounced in some regions of the scene. If highly emphasized, these could be read as depth fluctuations, where the bright parts are higher and the dark areas are lower.

Dataset	**Config.**	**DNS - C2C Absolute Distance - Global**							
		5 cm	**10 cm**	**20 cm**	**50 cm**	**80 cm**	**#gauss.**	**Mean**	**Std.Dev.**
D1	Sensor	27,26%	42,31%	60,71%	80,44%	85,44%	5.073.679	28,4 cm	33,0 cm
	Base	26.22%	43.12%	61.48%	80.66%	85,44%	5.063.774	28,0 cm	33,1 cm
	Edges	28,85%	42,43%	60,49%	79,28%	83,93%	4.644.470	29,2 cm	34,1 cm
	Tuning	26,21%	42,80%	60,75%	79,57%	83,96%	9.248.330	29,1 cm	34,1 cm
D1 Central	Sensor	29,84%	48,26%	65,65%	76,24%	-	1.586.771	19,4 cm	18,8 cm
	Base	32,34%	51,20%	67,96%	77,57%	-	1.639.654	18,5 cm	18,5 cm
	Edges	31,82%	50,32%	66,83%	76,27%	-	1.476.930	19,0 cm	18,9 cm
	Tuning	32,02%	50,36%	66,56%	75,77%	-	3.053.343	19,1 cm	19,0 cm
C1 H3DG	Base	19,22%	34,61%	56,56%	83,42%	92,04%	8.883.937	26,5 cm	27,2 cm
	Edges	19,39%	34,86%	56,61%	82,82%	91,50%	7.313.995	26,8 cm	27,7 cm
	Tuning	19,32%	35,15%	57,50%	84,71%	92,95%	5.197.802	25,6 cm	26,3 cm
C2 H3DG	Base	18,70%	34,60%	55,46%	91,33%	95,81%	8.294.564	23,0 cm	21,9 cm
	Edges	18,49%	32,82%	53,65%	87,93%	92,05%	6.724.556	25,8 cm	26,2 cm
	Tuning	18,76%	33,63%	55,52%	93,46%	96.78%	5.306.580	22,0 cm	20,2 cm

Table 5.6: The percentage of gaussians in the point cloud produced by 3DGS that are below a specific distance threshold in comparison to LiDAR is displayed in the table for each tested configuration. The mean and standard deviation global distance values, both in centimetres, are also displayed.

Dataset	Config.	DNS - C2C Absolute Distance - Planar (XY)							
		5 cm	**10 cm**	**20 cm**	**50 cm**	**80 cm**	**#gauss.**	**Mean**	**Std.Dev.**
D1	Sensor	36,49%	56,49%	78,11%	97,96%	99,80%	5.073.679	12,8 cm	13,3 cm
	Base	37,24%	57,55%	78,93%	98,06%	99,82%	5.063.774	12,5 cm	13,1 cm
	Edges	37,32%	57,58%	79,00%	98,09%	99,83%	4.644.470	12,4 cm	13,1 cm
	Tuning	37,60%	57,84%	79,77%	98,21%	99,84%	9.248.330	12,3 cm	12,9 cm
D1 Central	Sensor	49,69%	73,41%	93,29%	100%	-	1.586.771	7,41 cm	7,12 cm
	Base	50,63%	74,53%	93,69%	100%	-	1.639.654	7,22 cm	6,97 cm
	Edges	50,68%	74,48%	93,68%	100%	-	1.476.930	7,24 cm	7,01 cm
	Tuning	51,09%	74,86%	93,95%	100%	-	3.053.343	7,15 cm	6,89 cm
C1 H3DG	Base	36,57%	57,19%	79,14%	97,26%	99.78%	8.883.937	12,8 cm	13,7 cm
	Edges	36,60%	57,42%	79,06%	97,23%	99,76%	7.313.995	12,8 cm	13,8 cm
	Tuning	38,64%	59,43%	80,60%	97,81%	99,82%	5.197.802	12,1 cm	13,1 cm
C2 H3DG	Base	50,22%	76,91%	93,79%	99,43%	99,95%	8.294.564	7,52 cm	8,37 cm
	Edges	50,45%	75,58%	93,43%	99,30%	99,93%	6.724.556	7,62 cm	8,76 cm
	Tuning	52,09%	77,42%	94,98%	99,65%	99,97%	5.306.580	7,00 cm	7,51 cm

