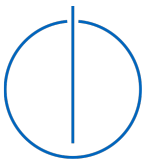
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Accurate Depth Extraction from 3DGS Models

Evelyn Regina Sidarta



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Accurate Depth Extraction from 3DGS Models

Genaue Tiefenextraktion aus 3DGS-Modellen

Author:	Evelyn Regina Sidarta
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	Prof. Dr. Rüdiger Westermann
	M.Sc. Han Liang
Submission Date:	April 1, 2025

I confirm that this bachelor's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, April 1, 2025 Evelyn Regina Sidarta

Acknowledgments

Before anything else, I would like to express my utmost gratitude to my advisor, Han Liang. Thank you for guiding me with patience and diligence throughout the entire process. It is only thanks to her indispensable help that I am able to complete this work and I am extremely grateful for this. I would also like to say thank you to my supervisor, Prof. Dr. Rüdiger Westermann, for granting me the opportunity to pursue this topic and affording me this invaluable experience.

Additionally, I would like to thank all the parties that have indirectly been involved in the creation of this work–thank you to Louis Bidou for creating the immaculate *Chocolate Bunny*, and also to creators of the depth extraction models involved in this study. Thank you for laying down the key foundations to my work and thank you for your contributions in the 3D Graphics community.

To you, the readers of my work: thank you for being a witness to the past few years of my life. I hope that in these pages you will find something useful to aid you in your own journeys.

With that said, I would also like to express my gratitude to my cherished friends and family. I know it has been a long time coming, but thank you for your unwavering support and belief in me. Through the highs and lows of life, you are a constant glow in my world and I'm forever grateful for your presence.

Finally, I would like to say thank you to my special person. Thank you for the warm hugs and the endless nights you spent accompanying me. You always make my days full of color and happiness. As I open the door to my next journey in life, I hope you will always be here to hold my hands.

Abstract

Gaussian Splatting is a rapidly-developing breakthrough in the field of visualization of 3D scene reconstruction. It has a wide range of application, spanning multiple fields like medical, aerospace, and even aviation and entertainment. With the discovery of the 3D Gaussian Splatting technique, it is now possible to do real-time rendering of scene reconstructions.

One of the most important aspects of 3DGS is depth estimation. Accurate depth extraction is needed in order to be able to capture and represent the scenes properly. This is, however, a difficult and ill-posed problem due to the nature of Gaussians and the infinite possibilities of scaling when extracting depth values.

This thesis aims to compile and dive into the depth extraction aspect of the 3D Gaussian Splatting method for scene reconstruction and examine some more current state-of-art Gaussian Splatting methods in comparison to one another. For a more complete comparison, we also take a look at current models for Monocular Depth Estimation tasks and compare their performances to the Gaussian Splatting methods. In order to do this, we selected a simple model to be the baseline for evaluation, extracted depth values from the model, and used it as training input for the models. A total of 3 MDE models are selected for the evaluation process: 1) Depth Anything v1, 2) Depth Antyhing v2, and 3) ZoeDepth, while a total of 4 GS models are used in this process: 1) 3DGS, 2) 2DGS, 3) RaDe-GS, and 4) GS2Mesh. Subsequently, the results of the training are evaluated statistically and visually and compared to each other. Here, we highlight models with exceptional performance like RaDe-GS and 2DGS, while also identifying weaknesses in the current available methods.

keywords: Gaussian Splatting, Monocular Depth Estimation, Depth Extraction

Contents

1 Introduction

1.1 Motivation

Visualization and reconstruction of three-dimensional (3D) scenes has always been a very important topic of interest across multiple disciplines- both scientific and social. It is used to aid in multiple different fields, such as in medical imaging, aerospace and aviation, and even in the entertainment field. In recent years, we see the rise of Neural Radiance Field (NeRF) methods that, unlike traditional rendering methods, are able to reconstruct accurate 3D representation of scenes from collections of two-dimensional (2D) images effectively with the help of deep learning and usage of neural networks [Mil+20]. The NeRF model also opens up the way for other practical implementations like novel view synthesis and geometry extraction and reconstruction from 2D scenes.

One major drawback of the employment of neural network in NeRF methods, however, lies in its costly resource consumption [Che+22]. Achieving high fidelity models require investing a huge amount time in training models and rendering. While able to provide impressive and highly-accurate results, this method still fails to achieve real-times display rates, thus hindering its application in scenarios where real-time rendering is important, particularly in virtual reality and computer vision sectors.

Consequently, 3D Gaussian Splatting (3DGS), a new method which revolutionized the field of real-time rendering, emerged. 3DGS is a method that employs 3D Gaussians [Ker+23] instead of using meshes to reconstruct a scene. Traditionally, in order to be able to construct a 3D scene, data from 2D images need to first be converted into surfaces or line primitives [CRF24] [FSG16] [Tan+19]. The introduction of the splatting technique allows this step to entirely be skipped and replaced by the construction of 3D Gaussians instead. These 3D Gaussians allow for the scene to be represented through continuous functions that retain density and radiance at a point, which, in turn, allows for faster and much less expensive–but still accurate–reconstruction of the 3D scene and the ability to support real-time rendering of the 3D scenes.

Despite its massive success in the field of novel view synthesis, there is a yet to be fully-explored potential of the employment of this method specifically for accurate depth extraction. In order to be able to capture and represent 3D scenes accurately, accurate depth estimation of objects inside the scene is needed. However, due to the unstructured and sparse nature of the 3D Gaussians, it is difficult to accurately get

depth data from the fully-reconstructed scenes, especially in scenes laden with details or scenes with reflective surfaces [Xu+24]. Lately, there has been some advancements in this field of research with the release of works specifically exploring depth or accurate geometrical extraction [Zha+24] [COL24] [Hua+24] [WBK24] in conjunction with the usage of Gaussian Splatting. This is further supplemented with recent works in monocular depth estimation [Bha+23] [Yan+24b] [Yan+24c], which aim to be able to identify depth accurately using only a single image source.

1.2 Contribution

This paper aims to review the mechanisms of current current available state-of-art methods for depth estimation and compare them to each other and also the currently available monocular depth estimation models. Firstly, the paper will delve deeper into the original implementation of the 3DGS method [Ker+23], then compare this to the implementations of current state-of-art methods like two-dimensional Gaussian Splatting (2DGS) [Hua+24], RaDe-GS [Zha+24], GS2Mesh [WBK24], and also other monocular depth estimation methods like DepthAnything [Yan+24b] [Yan+24c] and ZoeDepth [Bha+23]. By comparing these models, we aim to be able to provide an overview of the current advancements in Gaussian Splatting, specifically pertaining to the issue of depth extraction.

1.3 Outline

In this first chapter we have explained the underlying motivation behind this work and also the main goal of the creation of this paper. The second chapter provides a concise overview over main concepts relevant to this publication, specifically about the original 3DGS method: its structure, history, and underlying mechanisms. Here we also explain concepts that lay the groundwork for the current state-of-art methods in Gaussian Splatting and monocular depth extraction.

The third and fourth chapters provide an overview of current state-of-art methods in depth extraction from 3DGS models and, along with it, an overview of current monocular depth extraction methods that are used as comparison. Here we also compare and contrast differences of each methods to the original 3DGS method.

In the fifth chapter, we explain details of the methodology of our research: how the data is prepared and processed and then used as input to our models and how depth data is then extracted from the output of the models. We also outline the entire post-processing of the data, as well as the visual and statistical evaluation process.

In the sixth chapter, we provide both visual and statistical comparisons of depth data extracted from 3DGS, 2DGS, RaDe-GS, DepthAnything v1, DepthAnything v2, ZoeDepth, and GS2Mesh and compare them to the ground truth. Afterwards, we discuss the quality of these methods both statistically and visually in comparison to the original method and to each other.

In the seventh chapter, we draw a conclusion based on the results of our discussion and also discuss potential improvements for further research.

2 Background

2.1 Linear Transformation Techniques

Linear transformation [Axl23], or often also called a linear mapping or linear function, is a transformation of vectors from one vector space to the other ($T : V \to W$). Linear transformations between vectors can be represented by matrices.

2.1.1 Translation

Translation [Pau81] is a type of geometric transformation where a point in a coordinate system is moved a certain distance in a certain direction. It can be interpreted as a simple addition operation. Given a translation vector $\vec{v}$ and an initial point p, the translation function $T_{\vec{v}}$ is as follows:

$$T_{\vec{v}}(p) = p + \vec{v}$$

2.1.2 Rotation

Rotation is another type of geometric transformation where a point is rotated against a certain reference. A standard rotation in a two-dimensional (2D) space can be described as follows:

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix},$$

with θ representing the angle of the rotation.

In the three-dimensional (3D) plane, rotations about the $x-, y-$, and $z-$planes can be described as follows:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.2 Novel View Synthesis

Novel view synthesis (NVS) is a branch of research that focuses on the reconstruction of 3D scenes from a limited number of input images that normally represent a limited amount of differing viewpoints of the scene. The main task being tackled here is to be able to construct the scene from all other angles that are previously not provided by the input images [Wil+20].

2.2.1 Motion Parallax and Depth Extraction

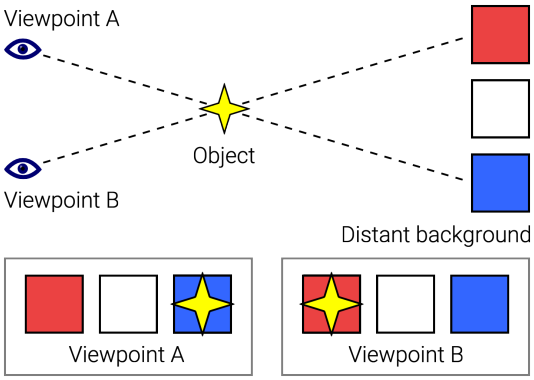


Figure 2.1: Parallax effect when viewing an object from two different viewpoints

Throughout its early development stages, one key idea mainly used to handle NVS problems is to take advantage of motion parallax: when an observer (usually denoted as the camera) moves around a scene, objects will naturally "move" from the current point of view depending on their distances from this observer [SS01]. Parallax is defined as the visual phenomenon where displacement or difference in the position of an object seem to change when viewed from different points of view [Cam16]. Objects that are located closer to the observer will display a larger parallax effect when compared to objects that are farther away. Distance of the object can, due to this, be measured by measuring angles and observing the parallax. This distance can then be interpreted as depth value from the observer's point of view.

2.2.2 Distance Measurement

Distance measurement using parallax distance is based on the principle of triangulation. Triangulation is a technique developed to determine the location of a point in 3D space by forming triangles to this point from other points that are already known.

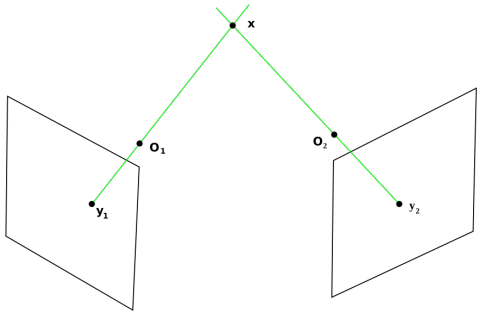


Figure 2.2: 3D point projection onto two different 2D images

Given two different 2D images of a scene with known camera parameters, if we want to know where a certain point of an object is located on a 3D plane, we can first extract the 2D coordinates (image points) of the same point from the two different images. Since we are also given each camera's parameters, we are also able to determine the focal point of both cameras in 3D space. Consequently, we are able to draw two different lines from each image points to their respective focal points in 3D space (or from the center or location of the camera in 3D space towards the image point). These two lines will, eventually and inevitably, intersect at a point in 3D space: this point can then be regarded as the location of our point in the 3D space. Next, to determine the exact distance, we first obtain the angle created by this intersection and use this along with the disparity between the two images (distance between the two camera centers) to calculate the depth, which is the height of the triangle (see Figure 2.3) [Bri08]:

$$d = \frac{0.5 \cdot b}{tan(0.5 \cdot \alpha)},$$

with d as the distance value we are trying to find, b as the distance between two camera centerpoints, and α as the angle created by the intersection.