Table 5.7: The percentage of gaussians in the point cloud produced by 3DGS that are below a specific distance threshold in comparison to LiDAR is displayed in the table for each tested configuration. The mean and standard deviation global planar distance values, both in centimetres, are also displayed.

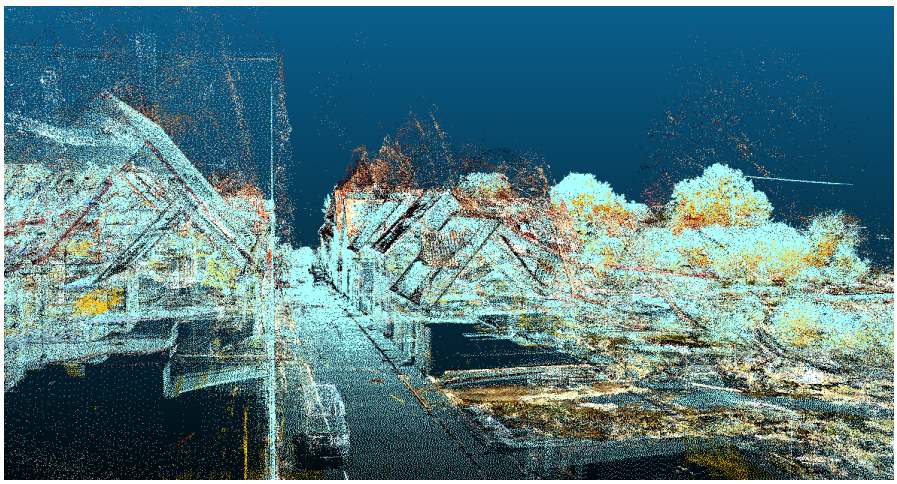

Figure 5.6: Point cloud representing block C2 of the H3DG dataset, generated via DN-Splatter in the basic configuration. In light blue, the LiDAR point cloud is shown, in RGB the generated point cloud. Clusters of outliers are evident on the rooftops.

This would cause issues during gaussian optimization and, consequently, while creating the final point cloud. In this case, a more homogeneous illumination turns out to be easier to reconstruct, as shown by the results obtained on the UseGeo dataset. Figure 5.7 shows the point clouds related to the distance calculated on block C2 obtained through basic configuration, both globally and in the plane, and the relative distribution of points shown on the histogram. It is clearly possible to see that the most distant points, colored red, are located at the roofs of the houses in the left Figure, which represents the global distance. In the right Figure, on the other hand, where the distance in the plane is shown, the Z component is not considered; thus, almost all the points are at a maximum distance of 50 cm from the LiDAR reference cloud.
However, in order to properly interpret the results obtained from both approaches, some considerations need to be made. All the point clouds generated by the models and used for distance calculation were compared with the corresponding LiDAR reference cloud, which is characterized by a GDS (Ground Sampling Distance) value of 2 cm, hence with a very high spatial resolution. However, as explained in