2.2.3 Structure from Motion

Following this main idea, various Structure from Motion (SfM) techniques are then developed to be able to specifically calculate points in a 3D plane purely based on

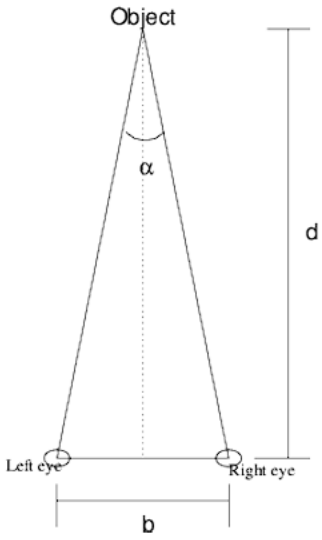


Figure 2.3: distance measurement principle from triangulation technique

2D transformations of the projected images [Ull79]. SfM techniques generally follow the same basic steps. Firstly, we try to find points from different points of view that correspondent to each other. This step is called feature extraction and feature matching. Typically, this is done by identifying some distinct "key points" in each image. These points are normally corners of an object. A minimum of 8 points are required, hence why this method is known as the Eight-point algorithm [Lon87]. After these points are identified, they are then matched to the other key points from all other images and then compared (see Figure 2.4).

Next, epipolar geometry is used to relate the points previously matched with the cameras. Given a single point in 3D-space $\mathbf{X}$ which is captured as $\mathbf{x}$ and $\mathbf{x}'$ in two different images, we are able to relate these two points with their respective camera centers $\mathbf{C}$ and $\mathbf{C}'$. Since the three points (the original point and the two projected points) are coplanar, which means they lie in the same plane [Swo83], we can create a relationship between them as follows:

$$\vec{\mathbf{Cx}} \cdot (\vec{\mathbf{CC}'} \times \vec{\mathbf{C}'\mathbf{x}'}) = 0$$

As for the rest, it is only a matter of applying triangulation strategies as aforementioned to get the desired points in space.

For the final step, bundle adjustment [Tri+99] is done to refine the 3D points gained from the previous calculations to minimize the cost function and filter out unneeded 3D points. This is done by minimizing the reprojection error, which is essentially a

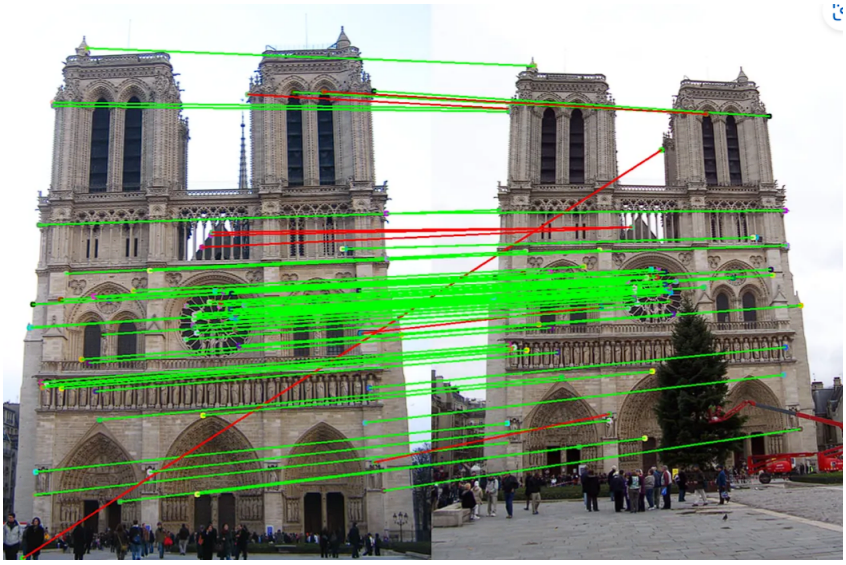


Figure 2.4: example of feature matching step

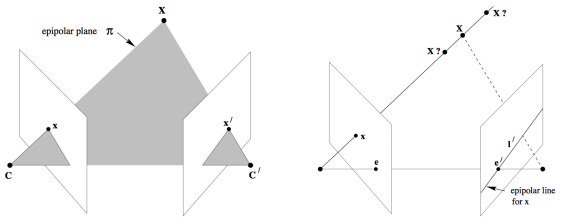


Figure 2.5: epipolar geometry

geometric error that pertains to the distance between an estimated point and the actual projected point [HZ04].

2.3 Monocular Depth Estimation

Monocular Depth Estimation (MDE) is a classical task in the computer vision field, commonly regarded as a problem under simultaneous localization and mapping (SLAM) [Hu+12], with a very broad usage in critical fields such as navigation [Zhu+15] and object detection [Cha+17]. Given a single monocular RGB image, this problem focuses on estimating the depth values in the image.

Traditional MDE methods employ methods such as SfM and stereo calibration. More recent advancements employ the usage of neural networks and learning techniques

[AW19] [Oqu+24] [Lai+16] [Zha+20] in order to provide a high quality output. Some notable works in this field include: 1) MiDAS [Ran+20], which introduces tools to train models on different datasets, 2) ZoeDepth [Bha+23], which focuses on combining Relative Depth Estimation techniques with Metric Depth Estimation techniques, and 3) Depth Anything [Yan+24b] [Yan+24c], which focuses on developing a more generalist model to predict images under any scenarios.

2.4 Radiance Field Methods

Following the advancements in SfM methods and deep learning, the neural radiance field method is discovered. Neural radiance field (NeRF) [Mil+20] is a deep learning technique aimed to address the task of three-dimensional (3D) reconstruction based solely on two-dimensional (2D) photographs or images of a scene. The basic NeRF algorithm outputs a scene reconstruction as a radiance field created by parameters assigned through a learning process by a deep neural network.

2.4.1 NeRF Method

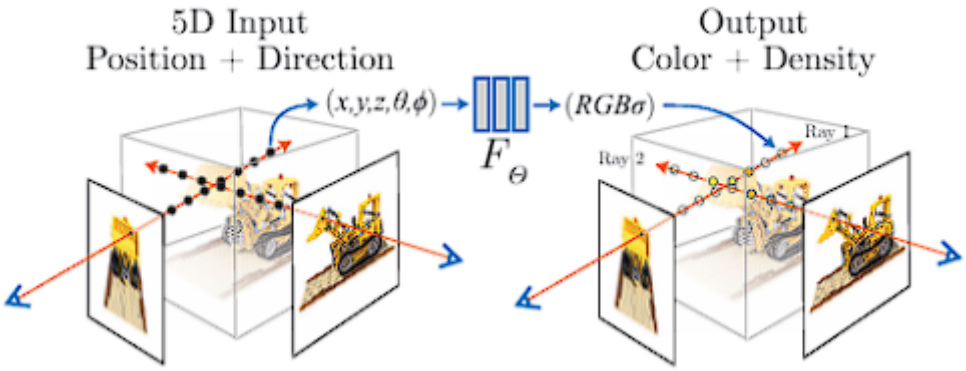


Figure 2.6: illustration of the input-output scheme of NeRF [Mil+20]

As input, the algorithm accepts a collection of images that represent a 3D scene from multiple viewing angles and camera poses. Each input image contains data on the image's camera's viewing direction $d = (\theta, \phi)$, as well as the image data itself that contains information on spatial location (x, y, z) in the coordinate system. Since multiple input images are received, this can then be condensed into a continuous 5D function and fed into a multilayer perceptron (MLP) network. An MLP network itself is basically a feedforward neural network that obtains a set of inputs which are then fed

forward to a set of layers that are tasked to adjust a set of weights to obtain the most optimal output [Hof91]. MLPs consist of neurons that are entirely connected to each other with nonlinear activation functions and are mainly used for input data that are typically more complex and not linearly separable [Cyb89]. This MLP network can be represented as such (see Figure 2.6):

$$F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma),$$

with Θ as the weights trained by the neural network, $\mathbf{x}$ as our input coordinate, $\mathbf{c}$ as the color output, and σ as the density output.

2.4.2 Improvements to Original NeRF Method

Since early applications of NeRF method still suffer from large optimization drawbacks, other related works aiming to improve this pipeline have since emerged. There have been some methods that optimize camera pose predictions and NeRF's volumetric function [Lin+21], improve the sharpness level of details produced by the method [Bar+22], and improved assignment of initial weights to the MLP [Tan+21]. There have also been other researches that focused on improving NeRF applications for dynamic scene [Mar+21] and experimentally added more outputs to enable custom lighting [Sri+20]. Most notably, more recent methods before the introduction of Gaussian Splatting aimed to address NeRF's costly training times and pursue real-time rendering [Yu+21b] [Hed+21] [Mül+22a].

2.4.3 Advantages of the NeRF Method

One of the main advantages of NeRF method is that it is multiview consistent, which means that we are able to see representations of the reconstructed scene from different points of view and it should still be consistent and coherent. NeRF achieves this by first restricting the volume density (opacity) function to depend only on the 3D location of the point, while allowing the RGB color $\mathbf{c}$ to be predicted based on the 3D location and the viewing direction of the camera. This ensures that the density will not change based on the viewpoint of the camera: what is dense at one point should also stay dense when looked at from other points and vice versa.

Another main advantage is that NeRF method is able to produce reconstructions of superior quality when compared to its predecessor methods. Previously, instead of outputting radiance field, the usage of deep neural networks that output discretized voxel representations are more prominent. However, when compared to these methods, NeRF was able to not only provide better reconstructions, but also maintain competitive rendering time.

2.4.4 Limitations of the NeRF Method

Despite its massive success in creating a scene reconstruction with superior quality, NeRF still suffers mainly from its long training times. While it is true that NeRF training times remain competitive when compared to other previous methods that also use deep convolutional networks, NeRF is highly restricted by hardware capabilities and compute power, since it demands a large amount of resources in exchange for quality. All advancements in NeRF methods still is restricted by the fact that it demands the usage of a full-scale neural network. Thus, it is still unable to be used for more practical applications in fields that require more real-time rendering capabilities.

2.5 Gaussian Splatting

Gaussian Splatting is a volume rendering algorithm first introduced in 1991 by Lee Alan Westover [Wes91]. The main idea of this method is "splatting", which is the projection of volume masses directly onto the image plane during the rendering process instead of properly describing the volume as surface or line primitives and then using geometric meshes to draw the scene. The result of this algorithm delivers a point cloud – a set of points (or, here, Gaussians) that, when rendered, represents the scene in question.

2.6 3D Gaussian Splatting

3D Gaussian Splatting [Ker+23] a major breakthrough in real-time rendering of reconstructed 3D radiance fields. It addresses NeRF's main weakness, which is its inability to render scenes in real-time. The main idea of this technique is the direct usage of "splats" of 3D Gaussians to represent scenes instead of converting data acquired from learning process into primitives first, therefore skipping an entire step in the rendering process. These 3D Gaussians retain important volumetric properties, such as density, color, and transmittance, which opens the gate for real-time rendering of high-quality multiview reconstructions of 3D scenes. Furthermore, the original 3DGS method offers further optimization on these properties in order to enhance rendered quality of the scene and also presents a faster rendering technique, runnable on GPU, that considers visibility during the rendering process and is therefore more resource-saving.

2.6.1 3DGS Pipeline

The 3DGS method follows several important steps from start to finish. First, it accepts as input a set of 2D images, along with the corresponding camera poses and locations (and a point cloud by-product) that have been calibrated by SfM techniques (specifically,

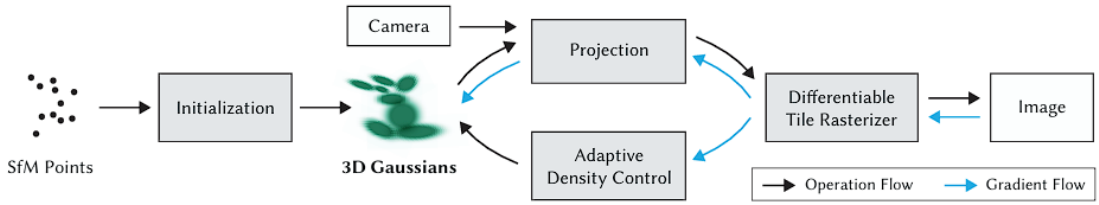


Figure 2.7: 3DGS pipeline

with COLMAP) [SF16] [Sch+16]. Next, the algorithm produces a set of 3D Gaussians to represent the scene. These Gaussians contain positional information, as well as covariance matrix and opacity. Afterwards, the algorithm determines the color of each pixels on the view-dependent scene using spherical harmonics [Yu+21a] [Mül+22b]. Then, a series of optimization steps are taken in order to adjust the distribution of the 3D Gaussians to ensure finer details are kept while also minimizing resource-consumption in blander areas. Lastly, a tile-based rasterizer is implemented to enable efficient and visibility-aware calculation of color. This allows fast rendering duration while also removing unnecessary resource consumption from the rendering algorithm.

2.6.2 3D Gaussians

In the traditional 3DGS method, Gaussian objects are defined as follows:

$$G(x) = e^{-\frac{1}{2}(x-x_p)^T \Sigma^{-1}(x-x_p)},$$

with $\sum$ as a 3D covariance matrix and x_p as the center of the Gaussian [Zwi+01].

During the rendering phase, 3D Gaussians are projected onto the image space. Given viewing transformation W and Jacobian of the affine projection onto image plane J, the covariance matrix Σ' in camera coordinates is as follows:

$$\Sigma' = JW\Sigma W^T J^T \text{ [Zwi+01].}$$

Furthermore, removing the last row and the last column from the resulting covariance matrix leaves us with the projected 2D Gaussians in the image plane.

2.6.3 Alpha Blending

For the color calculation, 3DGS method employs traditional alpha blending technique. Alpha blending is a color determination technique that considers transparency levels of different objects and also the background [PD84]. In 3DGS, alpha blending is done from front to back as follows:

$$C = \sum_{i \in N} \alpha_i c_i \prod_{j=1}^{i-1} (1 - \alpha_j),$$

with α_i as product of the i-th Gaussian with its opacity and c_i as the color component.

2.6.4 Spherical Harmonics

Spherical Harmonics (SH) are functions pertaining to the surface of a sphere that are often used to model different types of surfaces [RH01] [SKS23]. The traditional 3DGS method closely follows SH representation of color used by NeRF methods [Yu+21b]. Rather than outputting RGB values, a function that outputs spherical harmonics coefficients is created:

$$f(x) = (\mathbf{k}, \sigma), \text{ with } \mathbf{k} = (k_l^m)_{l:0 \leq l \leq l_{max}}^{m:-l \leq m \leq l}.$$

Here, $k_l^m \in \mathbb{R}^3$ represents a set of 3 coefficients corresponding to each components of RGB [Yu+21b]. With this representation, the color c at any point $\mathbf{x}$ can be obtained with the following function:

$$c(\mathbf{d}; \mathbf{k}) = S(\sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} k_l^m Y_l^m(\mathbf{d})),$$

with $\mathbf{d}$ as the desired viewing angle, $S : x \to (1 + e^{-x}))^{-1}$ as the sigmoid function for color normalization, and $Y_l^m : \mathbb{S}^2 \to \mathbb{R}$ as the SH function in question.

2.6.5 Optimization Pipeline

The optimization step of 3DGS aims to optimize the density of the 3D Gaussians so that it would build an accurate reconstruction of the scene. In addition to this, SH coefficients from the previous step are also optimized to accurately capture the scene. This is done by doing several successive rendering iterations and comparing the rendering result to the training images (photometric loss).

2.6.6 Rasterizer Improvements for Real-Time Rendering

The rasterization process of 3DGS is done in several steps [Ker+23]. First, tile-based rasterization is performed. This step splits the screen into a 16×16 set of tiles and then culls 3D Gaussians that do not significantly contribute to the scene representation to reduce computational overhead. Only Gaussians with a 99% confidence interval or above are kept and Gaussians located at unreliable positions (e.g. too close or too far away) are eliminated. Next, these remaining Gaussians are sorted and alpha

blending is done in a front-to-back style until full opacity is reached. Afterwards, in order to avoid excessive memory usage, the sorted list of Gaussians is traversed once again back-to-front for the backward pass step done in order to optimize the gradient computation.

2.6.7 Improvements to Traditional 3DGS Method

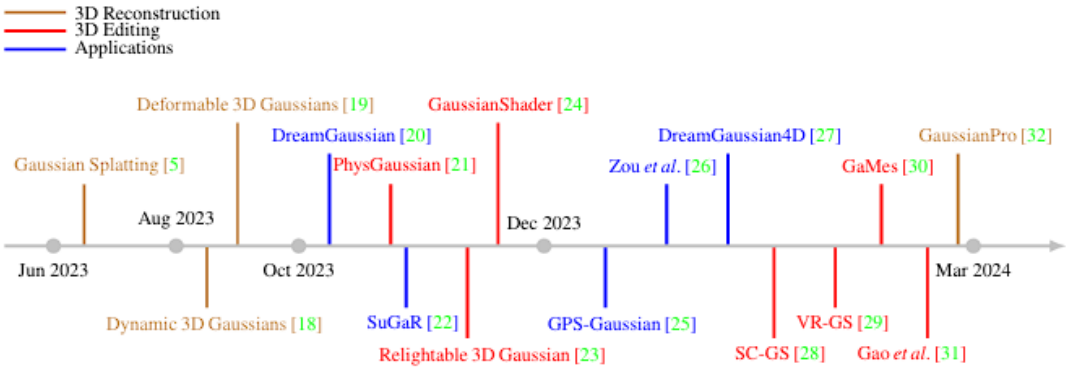


Figure 2.8: A timeline depicting advancements to the 3DGS method until March 2024 [Wu+24]

Since the release of the original 3DGS method, various further iterations and improvements have been made to the method. Most notably, improvements have been made to the rendering quality of 3DGS [Yu+23] [Zwi+01] [Che+24], explored geometric representation aspects of 3DGS [Hua+24] [Zha+24] [WBK24], or even specifically performing SLAM tasks [Yan+24a] [Kee+24] [Hu+24a].

2.6.8 Limitations of the Traditional 3DGS Method

Since it is a newly founded technique, traditional 3DGS tend to suffer from noisy reconstructions and optimization problems. In places where the scene is not captured well, elongated artifacts may be formed. This is made even worse by the fact that antialiasing techniques have not been implemented in the original 3DGS technique. Another limitation of 3DGS is that regularization techniques are not applied during optimization. This also, in turn, contributes to the noisy reconstruction of the scenes. Furthermore, training consumes a lot of resources since the technique is still rough around the edges and sometimes, this even results in higher memory consumption when compared to NeRF models [Ker+23].

3 Monocular Depth Estimation Models

In this chapter, we briefly go through the monocular depth estimation models used in this research.

3.1 ZoeDepth

ZoeDepth [Bha+23] is a pioneering work on Monocular Depth Estimation (MDE). Previously, MDE models focus on either providing Relative Depth Estimation (prediction of depth within a scene without providing actual scale) or Metric Depth Estimation (prediction of depth in absolute units, e.g. meters). ZoeDepth combines both approaches in order to achieve a better generalization ability with a high accuracy.

There are two main stages in the proposed framework for ZoeDepth: 1) pre-training an encoder-decoder model to estimate relative depth in order to achieve high generalization, and 2) fine-tuning on datasets with metric depth annotation in order to be able to adjust relative depth predictions to metric scale accurately. This is done using a similar training strategy to MiDaS [Ran+20] and attaching a *metric bins module* that outputs metric depth to the decoder of the model. Through this framework, ZoeDepth is able to achieve zero-shot transfer capability with good generalization and also provide state-of-art depth estimation capability when fine-tuned to specific datasets.

3.2 Depth Anything v1

Depth Anything [Yan+24b] is a current state-of-art generalist MDE model that is able to create an accurate depth map estimate of any images under any situations. In order to do this, Depth Anything v1 mainly takes advantage of two simple strategies: 1) using data augmentation tools to develop more complex scenarios as training material for the model and 2) the model is supervised and directed to inherit rich semantic priors from the usage of pre-trained encoders.

Depth Anything v1 is mainly based on MiDaS [Ran+20], a pioneering MDE foundation model which was able to demonstrate zero-shot ability, which means that it is able to achieve an accurate prediction on previously unseen images. The early MiDaS model is limited, in that it does not have enough data coverage, which means that it

will perform poorly on certain circumstances. The first iteration of Depth Anything aims to address this by scaling up the dataset, mainly leveraging large-scale unlabeled monocular data, which amounts to approximately 62 million unlabeled images. This is done by first creating an engine to automatically label these raw and unlabeled image datasets with the correct depth annotation. For this annotation tool, an initial 1.5 million already-labeled images were first collected and used to train an initial MDE model. This is then used to automatically annotate the unlabeled images in a self-learning manner.

After preliminary evaluation, however, it was found that simply being able to use a larger-scale unlabeled data does not necessarily improve the performance of initial models. Therefore, it was proposed that the models are trained with a more difficult optimization target during the learning process. Furthermore, the use of the second strategy is proposed in order to guide the model using previous pre-existing encoders trained on large datasets. This allows the model to better identify visual information on the objects in every image and how they relate to each other, which leads to overall improvement in depth estimation.

Following these two core strategies, the first iteration of Depth Anything is able to produce a foundational MDE model that, when evaluated across six representative unseen datasets: KITTI [Gei+13], NYUv2 [Sil+12], Sintel [But+12], DDAD [Gui+20], ETH3D [Sch+17], and DIODE [Vas+19], is able to outperform MiDaS v3.1 [BWM23], a more recent MiDaS iteration that uses more labeled datasets when compared to Depth Anything. Additionally, this model is also able to produce better results when compared to ZoeDepth when fine-tuned with metric depth.

3.3 Depth Anything v2

Depth Anything v2 [Yan+24c] is the second iteration of the previous Depth Anything model. Compared to the previous model, it is able to produce more robust and detailed depth map predictions by using three main strategies: 1) using synthetic images instead of labeled real images, 2) improving the capacity of the teacher model, and 3) generating a large amount of pseudo-labeled real images to train student models, enhancing the generality of the model in more diverse scenarios.

Through evaluating several zero-shot MDE models like Depth Anything v1 [Yan+24b], Metric3D v1 [Yin+23], Metric3D v2 [Hu+24b], and ZeroDepth [Gui+23], it was discovered that the usage of real labeled data has two main disadvantages: 1) labeled data often contain noise and there are often inaccuracies in the labels itself, and 2) depth map details are often overlooked. On the other side, synthetic images have very precisely labeled details and it is possible to obtain actual depth of challenging scenarios such as

transparent objects or reflective surfaces easily when employing synthetic images as training material. Of course, these advantages are not without limitations; synthetic images differ in style and color distribution when compared to real life images, and can be regarded as being "too clean" and "too ordered". Furthermore, synthetic images have restricted coverage, since synthetic images are taken from scenes with clear separation (e.g. "living room" or "streets"), while in real life images the possibilities are boundless (e.g. it is totally possible to take a picture containing both the living room and the streets at the same time).