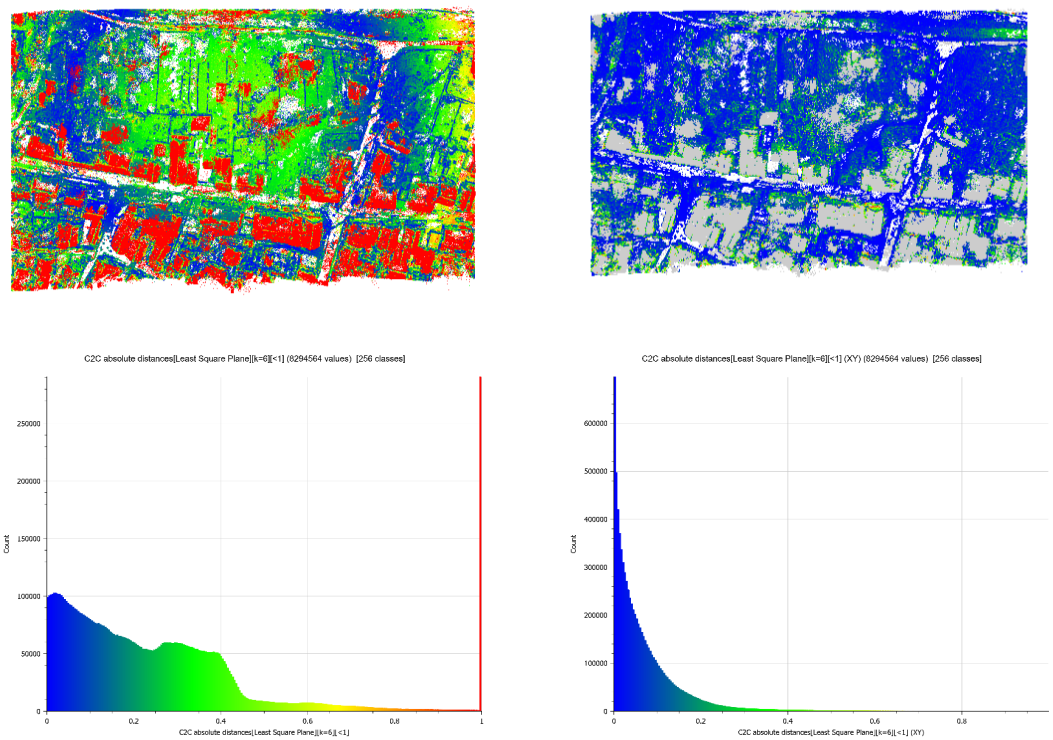

Figure 5.7: On the left, the point cloud representing the global distance with the relative distribution of points; on the right, the same representation but related to the planar distance.

Section 4.1, the images of both datasets considered underwent a rescaling operation that necessarily involved a change in the resolution of the ground images and consequently affecting the level of detail that the model can learn. Specifically,

- **UseGeo** images were reduced by a factor of 0.5, bringing the GDS to 4 cm;
- **H3DG** images were reduced by a factor of 0.28, bringing the GDS to 7cm.

As a result, errors below these thresholds will not be visible because they are below the single pixel size, thus below the detail capability that the model can generate. Errors below 10 cm can still be considered acceptable, as they correspond to a deviation of 1-2 pixels, depending on the dataset. Larger discrepancies, such as those bigger than 20 cm, may indicate a structural problem caused by the model's inaccurate reconstruction of the point cloud.
In general, it can be established that although the results obtained by the two algorithms are almost comparable from a photometric point of view, the DN-Splatter is more accurate from a geometric point of view. This algorithm manages to generate lighter point clouds, consequently reducing the computational impact given by gaussian optimization. The points, despite being fewer in number, turn out to be more accurately placed in geometry-critical areas such as edges. As can be seen from Figure 5.8, the DN-Splatter generates a "cleaner" point cloud in terms of geometry, with the points concentrated on edge areas and almost absent on flat surfaces, which are recreated during rendering using the few points available. In contrast, 3D Gaussian Splatting generates a much more chaotic and poorly structured cloud, with a large number of points in regions where they would not be needed. This therefore results in poor geometric accuracy of the algorithm. In addition, the higher geometric accuracy of the point cloud generated by DN-Splatter results in better photometric quality of the reconstruction in general. This increase is not very noticeable however, because although the gaussians are more precisely

Figure 5.8: On the left, the point cloud generated by the 3D Gaussian Splatting algorithm; on the right, the point cloud generated by the DN-Splatter algorithm, both in their basic configuration.

distributed, during the rendering phase they are not treated as discrete points, but as probability distributions of light in space. This therefore allows for good results from a photometric point of view, despite the presence of a non-geometrically accurate point cloud underneath.