Due to these problems, a pilot study was conducted to assess the performance on pre-trained encoders when fed only synthetic images. From this study, only DINOv2-G [Oqu+24] was found to be satisfactory and therefore able to be used as the base for the teacher model used by Depth Anything v2, since other models suffer from generalization issues.

By employing the aforementioned strategies, Depth Anything v2 is able to produce models that, when compared to its predecessor [Yan+24b], are able to provide more detailed depth predictions, especially when dealing with transparent objects and object with finer details. This is also the case when comparing to ZoeDepth [Bha+23]; Depth Anything v2 is able to spot miniature details (e.g. densely packed objects or thin surfaces), while ZoeDepth often lumps these minuscule details together since it is unable to recognize it.

4 Derivative Works of 3DGS

In this chapter, we briefly go through each Gaussian Splatting model used in this research.

4.1 2D Gaussian Splatting

Two-dimensional Gaussian Splatting (2DGS) [Hua+24] is a derivative work based on the original 3DGS technique. One of the main goals of 2DGS is to be able to create a multi-view-accurate surface representation of a scene, which is a main weakness of its predecessor [Ker+23]. 3DGS, while able to achieve high-quality results in real time, is unable to represent surfaces of objects in the scene accurately due to its lack of explicit surface representation. 3D Gaussians are optimized independently without looking at the overall structure of the scene, which leads to it not being coherent with each other. This means that when we try to look at the overall result from different perspective, the overall geometry of the scene may appear to change. The main difference of 2DGS is that instead of using 3D Gaussians, it focuses on using 2D Gaussians, which allows for more accurate surface reconstruction and view-consistent geometry reconstruction using intrinsic modelling of surfaces. Processed values are then regularized through depth distortion and normal consistency. Through these means, 2DGS is able to create more geometrically-accurate surfaces while maintaining fast training and rendering speed.

4.1.1 Modeling and Splatting of Gaussians

As aforementioned, 2DGS employs the usage of 2D Gaussians instead of 3D Gaussians in an effort to provide better alignment with thin surfaces. 2D Gaussians are planar elliptical disc-like two-dimensional structures which possess a center point p_k, two tangential vectors t_u and t_v, and two scaling factors s_u and s_v, combined into a scaling vector $S = (s_u, s_v)$ to control the variance. The variance of a Gaussian describes how elliptical or circular a Gaussian is in each axes, while the tangential vectors describe the orientation and can be arranged into a 3×3 rotation matrix $\mathbf{R} = [t_u, t_v, t_w]$ with $t_w = t_u \times t_v$. The scaling factor can also be converted into a 3×3 diagonal matrix $\mathbf{S}$ with zero as its last entry.

In a local tangent plane, a 2D Gaussian is described as follows:

$$P(u,v) = p_k + s_u t_u u + s_v t_v v = \mathbf{H}(u,v,1,1)^T$$
$$\text{where } \mathbf{H} = \begin{bmatrix} s_u t_u & s_v t_v & 0 & p_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}S & p_k \\ 0 & 1 \end{bmatrix}$$

In order to evaluate the value of a certain 2D Gaussian at a certain point $\mathbf{u} = (u,v)$ we can use the following formula:

$$\mathcal{G}(u) = exp(-\tfrac{u^2+v^2}{2})$$

In 2DGS, center point p_k, scaling $S = (s_u, s_v)$, as well as rotation (t_u, t_v) are learnable parameters. Additional parameters opacity α and view-dependent appearance c are also attached to each Gaussians.

For the splatting process, a formula based on homogeneous coordinates [Zwi+04] is used:

$$\mathbf{x} = (xz, yz, z, z)^T = \mathbf{W}P(u,v) = \mathbf{WH}(u,v,1,1)^T,$$

where $\mathbf{x}$ represents 2D coordinates (x,y) in screen space point and z represents depth at ray-splat intersection.

For the rasterization process, explicit ray-splat intersection [Sig+06] method is used in order to avoid ill-conditioned transformations induced by the usage of implicit methods [Zwi+04]. Given a coordinate point $\mathbf{x} = (x,y)$, a ray being shot into the scene is parametrized as the intersection of the x-plane and the y-plane, where the x-plane can be represented as $\mathbf{h}_x = (-1,0,0,x)^T$ and the y-plane can similarly be represented as $\mathbf{h}_y = (0,-1,0,y)^T$. Afterwards, a transformation matrix $\mathbf{M} = (\mathbf{WH})^{-1} = (\mathbf{WH})^T$ is used to transform both planes into the local coordinate system of the 2D Gaussian primitives, yielding:

$$h_u = (\mathbf{WH})^T h_x \text{ and } h_v = (\mathbf{WH})^T h_y, \text{ with } h_u \cdot (u,v,1,1)^T = h_v \cdot (u,v,1,1)^T = 0$$

For the intersection point $\mathbf{u}(x)$, the solution is:

$$u(x) = \tfrac{h_u^2 h_v^4 - h_u^4 h_v^2}{h_u^1 h_v^2 - h_u^2 h_v^1} \text{ and } v(x) = \tfrac{h_u^4 h_v^1 - h_u^1 h_v^4}{h_u^1 h_v^2 - h_u^2 h_v^1},$$
$$\text{with } h_u^i and h_v^i \text{ as } i\text{-th parameter of the 4D plane and } h_u^3 and h_v^3 = 0$$

Due to the usage of 2D Gaussians, when observed from specific standpoints the shape will degenerate into a line and could then be missed in the rasterization process. To overcome this, object-space low-pass filter is used [Bot+05]:

$\hat{\mathcal{G}}(x) = max\{\mathcal{G}(\mathbf{u}(x)), \mathcal{G}(\frac{x-c}{\sigma})\}$, with c as the projection of center point p_k.

This low-pass filter essentially "fixes" the minimum value of the Gaussians so that such small-valued Gaussians are not be missed during the rendering process. This value is, in this case, bounded by a filter with center c and radius σ with a fixed value of $\sigma = \frac{\sqrt{2}}{2}$.

Next, for the actual rasterization, a process similar to 3DGS is used. First, bounding boxes are computed for each Gaussians and then these Gaussians are sorted and organized based on the depth of its center point into their respective bounding boxes. As in 3DGS, alpha front-to-back blending is then used during rendering:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} (\mathbf{c}_i \alpha_i \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$$

4.1.2 Training of 2DGS Models

During the training process, two regularization terms: 1) depth distortion and 2) normal consistency are introduced. This is done in an effort to prevent too much noise during reconstruction. Depth distortion is first introduced by the predecessor of 2DGS, Mip-NeRF360 [Bar+22] to prevent "background collapse" phenomenon, in which distant surfaces are incorrectly modeled as semi-transparent. In 2DGS, depth distortion is done to minimize distance between ray-splat intersections and encourage concentration of weight distribution along the rays:

$$\mathcal{L}_d = \sum_{i,j} w_i w_j |z_i - z_j|,$$

with $\omega_i = \alpha_i \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$ as blending weight of the i-th intersection and z_i as depth of its intersection points.

The second term, normal consistency, is used to guarantee that the 2D Gaussians are locally aligned with surfaces having an accumulated opacity of 0.5 at median intersection point p_s, such that the 2D Gaussians are able to be an approximate to the actual object's surface:

$$\mathcal{L}_n = \sum_i \omega_i (1 - n_i^T N),$$

with i as index indicator of the number of intersecting splats along the ray, ω as blending weight of the intersection points, n_i as normal of the i-th splat, oriented towards the camera position, and N as normal estimated by the gradient of the depth map:

$$N(x, y) = \frac{\nabla_x p_s \times \nabla_y p_s}{|\nabla_x p_s \times \nabla_y p_s|}$$

4.1.3 Limitations of 2DGS

Although 2DGS is able to achieve accurate geometric modeling of 3D scenes, it suffers from some limitations still. In 2DGS, surfaces are assumed to always be at full opacity, which means that there will be problems when dealing with glass and other semi-transparent surfaces. Another limitation of 2DGS is that through its usage of depth distortion regularization and oversmoothing, the reconstruction tends to be less accurate on structures with fine details. There is a trade-off between image quality and geometry happening here; fixed, smooth structures are more preferred and due to this, the reconstruction process will view extra details as mere noise and smooth it over.

4.2 RaDe-GS

Rasterizing Depth in Gaussian Splatting (RaDe-GS) [Zha+24] is a further attempt to improve geometric reconstruction in Gaussian Splatting while addressing problems faced by 2DGS with quality reductions and rendering efficiency. This is done by creating a rasterized approach to render depth maps and surface normal maps of 3D Gaussians. Through this strategy, RaDe-GS is able to improve the reconstruction accuracy and computational efficiency and even achieve a Chamfer distance error comparable to NeuraLangelo [Li+23], a Neural Surface Reconstruction method spearheaded by projects such as NeuS [Wan+23] and UNISURF [OPG21], on the DTU dataset.

4.2.1 Splatting Method

RaDe-GS closely follows basic 3DGS method [Ker+23] for scene representation using 3D Gaussians. The usage and definition of 3D Gaussians stays the same in both works, and RaDe-GS also employs approximations for perspective camera projections and the usage of alpha blending to determine color (see 3DGS explanation for complete equation on 3D Gaussians, approximation of local affine projections, and alpha blending).

4.2.2 Rasterizing Depth for Splats

Unlike its predecessors, RaDe-GS adopts a rasterized approach for the efficient evaluation of the depth of each Gaussians. This is done in order to preserve the finer details of the objects being reconstructed. With this method, the depth calculation of a projected 2D Gaussian is as follows:

$$d = z_c + p \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix},$$

where (u_c, v_c) is the center point of the Gaussian, z_c is the depth of the center point of the Gaussian, $\Delta u = u_c - u$ and $\Delta v = v_c - v$ are relative positions of the pixel in coordinate point (u, v), and $p \in \mathbb{R}^2$ is a 1×2 vector determined by Gaussian parameters and camera extrinsic parameters.

Surface normal directions as projected by the Gaussians are then calculated by taking intersecting points and forming a plane in ray space with these points (indicated by the green line in Figure 4.1).

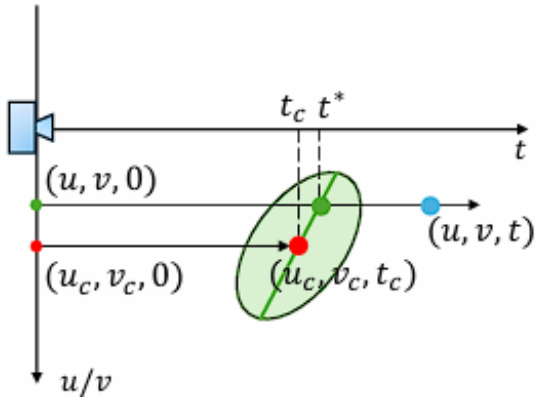


Figure 4.1: Plane formed in ray space by intersecting points. [Zha+24]

The normal direction of the projected Gaussian can then be regarded as the plane's normal direction and then transformed into camera space for normal map computation.

4.2.3 Limitations of RaDe-GS

Although able to reproduce highly-sophisticated depth information during the reconstruction process and providing a solid improvement to the general 3DGS method, there are some questions that arose from the usage of RaDe-GS in more realistic scenarios. First, even though RaDe-GS is able to provide more sophisticated geometrical details when compared to previous methods like 2DGS that suffers from oversmoothing, it may still be unable to perform well in scenes with a lot of noise, since it does not employ any particular strategies to filter out noise. Moreover, it still does not address the problem with special types of reflective surface that 2DGS also suffers from.

4.3 GS2Mesh

GS2Mesh [WBK24] is another novel approach aimed to address the issue of accurate geometric representation in Gaussian Splatting methods. Unlike the traditional Gaussian Splatting method [Ker+23] where geometry is immediately extracted out of the properties of the created Gaussians, GS2Mesh aims to do this by extracting geometry through a pre-trained stereo-matching model instead. Stereo-aligned pairs of images corresponding to the original poses are rendered and fed into a stereo model to get depth information and then fused together to form a single mesh. This results in a much smoother and more accurate reconstruction that still maintains intricate details when compared to 2DGS [Hua+24].

4.3.1 GS2Mesh Pipeline

GS2Mesh accepts videos or a collection of images of a static scene as input. This input is then passed on to COLMAP [SF16] [Sch+16], a general-use Structure-from-Motion (SfM) pipeline that is used to deduce camera matrices and identify corresponding image pairs and also points of interests from the corresponding input images.

Afterwards, the output from COLMAP is then passed on to the 3DGS model [Ker+23], where 3D Gaussians are identified and optimized based on photometric loss of the input images when compared to the corresponding rendered images. A scene reconstruction is then created in this process. GS2Mesh pipeline then generates novel stereo views of the scene to form an image pair of the same pose (R_L, T_L) to be paired together. The left image is first generated directly from the training images, and the right images are then created by shifting it along a horizontal baseline b in such a way that the pair is stereo-calibrated:

$$R_R = R_L, T_R = T_L + (R_L \times [b, 0, 0]),$$

with R as the rotation matrix of the pose and T the translation vector.

Next, the stereo-calibrated image pairs are inputted into a stereo matching algorithm with the objective of forming depth profiles for every image pairs. Here, the DLNR model [Zha+23] is used, as it can be considered as the state-of-art neural stereo matching model at the time. During this process, an occlusion mask is first applied to mask parts of the scene that is only visible from one side of the cameras. Consequently, a second mask that filters out objects that are too close to the camera is applied. This is done since current stereo matching algorithms are inaccurate when dealing with such objects; the stereo matching error itself can be described as such:

$$\epsilon(Z) \approx \frac{\epsilon(d)}{f_x \cdot B} Z^2 \text{ [Bra+21]},$$

with $\epsilon(d)$ as disparity output error, Z as ground truth depth, $\epsilon(Z)$ as error of depth estimate, f_x as the camera's horizontal focal length, and B as baseline, which with further experiments is then fixed at 7% of the scene radius. Thus, simply eliminating the objects with the hopes that it will be covered by a more appropriate image pair is reasonable to improve the accuracy of the model. Finally, a Truncated Signed Distance Function (TSDF) algorithm [CL23] is applied, followed by the Marching-Cubes meshing algorithm [LC87] in order to generate a polygonal mesh as output.

4.3.2 Limitations of GS2Mesh

Since GS2Mesh mainly focuses on creating a pipeline that would maximize the traditional 3DGS method, it still retains some of the weaknesses of 3DGS. One weakness is that it tends to produce noisy results, especially in areas that are not sufficiently covered by the input images. It also still has not addressed the main limitation of more recent models like 2DGS and RaDe-GS: transparent surfaces. Furthermore, the usage of TSDF fusion process is not efficient for larger inputs; rendering these take up a large amount of resources and is extremely inefficient.

5 Methodology

In this chapter, we provide a detailed step-by-step of how we extracted depth values from our synthetic model and adapted the GS and MDE codes to accept synthetic data. We also provided our analysis and data visualization process in great detail.

5.1 Dataset Creation

In this section, we explain how we selected a synthetic model and prepared it to be used for the training process of the 3DGS and MDE models.

5.1.1 Model Selection

Figure 5.1: Chocolate Easter Bunny model by Louis Bidou

Since this is an exploratory project aimed to test out different Gaussian Splatting models and pipelines, an object model had to be chosen to serve as an input for all the monocular depth estimation and Gaussian Splatting models being tested. A synthetic model is chosen since it is easier to gain Ground Truth data on it to be compared to the

outputs of the models. In the interest of doing a simple evaluation to compare between depth values generated from the GS and depth estimation models, a simple object with not too many features and colors is chosen: the Chocolate Easter Bunny created by Louis Bidou (2015), licensed under CC BY 4.0 (see figure 5.1).

5.1.2 Preprocessing with Blender

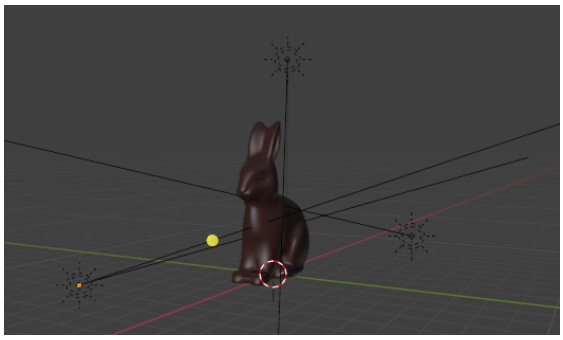


Figure 5.2: Addition of lighting in Blender to illuminate the model

In order to compare the accuracy between models and also use this object model as an input for the monocular depth estimation and Gaussian Splatting models, the object model is first loaded inside Blender 4.3, an software for 3D modeling and simulations. Blender is chosen since it is a free and open-source software that has in-built Python scripting. Some lighting aimed at the Chocolate Bunny model is then added to illuminate the model's surrounding. This is done so that the model would have sufficient illumination during the rendering phase and would not appear dark. Proper illumination is needed to ensure that the training phase of each models would run well.

5.1.3 Extraction of Synthetic Depth Data (Ground Truth Data) and Input Dataset Creation from the Model

Gaussian Splatting models require as input a collection of 2D images, along with its camera parameters. For this purpose, we create a Python script to process the model into a set of images to be used as input for the Gaussian Splatting models. The main goal of the script is to systematically take multiple images of the model following a spherical "grid" that surrounds the object and then save the images under a folder with a randomized name alongside with its camera parameters. This is done in order to capture the model from multiple different angles as best as possible while keeping

the object in its entirety in the frame, so as not to produce a bad input that cannot be identified well by the Gaussian Splatting pipeline. Aside from storing images and camera parameters, the script also stores the absolute depth of each produced image in a separate folder. This allows us to view information of the ground truth depth map, which serves as a valuable point of evaluation for the Gaussian Splatting models.

With this whole thing in mind, the script is created as follows: given two points in the Cartesian coordinate system, a box-like region of interest (where the object is located) is first identified, and from this, we are able to pinpoint the center of the object in the coordinate system. Afterwards, the $x-, y-, z-$Radius values of our spherical grid are calculated by looking at its difference to the extremes (the given input) of the box-like region. Next, the script requires two more input values to be specified to determine the number of images it needs to create as output: the number of "vertical steps" (height values) the spherical object will be divided into, and the number of captures (horizontal steps) we require per height values. In total, a number of $verticalSteps \times horizontalSteps$ images are created. In the case of our Chocolate Bunny model, 6 different height values and 20 horizontal steps are specified, resulting in a total of 120 images.

Following this, we calculate the position of each camera. Each camera in principle is aimed at the center of the object, but they are placed in different locations according to the current height value and horizontal step combination. Hence, in order to calculate this, we do the following steps:

1. We translate the object such that it lies in the center origin point of the Cartesian plane.

2. We place an initial point a certain $z-$value length away from the center of the object. The length is calculated by considering the maximum value between the 3 different radius values calculated previously. We only consider the maximum value since our main goal is only to fit every part of the object into the camera frame. In the formula above, we also include a custom multiplier (to manually add some distance to the model), while the 'FoV' (Field of View) value is set according to the standard FoV in Blender ($50.0°$).

```
maxRad = max(xRadius, yRadius, zRadius)
alpha = math.sin(FoV)
maxDist = Multiplier * (maxRad / alpha) * math.sqrt(1 - \
    (alpha * alpha))
```

3. We rotate the point around the $y-$Axis for $180.0° / (heightValue + 1)$.

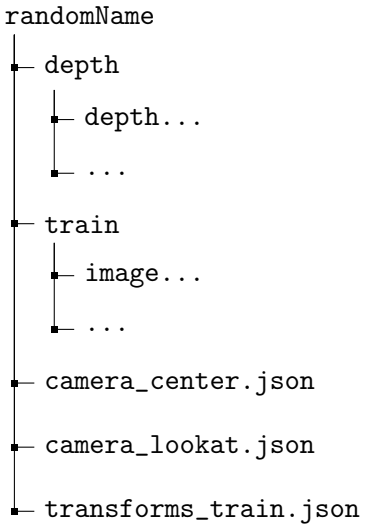


Figure 5.3: File structure of the result of the Blender script for Dataset Preparation

4. We rotate the point around the $z-$Axis for $360.0° / (horizontalStep)$.

Afterwards, it is only a matter of saving the resulting images inside the specified folder, alongside with its camera parameters and depth information (to serve as GT). The structure of the resulting output can be seen in figure 5.3. In this way, we have achieved our original goal of producing a set of input images for our Gaussian Splatting models and, along with it, saved the ground truth depth images to serve as a point of reference.

5.2 Code Adaption

In order to ensure that the code runs perfectly on the synthetic data we produced from our Blender script, tiny adaptations are made in the code of each of the models we employed in this research.

The traditional 3DGS pipeline [Ker+23] originally only accepts inputs that have been inserted into COLMAP (see Figure 5.4). This needed to be adjusted since reinserting our rendered images once more to COLMAP serves no purpose when we essentially already have everything that is needed and asked for by the 3DGS pipeline (e.g. transformation matrix, Field of Vision (FoV) values, camera angles). Therefore, some changes were made to the dataset_readerpy file of 3DGS to convert the camera information saved

by Blender 4.3 into the format demanded by 3DGS to force it to accept synthetic information.

Similarly to how this is done in 3DGS, we also adapt the dataset reader files of 2DGS, RaDe-GS, GS2Mesh, ZoeDepth, and DepthAnything (both v1 and v2). Code adaptions and other scripts used in this thesis is made available in the following github repository: https://github.com/evelynsidarta/gaussian-splatting-thesis.

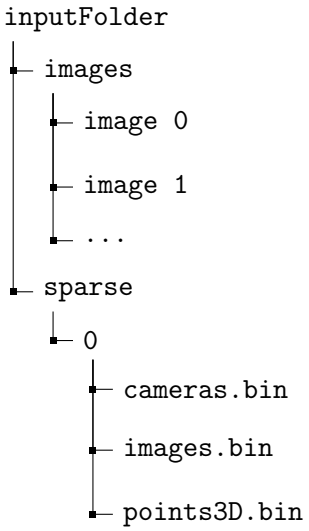


Figure 5.4: File structure expected by the COLMAP loader of 3DGS

5.3 Training Process

Training is done using the slightly-adapted code cloned from the respective github repositories of each model. For the number of training iterations, each model's number of training iterations is fixed at 10000 in order to give a fair evaluation when compared to each other. After the training is done, similar to gs2mesh, we render each outcome using the renderer script from 2DGS, since it also renders the depth map with the same camera poses as the original training images as a byproduct, which makes it a lot easier to do the evaluation step and compare the rendered outcome to the ground truth. The same number of images are therefore taken from the output model (120) after the rendering step.