Chapter 6

Conclusions

This research project was based on the exploration of the 3D Gaussian Splatting algorithm and its ability to reconstruct urban environments using aerial images captured by UAVs as input. The presented approach is part of an evolving technological scenery, where traditional active and passive reconstruction methods, such as LiDAR, Structure from Motion (SfM) and Multi-View Stereo (MVS), are progressively integrated with advanced deep learning techniques, including Neural Radiance Fields (NeRF) and their variants.
The algorithm was assessed photometrically, by comparing the reconstructed scene with input images, as well as geometrically, by calculating the distance between the generated point cloud and the ground-truth data, acquired via LiDAR sensor. On the photometric side, the conducted analyses showed a good reconstruction capability of the algorithm. The results did not appear as good in term of geometry. This is explained by the fact that the optimization process is based on an approximation of gaussians, whose primary objective is to make the reconstructed scene as similar as possible to ground-truth images. The quality of the underlying point cloud cannot be controlled, even though the final outputs may be of high quality.
In order to consider this factor as well, we examined the DN-Splatter algorithm, an extension of 3D Gaussian Splatting that exploits the supervision of depth and normals to improve the geometric consistency of the reconstruction. Experimental results showed that the integration of depth and normals maps helps improving the geometric fidelity of reconstructed scenes. The evaluated results were obtained following an intensive testing operation, which took place on both algorithms with different training configurations. Since handling a large number of high-resolution images involves a large demand in terms of computational resources, the training phase took place on a Cluster provided by the Politecnico di Torino. However, some useful optimizations were presented for those who would like to test the capabilities of these algorithms without specialized hardware resources.

Photometric and geometric tests showed that 3D Gaussian Splatting offers competitive results compared to other 3D reconstruction approaches, especially with regard to visual quality, although it still has room for improvement in point cloud management and geometric accuracy. The quality of the generated point clouds in fact proved to be unsatisfactory to obtain a geometrically accurate reconstruction. This could therefore create problems in all those contexts where having correct geometry is of critical importance, such as urban mapping or disaster management. Looking forward, the research work done in this thesis opens up several research directions. A key improvement could come from optimizing the distribution of gaussians. Currently, the algorithm distributes gaussians based on an iterative refinement strategy but does not explicitly take into account geometric consistency between surfaces or the relative importance of different areas in the scene. A possible improvement could come from using strategies to assign varying density to gaussians depending on the local complexity of the scene, reducing the number of gaussians in less significant regions and increasing resolution in more detailed areas. This would help improve computational efficiency without compromising the quality of the reconstruction. This aspect is very relevant, as the approach is difficult to scale up for large-scale real-world applications. Although 3D Gaussian Splatting is already more efficient than approaches such as Neural Radiance Fields (NeRF), processing large datasets remains an open problem. One possible development could involve the use of distributed architectures for gaussian training and optimization, so that the computational load can be spread over multiple nodes.
In summary, the work presented in this thesis is a first step toward the application of 3D Gaussian Splatting for UAV-based reconstruction of urban environments. However, the possible future developments are numerous and could lead this technology to become a key element in a wide range of scientific applications.

Bibliography

[1] A. Zingoni, M. Diani, G. Corsini, and A. Masini. «REAL-TIME 3D RECONSTRUCTION FROM IMAGES TAKEN FROM AN UAV». In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-3/W3 (2015), pp. 313–319. DOI: 10.5194/isprsarchives-XL-3-W3-313-2015. URL: https://isprs-archives.copernicus.org/articles/XL-3-W3/313/2015/ (cit. on p. 1).