5.4 Evaluation Methodology

5.4.1 Data Postprocessing

Since there might be slight differences in the output format of each model, some adjustments needed to be applied to the depth values of the outcomes of each model. To uniformize the outcome, it was decided that all of the depth map should be converted to inverted depth maps, which means that infinite depth (or empty spaces) will result in a depth value of 0, and objects that appear nearer to the camera will have a higher depth value when compared to objects that are farther away. Inverted depth maps have an advantage of having better numerical stability when compared to normal depth maps, especially for our Chocolate Easter Bunny model, since it has no background and comparing infinite values will only jeopardize the evaluation metrics. In order to invert the depth value, the following formula is applied to the resulting depth value outcomes:

$$x' = \frac{1}{x+\epsilon},$$

with x as the original value of the outcome, x' as the new inverted value, and $\epsilon = 10^{-3}$, which is a small margin that is added to prevent division by zero. Overall, depth inversion is applied to the ground truth images, as well as 3DGS, 2DGS, ZoeDepth, RaDe-GS, and GS2Mesh images. This depth inversion is not applied to the DepthAnything v1 and v2 outcome images since they are already inverted from the start.

Other than depth inversion, normalization using min-max scaling is also applied to all the outcome images in order to generalize the outcome values to be comparable. Normalization is done with the following formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}},$$

with x_{min} as the current lowest value of the depth map (the background value, since it is fixed at 0) and x_{max} as the current highest value in the depth map.

5.4.2 Evaluation Metrics

In order to provide a thorough analysis of the depth results of the rendered models, and inspired by the evaluation method of ZoeDepth [Bha+23], each of the 120 depth images from the outcome are compared to the ground truth depth values and the following error metrics are calculated and averaged for each model: 1) Scale Invariant Logarithmic (SiLog) loss, 2) Threshold accuracy $(\delta_1, \delta_2, \delta_3)$, 3) Root Mean Squared Error (RMSE), 4) Root Mean Squarde Log Error (RMSLE), 5) Absolute Relative Error (REL), 6) Logarithmic Error, and 7) Relative Square Error (RSE).

Absolute Relative Error (REL)

Absolute Relative Error (REL) measures the absolute difference between the predicted and the ground truth depth value when compared to the ground truth depth value. Absolute Relative Error is calculated as follows:

$$\text{REL} = \frac{1}{M} \sum_{i=1}^{M} \frac{|d_i - \hat{d}_i|}{\hat{d}_i},$$

with d_i as predicted depth value at the i-th pixel, $\hat{d}_i$ as ground truth depth value at the i-th pixel and M as the total amount of pixels. A lower REL value generally indicates better performance.

Scale Invariant Logarithmic Loss

Scale Invariant Log loss (SiLog) is a metric often used to evaluate Monocular Depth Estimation models. MDE is an ill-posed problem, which means that there is an infinite amount of geometrical solutions due to scaling. Therefore, SiLog loss function is created to minimize the impact of this scaling problem and only consider relative error in the logarithmic space instead of directly comparing absolute values. The formula for SiLog error is as follows:

$$\text{SiLog} = \sqrt{\frac{1}{M} \sum_{i=1}^{M} (\ln d_i - \ln \hat{d}_i)^2 - \frac{1}{M^2} (\sum_{i=1}^{M} (\ln d_i - \ln \hat{d}_i))^2},$$

with d_i as the predicted depth value at the i-th pixel, $\hat{d}_i$ as the ground truth value at the i-th pixel, and M as the total amount of pixels. SiLog value gets higher along with the value discrepancies, so a lower SiLog value is generally preferred.

Threshold Accuracy

Threshold accuracy typically evaluates how well the predicted depth map values align with the ground truth values. It measures the percentage number of pixels that are within a factor of δ^n when compared to the ground truth values. The calculation for threshold accuracy is as follows:

$$\delta_n = \% \text{ of pixels that satisfy } max(\frac{d_i}{\hat{d}_i}, \frac{\hat{d}_i}{d_i}) < \delta^n \text{ [Bha+23]},$$

with d_i as predicted depth value at the i-th pixel and $\hat{d}_i$ as ground truth depth value at the i-th pixel. The value of δ is set to be 1.25, which means that δ_1 evaluates the percentage number of pixels whose value lie within 25% of the ground truth value,

while δ_2 and δ_3 respectively evaluate the number of pixels whose value lie between 56.25% and 95.31% of the value of the ground truth. For threshold accuracy, the closer the value is to 100%, the better the performance of the evaluated model is, since it means that the predicted values are close to the values of the ground truth.

Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is an error that describes absolute differences between ground truth values when compared to the predicted values. Here, high discrepancies are highlighted due to the squaring operation in the formula. RMSE is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(d_i - \hat{d}_i)^2},$$

with d_i as predicted depth value at the i-th pixel, $\hat{d}_i$ as ground truth depth value at the i-th pixel and M as the total amount of pixels. Since this metric value gets higher along with the value discrepancies, a lower RMSE value is generally preferred and indicate better performance.

Root Mean Squared Logarithmic Error (RMSLE)

Root Mean Squared Logarithmic Error works in a similar manner to RMSE, but the values are first log-transformed before being compared to the also-transformed ground truth value. RMSLE is useful to proportionally scale higher discrepancies compared to just taking the absolute difference calculation. RMSLE is measured as follows:

$$\text{RMSLE} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(\ln(d_i) - \ln(\hat{d}_i))^2},$$

with d_i as predicted depth value at the i-th pixel, $\hat{d}_i$ as ground truth depth value at the i-th pixel and M as the total amount of pixels. Similarly to RMSE, a lower RMSLE value is preferred during evaluation.

Logarithmic Error

Logarithmic error measures the log-transformed difference between the predicted and ground truth depth values. Logarithmic error is measured as follows:

$$\text{Log Error} = \frac{1}{M}\sum_{i=1}^{M}|\log_{10}(d_i) - \log_{10}(\hat{d}_i)|,$$

with d_i as predicted depth value at the i-th pixel, $\hat{d}_i$ as ground truth depth value at the i-th pixel and M as the total amount of pixels. A lower logarithmic error value indicates better model performance.

Relative Square Error (RSE)

Relative Square Error (RSE) works similarly to absolute relative error, but instead of calculating absolute difference it squares the result, which means that it magnifies error values more. RSE is calculated as follows:

$$\text{RSE} = \frac{1}{M} \sum_{i=1}^{M} \frac{(d_i - \hat{d}_i)^2}{\hat{d}_i^{\,2}},$$

with d_i as predicted depth value at the i-th pixel, $\hat{d}_i$ as ground truth depth value at the i-th pixel and M as the total amount of pixels. A lower RSE indicates better performance.

5.5 Data Display

Aside from statistical evaluation, we also compare the results of each model visually. In order to do this, samples from each models are gathered and compared side-by-side to each other with pyplot. For all the depth maps except for the Depth Anything models, the depth is inverted, similarly to how it is done during evaluation process. For 3DGS, RaDe-GS, and GS2Mesh, aside from the usual depth map inversion to uniformize the depth maps, the sigmoid transfer function is applied. This is done in order to better show the differences in values, since these three depth maps have only very slight differences between values inside the depth map.

5.5.1 Sigmoid Transfer Function

Sigmoid transfer function is commonly used to address the vanishing gradient problem: when inputs are too extreme, it is often hard to see tiny changes in the value. The sigmoid transfer function addresses this problem by compressing extreme values in an image and expanding the middle range of the values. Values after applying the sigmoid transfer function is calculated as follows:

$$\sigma(x) = \frac{1}{1+e^{-k\cdot(x-x_0)}},$$

with x_0 as the "middle range" we are trying to expand and k as the steepness of the sigmoid curve.

6 Evaluation

6.1 Statistical Analysis Results

Methods	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	REL $\downarrow$	RSE $\downarrow$	$\log_{10} \downarrow$	RMSE $\downarrow$	RMSLE $\downarrow$	SiLog $\downarrow$
3DGS	0.851	0.851	0.852	0.139	0.111	0.140	0.311	0.841	77.96
2DGS	0.855	0.855	0.855	0.131	0.107	0.139	0.311	0.838	77.50
RaDe	**0.856**	0.856	0.856	**0.128**	0.106	0.138	0.310	0.837	77.34
GS2M	0.853	0.853	0.853	0.155	0.124	0.138	0.309	0.833	77.04
DA1-s	0.807	0.903	**0.957**	0.143	**0.027**	**0.048**	0.097	0.252	24.71
DA1-b	0.828	**0.907**	0.948	0.154	0.032	0.049	**0.095**	**0.251**	**24.25**
DA1-l	0.725	0.831	0.898	0.278	0.061	0.081	0.118	0.359	33.88
DA2-s	0.561	0.688	0.788	0.502	0.094	0.140	0.127	0.503	43.32
DA2-b	0.583	0.710	0.811	0.443	0.079	0.129	0.127	0.472	41.95
DA2-l	0.613	0.745	0.850	0.354	0.058	0.110	0.124	0.419	38.47
ZoeD	0.472	0.620	0.742	0.679	0.163	0.179	0.159	0.581	47.51

Table 6.1: Statistical analysis on GT values

Table 6.1 shows the result of our statistical analysis on all the MDE and Gaussian Splatting models. From the results above, it can be noted that RaDe-GS seem to have the overall best performance when compared to other Gaussian Splatting models. This seem to be consistent with evaluations done by the authors of RaDe-GS [Zha+24]. RaDe-GS is able to produce more detail-sensitive results when compared to 2DGS, GS2Mesh, and the original 3DGS method. Most notably, RaDe-GS is able to achieve the lowest absolute relative error, which means that the RaDe-GS model produce values that most closely resembles the absolute value of the Ground Truth. It also achieves the highest δ_1 score, which means that out of all of the models, it has the highest percentage of values closely resembling the ground truth.

2DGS, on the other hand, delivers a strong performance and seems to be the overall runner-up amongst the Gaussian Splatting models, although 3DGS and GS2Mesh seem to still perform closely to the other two, with 3DGS performing the worst out of all

the Gaussian Splatting models. These findings are consistent with the expected results; 3DGS have a less optimized algorithm, since it is the predecessor of all of the GS models, and GS2Mesh suffer from the same drawbacks as the original 3DGS model, since it employs 3DGS as the core of its pipeline.

When compared to the MDE models, the GS models seem to be competitive. GS models seem to have higher δ_1, δ_2, and δ_3 values overall, even though the early Depth Anything v1 models seem to be performing well too in this measurement.

Most surprisingly, DepthAnything v1 models seem to be delivering a stronger performance than its successor, the DepthAnything v2, even though DepthAnything v2 is also trained on synthetic data. This may be due to DepthAnything v2 trying to produce more detailed results, which in turn spoiled its evaluation scores since the model being tested is an extremely simple model. ZoeDepth does not seem to be performing well too on the tests. This may be due to ZoeDepth's tendency to add some noise in the background, which might make sense when dealing with real life data, but does not work well when dealing with models with no background.

Overall, the most accurate depth values seem to be generated by the DepthAnything v1 base model, with it having the overall lowest RMSE, RMSLE, and SiLog values and the highest δ_2 value. The GS models seem to perform poorly on the SiLog evaluation, which mean that the GS models might be consistently off by some factor, even though this factor seem to not be significant when compared to the ground truth value, since the GS models have high threshold accuracy percentages.

6.2 Visual Analysis Results

In order to facilitate the visual analysis of the depth maps, some transfer functions to invert the depth maps and to adjust extreme values have been applied. Figure 6.1 displays the result of one of the poses used for evaluation, placed side-by-side for ease of viewing.

From the results, we can see that the ground truth image generally possess a very basic form with not much variation in depth values. At a glance, out of all the Gaussian Splatting models, the 2DGS model seem to be performing the best visually, since the splats seem to be smoother and deviate less from the general shape of the object. It seem to also be quite detailed, since the folds of the ears are given different depth values and it is possible to visually tell that these details exist solely by looking at the depth map. The legs and the bumps in the bunny's tail is also expressed nicely in the 2DGS model, and we can tell that the bunny is facing to the back. However, we can also tell that 2DGS tends to apply smoothing to the image; the chocolate bunny that is supposed to be a bit flatter is illustrated as being more curvy and more like a real-life

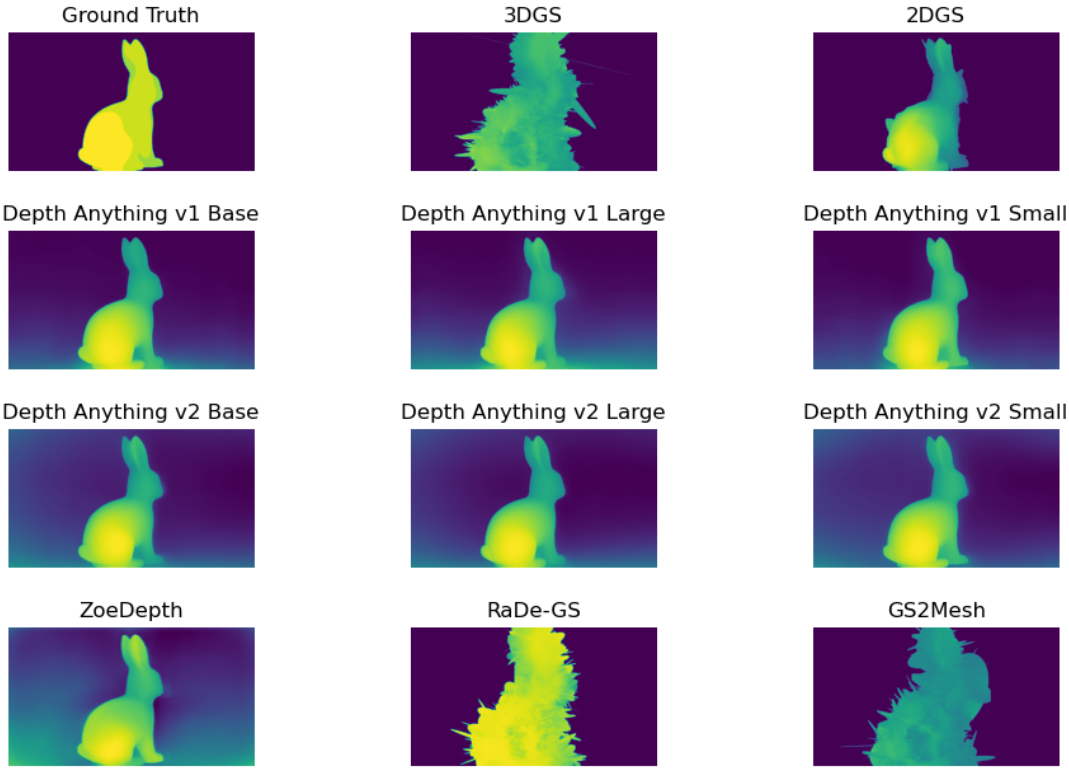


Figure 6.1: Visual comparison of Chocolate Easter Bunny, facing to the back

bunny with the 2DGS model. This may be the reason that it performs more poorly in the statistical analysis; the ground truth value demands a flatter distribution when compared to the 2DGS bunny.

In contrast to the 2DGS result, the original 3DGS model seem to be very rough and unrefined; the general shape of the bunny can be seen from the depth image, but details are not clear at all. The bunny also does not appear to be facing the back, but rather it creates the illusion that the bunny is facing to the right instead. Furthermore, the splats are very all over the place and often sticks out at a weird angle. This might be due to the model misclassifying general lighting as object.

Along with the 3DGS model, the GS2Mesh model also seem to be doing poorly; a weirdly-placed splat can be seen around the head area of the bunny and the details are missing, although overall it seem to have a better overall shape than the 3DGS model. The GS2Mesh result also does not seem to be looking to the back, but rather to the side like its 3DGS counterpart.

RaDe-GS, on the other hand, seem to be performing pretty well; it seems to preserve

the flatter values of the original training image better than even the 2DGS result, which may be why it performs the best during the statistical evaluation. Visually, though, it is hard to see that the bunny is facing the back, even though it is still more detailed and value-accurate when compared to 3DGS and GS2Mesh. The RaDe-GS result also has a better overall shape compared to 3DGS and GS2Mesh.

Compared to the GS models, the MDE models tend to produce a very smooth and detailed depth map, since here we do not suffer from having any Gaussians floating around the depth map. However, this also makes them suffer from the same problem that the 2DGS model suffers from: oversmoothing the values is not ideal since the Chocolate Bunny model is flatter in general. When compared to each other, it is clear that the DepthAnything v2 large model offers the most-detailed depth values; the folds of the ears are apparent and the tail of the bunny is sculpted more. However, due to these details, the DepthAnything v2 large performs more poorly in the statistical analysis; the small DepthAnything v1 model tends to provide less details, which mean less oversmoothing happened and the images are flatter in general. In Figure 6.2, we highlight the difference in details between the DepthAnything v2 large model and the DepthAnything v1 small model. The red rectangle shows the ear curves, while the blue rectangle shows the oversmoothing problem that the larger models suffer from. Another thing that is detrimental to larger and more recent DepthAnything models is the background; they tend to "bleed", meaning some noise were added into the depth map, while the smaller, older models tend to be less expressive in general.

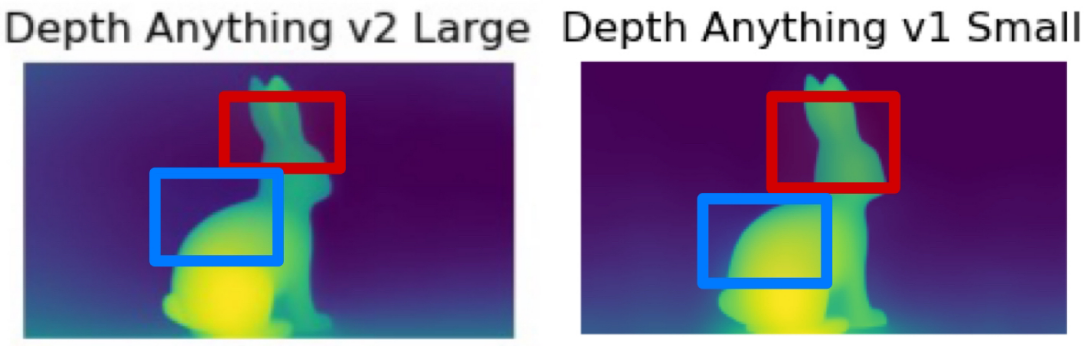


Figure 6.2: Highlighting discrepancies between DA v2 large model and DA v1 small model

The ZoeDepth model, while offering a pretty detailed depth map, suffers more from this background bleeding problem. We can see that the ZoeDepth background tends to be very noisy in general. While this might have some use in real life datasets, this is, of course, detrimental when dealing with models with no background.

6.3 Limitations of Current State-of-Art Methods

From the results of our analysis, we can see that current state-of-art methods were able to perform well on our simple model. Nonetheless, we acknowledge that there are still some weaknesses to the current models that can be observed from our results. First, MDE models seem to struggle from the background bleeding, since it is not used to dealing with situations that are unrealistic, which is the case in our current model, as it is a synthetic model placed on a completely silent background. In contrast to this, the GS models seem to suffer more from the awkward shape of the Gaussians; they tend to be less smooth than expected, even though the overall shape of the objects seem to be retained well. Due to the density of the Gaussians, GS models often appear much more noisy than they really are, since Gaussians with low opacity still has depth values that needed to be considered for the evaluation.

7 Conclusion

7.1 Conclusion on the Results of the Research

In this thesis, we have explored some of the current state-of-art Gaussian Splatting and Monocular Depth Estimation models and evaluated it on different metrics, both visual and statistical. Our research started by doing a thorough review of the current available Gaussian Splatting and Monocular Depth Estimation methods, carefully dissecting each layer of the main method and their optimizations and laying them out as groundwork for our evaluation.

Afterwards, we perused some synthetic models to find a simple model for our evaluation; the Chocolate Easter Bunny was chosen and synthetic data was extracted from it with the help of Blender 4.3. Next, we trained the chosen GS and MDE models on our bunny images and compared the results against our chosen evaluation benchmarks. Here, we were able to compare both visual and statistical results of our trained models, thus highlighting the exceptional performances of RaDe-GS, 2DGS, and also the MDE models like DepthAnything.

7.2 Suggestions for Further Improvements to the Research

We recognize nonetheless that there is a lot of room for improvement that can be made from this thesis. First, we recognize that using only the Chocolate Easter Bunny model as subject for evaluation is highly detrimental to our evaluation; some models simply do not perform well statistically on synthetic data, and small problems like background bleeding ended up being more detrimental to the overall performance of the models. Creating and using more datasets as evaluation material will improve the reliability of our evaluation. It may also be beneficial to include datasets that depict real-life scenarios, since these models are mostly trained for that purpose.

Furthermore, simply comparing depth maps is not a good indicator of the overall quality of Gaussian Splatting models, since rogue splats with very low opacity that end up not mattering at all when looking at rendered results may end up severely affecting the results of the evaluation. It would be nice to devise some methods to take into more account the visibility of the Gaussians when comparing the depth maps. A

more comprehensive evaluation method needs to be devised in order to more properly evaluate these models.

In the future, we would like to also be able to include more models in our evaluation; Gaussian Splatting is still a very rapidly expanding method, since it is relatively new. There are still a lot of room for improvements on the current models, especially on more specific points of interest such as dealing with highly reflective materials and transparent objects. Being able to include more models will allow for a more thorough overview of the GS method to be laid down.

List of Figures

List of Tables

Bibliography

[AW19] I. Alhashim and P. Wonka. *High Quality Monocular Depth Estimation via Transfer Learning*. 2019. arXiv: 1812.11941 [cs.CV].

[Axl23] S. Axler. *Linear Algebra Done Right*. Springer Cham, 2023.

[Bar+22] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. *Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields*. 2022. arXiv: 2111.12077 [cs.CV].

[Bha+23] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller. *ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth*. 2023. arXiv: 2302.12288 [cs.CV].

[Bot+05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. "High-quality surface splatting on today's GPUs." In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* 2005, pp. 17–141. DOI: 10.1109/PBG.2005.194059.

[Bra+21] A. Bracha, N. Rotstein, D. Bensaïd, R. Slossberg, and R. Kimmel. *Depth Refinement for Improved Stereo Reconstruction*. 2021. arXiv: 2112.08070 [cs.CV].

[Bri08] R. Brinkmann. "CHAPTER TWO - Learning to See." In: *The Art and Science of Digital Compositing (Second Edition)*. Ed. by R. Brinkmann. Second Edition. The Morgan Kaufmann Series in Computer Graphics. Boston: Morgan Kaufmann, 2008, pp. 15–51. ISBN: 978-0-12-370638-6. DOI: https://doi.org/10.1016/B978-0-12-370638-6.00002-X.

[But+12] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. "A Naturalistic Open Source Movie for Optical Flow Evaluation." In: *Computer Vision – ECCV 2012*. Ed. by A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 611–625. ISBN: 978-3-642-33783-3.

[BWM23] R. Birkl, D. Wofk, and M. Müller. *MiDaS v3.1 – A Model Zoo for Robust Monocular Relative Depth Estimation*. 2023. arXiv: 2307.14460 [cs.CV].

[Cam16] Cambridge Dictionary. *Parallax*. Accessed: 2025-03-18. Cambridge University Press, 2016.

[Cha+17] X. Chai, F. Gao, C. Qi, Y. Pan, Y. Xu, and Y. Zhao. "Obstacle avoidance for a hexapod robot in unknown environment." In: *Science China Technological Sciences* (2017). DOI: 10.1007/s11431-016-9017-6.

[Che+22] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su. *TensoRF: Tensorial Radiance Fields*. 2022. arXiv: 2203.09517 [cs.CV].