[2] F. Nex and F. Remondino. «UAV for 3D mapping applications: a review». In: *Applied Geomatics* 6 (2014), pp. 1–15. DOI: 10.1007/s12518-013-0120-x. URL: https://doi.org/10.1007/s12518-013-0120-x (cit. on p. 1).

[3] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. «3D Gaussian Splatting for Real-Time Radiance Field Rendering». In: *ACM Transactions on Graphics* 42.4 (July 2023). URL: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/ (cit. on pp. 2, 13).

[4] Matias Turkulainen, Xuqian Ren, Iaroslav Melekhov, Otto Seiskari, Esa Rahtu, and Juho Kannala. «DN-Splatter: Depth and Normal Priors for Gaussian Splatting and Meshing». In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2025 (cit. on pp. 2, 22).

[5] Mingda Jia and Mingchuan Zhang. «An Overview of Methods and Applications of 3D Reconstruction». In: *International Journal of Computer Science and Information Technology* 3.1 (June 2024), pp. 16–23. DOI: 10.62051/ijcsit.v3n1.03. URL: https://wepub.org/index.php/IJCSIT/article/view/1981 (cit. on p. 4).

[6] Zhiliang Ma and Shilong Liu. «A review of 3D reconstruction techniques in civil engineering and their applications». In: *Advanced Engineering Informatics* 37 (2018), pp. 163–174. ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2018.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1474034617304275 (cit. on p. 4).

[7] Zhizhong Kang, Juntao Yang, Zhou Yang, and Sai Cheng. «A Review of Techniques for 3D Reconstruction of Indoor Environments». In: *ISPRS International Journal of Geo-Information* 9.5 (2020). ISSN: 2220-9964. DOI: 10.3390/ijgi9050330. URL: https://www.mdpi.com/2220-9964/9/5/330 (cit. on p. 4).

[8] Tam Do, Choi Jinwon, Viet Le, Philippe Gentet, Leehwan Hwang, and Seunghyun Lee. *HoloGaussian Digital Twin: Reconstructing 3D Scenes With Gaussian Splatting for Tabletop Hologram Visualization of Real Environment*. Oct. 2024. DOI: 10.20944/preprints202410.1253.v1 (cit. on p. 4).

[9] Ulrich Weiss and Peter Biber. «Plant detection and mapping for agricultural robots using a 3D LIDAR sensor». In: *Robotics and Autonomous Systems* 59.5 (2011). Special Issue ECMR 2009, pp. 265–273. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2011.02.011. URL: https://www.sciencedirect.com/science/article/pii/S0921889011000315 (cit. on p. 5).

[10] Kai-Wei Chiang, Guang-Je Tsai, Yu-Hua Li, and Naser El-Sheimy. «Development of LiDAR-Based UAV System for Environment Reconstruction». In: *IEEE Geoscience and Remote Sensing Letters* 14.10 (2017), pp. 1790–1794. DOI: 10.1109/LGRS.2017.2736013 (cit. on p. 5).

[11] Yu-Cheng Fan, Li-Juan Zheng, and Yi-Cheng Liu. «3D Environment Measurement and Reconstruction Based on LiDAR». In: *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2018, pp. 1–4. DOI: 10.1109/I2MTC.2018.8409759 (cit. on p. 5).

[12] Niu, Yuandong, Liu, Limin, Huang, Fuyu, Huang, Siyuan, and Chen, Shuangyou. «Overview of image-based 3D reconstruction technology». In: *J. Eur. Opt. Society-Rapid Publ.* 20.1 (2024), p. 18. DOI: 10.1051/jeos/2024018. URL: https://doi.org/10.1051/jeos/2024018 (cit. on p. 6).

[13] Linglong Zhou, Guoxin Wu, Yunbo Zuo, Xuanyu Chen, and Hongle Hu. «A Comprehensive Review of Vision-Based 3D Reconstruction Methods». In: *Sensors* 24.7 (2024). ISSN: 1424-8220. DOI: 10.3390/s24072314. URL: https://www.mdpi.com/1424-8220/24/7/2314 (cit. on p. 6).