[Che+24] K. Cheng, X. Long, K. Yang, Y. Yao, W. Yin, Y. Ma, W. Wang, and X. Chen. "GaussianPro: 3D Gaussian Splatting with Progressive Propagation." In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp. Vol. 235. Proceedings of Machine Learning Research. PMLR, 21–27 Jul 2024, pp. 8123–8140.

[CL23] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images." In: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 1st ed. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400708978.

[COL24] J. Chung, J. Oh, and K. M. Lee. *Depth-Regularized Optimization for 3D Gaussian Splatting in Few-Shot Images*. 2024. arXiv: 2311.13398 [cs.CV].

[CRF24] C. Cao, X. Ren, and Y. Fu. *MVSFormer++: Revealing the Devil in Transformer's Details for Multi-View Stereo*. 2024. arXiv: 2401.11673 [cs.CV].

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals and Systems* (1989). DOI: 10.1007/BF02551274.

[FSG16] H. Fan, H. Su, and L. Guibas. *A Point Set Generation Network for 3D Object Reconstruction from a Single Image*. 2016. arXiv: 1612.00603 [cs.CV].

[Gei+13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. "Vision meets robotics: The KITTI dataset." In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. DOI: 10.1177/0278364913491297. eprint: https://doi.org/10.1177/0278364913491297.

[Gui+20] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon. *3D Packing for Self-Supervised Monocular Depth Estimation*. 2020. arXiv: 1905.02693 [cs.CV].

[Gui+23] V. Guizilini, I. Vasiljevic, D. Chen, R. Ambrus, and A. Gaidon. *Towards Zero-Shot Scale-Aware Monocular Depth Estimation*. 2023. arXiv: 2306.17253 [cs.CV].

[Hed+21] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. *Baking Neural Radiance Fields for Real-Time View Synthesis*. 2021. arXiv: 2103.14645 [cs.CV].

[Hof91] N. Hoffmann. *Simulation Neuronaler Netze*. Wiesbaden: Vieweg+Teubner Verlag, 1991. ISBN: 978-3-322-87789-5. DOI: 10.1007/978-3-322-87789-5.

[Hu+12] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake. "A robust RGB-D SLAM algorithm." English. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE International Conference on Intelligent Robots and Systems. 25th IEEE/RSJ International Conference on Robotics and Intelligent Systems, IROS 2012 ; Conference date: 07-10-2012 Through 12-10-2012. United States: Institute of Electrical and Electronics Engineers, Dec. 2012, pp. 1714–1719. ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6386103.

[Hu+24a] J. Hu, X. Chen, B. Feng, G. Li, L. Yang, H. Bao, G. Zhang, and Z. Cui. *CG-SLAM: Efficient Dense RGB-D SLAM in a Consistent Uncertainty-aware 3D Gaussian Field*. 2024. arXiv: 2403.16095 [cs.CV].

[Hu+24b] M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, H. Chen, K. Wang, G. Yu, C. Shen, and S. Shen. "Metric3D v2: A Versatile Monocular Geometric Foundation Model for Zero-Shot Metric Depth and Surface Normal Estimation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (Dec. 2024), pp. 10579–10596. ISSN: 1939-3539. DOI: 10.1109/tpami.2024.3444912.

[Hua+24] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao. "2D Gaussian Splatting for Geometrically Accurate Radiance Fields." In: *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*. SIGGRAPH '24. ACM, July 2024, pp. 1–11. DOI: 10.1145/3641519.3657428.

[HZ04] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004.

[Kee+24] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten. *SplaTAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM*. 2024. arXiv: 2312.02126 [cs.CV].

[Ker+23] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. "3D Gaussian Splatting for Real-Time Radiance Field Rendering." In: *ACM Transactions on Graphics* 42.4 (July 2023).

[Lai+16] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. *Deeper Depth Prediction with Fully Convolutional Residual Networks*. 2016. arXiv: 1606.00373 [cs.CV].

[LC87] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm." In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. ISBN: 0897912276. DOI: 10.1145/37401.37422.

[Li+23] Z. Li, T. Müller, A. Evans, R. H. Taylor, M. Unberath, M.-Y. Liu, and C.-H. Lin. *Neuralangelo: High-Fidelity Neural Surface Reconstruction*. 2023. arXiv: 2306.03092 [cs.CV].

[Lin+21] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey. *BARF: Bundle-Adjusting Neural Radiance Fields*. 2021. arXiv: 2104.06405 [cs.CV].

[Lon87] H. Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections." In: *Readings in Computer Vision*. Ed. by M. A. Fischler and O. Firschein. San Francisco (CA): Morgan Kaufmann, 1987, pp. 61–62. ISBN: 978-0-08-051581-6. DOI: https://doi.org/10.1016/B978-0-08-051581-6.50012-X.

[Mar+21] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth. *NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections*. 2021. arXiv: 2008.02268 [cs.CV].

[Mil+20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV].

[Mül+22a] T. Müller, A. Evans, C. Schied, and A. Keller. "Instant neural graphics primitives with a multiresolution hash encoding." In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–15. ISSN: 1557-7368. DOI: 10.1145/3528223.3530127.

[Mül+22b] T. Müller, A. Evans, C. Schied, and A. Keller. "Instant neural graphics primitives with a multiresolution hash encoding." In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–15. ISSN: 1557-7368. DOI: 10.1145/3528223.3530127.

[OPG21] M. Oechsle, S. Peng, and A. Geiger. *UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction*. 2021. arXiv: 2104.10078 [cs.CV].

[Oqu+24] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV].

[Pau81] R. Paul. *Robot Manipulators: Mathematics, Programming, and Control : the Computer Control of Robot Manipulators*. Artificial Intelligence Series. MIT Press, 1981. ISBN: 9780262160827.

[PD84] T. Porter and T. Duff. "Compositing digital images." In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 253–259. ISSN: 0097-8930. DOI: 10.1145/964965.808606.

[Ran+20] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. *Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer*. 2020. arXiv: 1907.01341 [cs.CV].

[RH01] R. Ramamoorthi and P. Hanrahan. "On the relationship between radiance and irradiance: determining the illumination from images of a convex Lambertian object." In: *J. Opt. Soc. Am. A* 18.10 (Oct. 2001), pp. 2448–2459. DOI: 10.1364/JOSAA.18.002448.

[Sch+16] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm. "Pixelwise View Selection for Unstructured Multi-View Stereo." In: *European Conference on Computer Vision (ECCV)*. 2016.

[Sch+17] Schops, Schonberger, Galliani, Sattler, Schindler, Pollefeys, and Geiger. "A Multi-view Stereo Benchmark with High-Resolution Images and Multi-camera Videos." In: IEEE, 2017. DOI: 10.1109/cvpr.2017.272.

[SF16] J. L. Schönberger and J.-M. Frahm. "Structure-from-Motion Revisited." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Sig+06] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. "GPU-based ray-casting of quadratic surfaces." In: *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*. SPBG'06. Boston, Massachusetts: Eurographics Association, 2006, pp. 59–65. ISBN: 3905673320.

[Sil+12] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images." In: *Computer Vision – ECCV 2012*. Ed. by A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 746–760. ISBN: 978-3-642-33715-4.

[SKS23] P.-P. Sloan, J. Kautz, and J. Snyder. "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments." In: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 1st ed. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400708978.

[Sri+20] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron. *NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis*. 2020. arXiv: 2012.03927 [cs.CV].

[SS01] L. Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, 2001. ISBN: 9780130307965.

[Swo83] E. W. Swokowski. *Calculus with Analytic Geometry*. Alternate. Accessed: 2025-03-29. Prindle, Weber & Schmidt, 1983.

[Tan+19] J. Tang, X. Han, J. Pan, K. Jia, and X. Tong. "A Skeleton-Bridged Deep Learning Approach for Generating Meshes of Complex Topologies From Single RGB Images." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[Tan+21] M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng. *Learned Initializations for Optimizing Coordinate-Based Neural Representations*. 2021. arXiv: 2012.02189 [cs.CV].

[Tri+99] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. "Bundle Adjustment - A Modern Synthesis." In: *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. ICCV '99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 298–372. ISBN: 3540679731.

[Ull79] S. Ullman. "The interpretation of structure from motion." In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203 (1979), pp. 405–426.

[Vas+19] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter, and G. Shakhnarovich. *DIODE: A Dense Indoor and Outdoor DEpth Dataset*. 2019. arXiv: 1908.00463 [cs.CV].

[Wan+23] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang. *NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction*. 2023. arXiv: 2106.10689 [cs.CV].

[WBK24] Y. Wolf, A. Bracha, and R. Kimmel. *GS2Mesh: Surface Reconstruction from Gaussian Splatting via Novel Stereo Views*. 2024. arXiv: 2404.01810 [cs.CV].

[Wes91] L. A. Westover. *SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm*. Tech. rep. USA, 1991.

[Wil+20] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. *SynSin: End-to-end View Synthesis from a Single Image*. 2020. arXiv: 1912.08804 [cs.CV].

[Wu+24] T. Wu, Y.-J. Yuan, L.-X. Zhang, J. Yang, Y.-P. Cao, L.-Q. Yan, and L. Gao. *Recent Advances in 3D Gaussian Splatting*. 2024. arXiv: 2403.11134 [cs.CV].

[Xu+24] H. Xu, S. Peng, F. Wang, H. Blum, D. Barath, A. Geiger, and M. Pollefeys. *DepthSplat: Connecting Gaussian Splatting and Depth*. 2024. arXiv: 2410.13862 [cs.CV].

[Yan+24a] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li. *GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting*. 2024. arXiv: 2311.11700 [cs.CV].

[Yan+24b] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao. *Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data*. 2024. arXiv: 2401.10891 [cs.CV].

[Yan+24c] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. *Depth Anything V2*. 2024. arXiv: 2406.09414 [cs.CV].

[Yin+23] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, and C. Shen. *Metric3D: Towards Zero-shot Metric 3D Prediction from A Single Image*. 2023. arXiv: 2307.10984 [cs.CV].

[Yu+21a] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. *Plenoxels: Radiance Fields without Neural Networks*. 2021. arXiv: 2112.05131 [cs.CV].

[Yu+21b] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. *PlenOctrees for Real-time Rendering of Neural Radiance Fields*. 2021. arXiv: 2103.14024 [cs.CV].

[Yu+23] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger. *Mip-Splatting: Alias-free 3D Gaussian Splatting*. 2023. arXiv: 2311.16493 [cs.CV].

[Zha+20] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian. "Monocular depth estimation based on deep learning: An overview." In: *Science China Technological Sciences* 63.9 (June 2020), pp. 1612–1627. ISSN: 1869-1900. DOI: 10.1007/s11431-020-1582-8.

[Zha+23] H. Zhao, H. Zhou, Y. Zhang, J. Chen, Y. Yang, and Y. Zhao. "High-Frequency Stereo Matching Network." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 1327–1336.

[Zha+24] B. Zhang, C. Fang, R. Shrestha, Y. Liang, X. Long, and P. Tan. *RaDe-GS: Rasterizing Depth in Gaussian Splatting*. 2024. arXiv: 2406.01467 [cs.GR].

[Zhu+15] Z. Zhu, A. Su, H. Liu, Y. Shang, and Q. Yu. "Vision navigation for aircrafts based on 3D reconstruction from real-time imeage sequences." In: *Science China Technological Sciences* (2015). DOI: 10.1007/s11431-015-5828-x.

[Zwi+01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. "EWA volume splatting." In: *Proceedings Visualization, 2001. VIS '01.* 2001, pp. 29–538. DOI: 10.1109/VISUAL.2001.964490.

[Zwi+04] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly. "Perspective accurate splatting." In: *Proceedings of Graphics Interface 2004*. GI '04. London, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 247–254. ISBN: 1568812272.