[14] Mariusz Siudak and Przemyslaw Rokita. «A Survey of Passive 3D Reconstruction Methods on the Basis of More than One Image». In: *Machine Graphics and Vision* 23 (Jan. 2012), pp. 57–117. DOI: 10.22630/MGV.2014.23.3.5 (cit. on p. 6).

[15] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. «Image-Based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.5 (May 2021), pp. 1578–1604. ISSN: 1939-3539. DOI: 10.1109/tpami.2019.2954885. URL: http://dx.doi.org/10.1109/TPAMI.2019.2954885 (cit. on p. 6).

[16] Ives Rey Otero. «Anatomy of the SIFT method». Theses. École normale supérieure de Cachan - ENS Cachan, Sept. 2015. URL: https://theses.hal.science/tel-01226489 (cit. on p. 7).

[17] Johannes L. Schönberger and Jan-Michael Frahm. «Structure-from-Motion Revisited». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4104–4113. DOI: 10.1109/CVPR.2016.445 (cit. on p. 8).

[18] Penjani Nyimbili, Hande Demirel, Dursun Seker, and Turan Erden. «Structure from Motion (SfM) - Approaches and Applications». In: Sept. 2016 (cit. on p. 7).

[19] M.J. Westoby, J. Brasington, N.F. Glasser, M.J. Hambrey, and J.M. Reynolds. «'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications». In: *Geomorphology* 179 (2012), pp. 300–314. ISSN: 0169-555X. DOI: https://doi.org/10.1016/j.geomorph.2012.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0169555X12004217 (cit. on p. 7).

[20] Johannes Lutz Schönberger and Jan-Michael Frahm. «Structure-from-Motion Revisited». In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 8).

[21] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. «Pixelwise View Selection for Unstructured Multi-View Stereo». In: *European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 8).

[22] M. Goesele, B. Curless, and S.M. Seitz. «Multi-View Stereo Revisited». In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 2402–2409. DOI: 10.1109/CVPR.2006.199 (cit. on p. 9).

[23] Xiang Wang, Chen Wang, Bing Liu, Xiaoqing Zhou, Liang Zhang, Jin Zheng, and Xiao Bai. «Multi-view stereo in the Deep Learning Era: A comprehensive review». In: *Displays* 70 (2021), p. 102102. ISSN: 0141-9382. DOI: https://doi.org/10.1016/j.displa.2021.102102. URL: https://www.sciencedirect.com/science/article/pii/S0141938221001062 (cit. on p. 9).

[24] George Fahim, Khalid Amin, and Sameh Zarif. «Single-View 3D reconstruction: A Survey of deep learning methods». In: *Computers Graphics* 94 (2021), pp. 164–190. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2020.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0097849320301849 (cit. on p. 10).

[25] Anny Yuniarti and Nanik Suciati. «A Review of Deep Learning Techniques for 3D Reconstruction of 2D Images». In: *2019 12th International Conference on Information Communication Technology and System (ICTS)*. 2019, pp. 327–331. DOI: 10.1109/ICTS.2019.8850991 (cit. on p. 10).

[26] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. «NeRF: representing scenes as neural radiance fields for view synthesis». In: *Commun. ACM* 65.1 (Dec. 2021), pp. 99–106. ISSN: 0001-0782. DOI: 10.1145/3503250. URL: https://doi.org/10.1145/3503250 (cit. on p. 11).

[27] Guikun Chen and Wenguan Wang. *A Survey on 3D Gaussian Splatting*. 2025. arXiv: 2401.03890 [cs.CV]. URL: https://arxiv.org/abs/2401.03890 (cit. on p. 14).

[28] Martin Hornáček and Gregor Rozinaj. «Exploring 3D Gaussian Splatting: An Algorithmic Perspective». In: *2024 International Symposium ELMAR*. 2024, pp. 149–152. DOI: 10.1109/ELMAR62909.2024.10693978 (cit. on p. 14).

[29] Anurag Dalal, Daniel Hagen, Kjell G. Robbersmyr, and Kristian Muri Knausgård. «Gaussian Splatting: 3D Reconstruction and Novel View Synthesis: A Review». In: *IEEE Access* 12 (2024), pp. 96797–96820. DOI: 10.1109/ACCESS.2024.3408318 (cit. on p. 14).

[30] Umme Sara, Morium Akter, and Mohammad Shorif Uddin. «Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study». In: *Journal of Computer and Communications* 07 (Jan. 2019), pp. 8–18. DOI: 10.4236/jcc.2019.73002 (cit. on p. 19).

[31] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. «Image quality assessment: from error visibility to structural similarity». In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861 (cit. on p. 20).

[32] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. arXiv: 1801.03924 [cs.CV]. URL: https://arxiv.org/abs/1801.03924 (cit. on p. 20).

[33] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. «Omnidata: A Scalable Pipeline for Making Multi-Task Mid-Level Vision Datasets From 3D Scans». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 10786–10796 (cit. on p. 22).

[34] Gwangbin Bae and Andrew J. Davison. «Rethinking Inductive Biases for Surface Normal Estimation». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024 (cit. on p. 22).

[35] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. *ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth*. 2023. arXiv: 2302.12288 [cs.CV]. URL: https://arxiv.org/abs/2302.12288 (cit. on p. 23).

[36] F. Nex et al. «UseGeo - A UAV-based multi-sensor dataset for geospatial research». In: *ISPRS Open Journal of Photogrammetry and Remote Sensing* 13 (2024), p. 100070. ISSN: 2667-3932. DOI: https://doi.org/10.1016/j.ophoto.2024.100070. URL: https://www.sciencedirect.com/science/article/pii/S2667393224000140 (cit. on p. 25).

[37] *UAV-BASED Multi-sensor dataset for geospatial research*. URL: https://usegeo.fbk.eu/ (cit. on p. 25).

[38] Michael Kölle, Dominik Laupheimer, Stefan Schmohl, Norbert Haala, Franz Rottensteiner, Jan Dirk Wegner, and Hugo Ledoux. «The Hessigheim 3D (H3D) benchmark on semantic segmentation of high-resolution 3D point clouds and textured meshes from UAV LiDAR and Multi-View-Stereo». In: *ISPRS Open Journal of Photogrammetry and Remote Sensing* 1 (2021), p. 11. ISSN: 2667-3932. DOI: https://doi.org/10.1016/j.ophoto.2021.100001. URL: https://www.sciencedirect.com/science/article/pii/S2667393221000016 (cit. on p. 26).

[39] *Computational resources provided by HCP@POLITO, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://hpc.polito.it)*. URL: https://hpc.polito.it/ (cit. on p. 27).

[40] *Singularity Container Documentation*. URL: https://docs.sylabs.io/guides/3.5/user-guide/introduction.html (cit. on p. 29).

[41] *SLURM Documentation*. URL: https://slurm.schedmd.com/ (cit. on p. 30).

[42] Matthew Tancik et al. «Nerfstudio: A Modular Framework for Neural Radiance Field Development». In: *ACM SIGGRAPH 2023 Conference Proceedings*. SIGGRAPH '23. 2023 (cit. on p. 38).

[43] Matthew Tancik et al. «Nerfstudio: A Modular Framework for Neural Radiance Field Development». In: *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings*. SIGGRAPH '23. ACM, July 2023, pp. 1–12. DOI: 10.1145/3588432.3591516. URL: http://dx.doi.org/10.1145/3588432.3591516 (cit. on p. 38).

[44] Francesco Fugazzi. *3D Gaussian Splatting Converter*. URL: https://github.com/francescofugazzi/3dgsconverter (cit. on p. 46).