ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Image Processing And Computer Vision

3D RECONSTRUCTION FROM UNCALIBRATED COLLECTIONS OF NORMAL MAPS

CANDIDATE
Edoardo Saturno

SUPERVISOR
Prof. Samuele Salti

CO-SUPERVISOR
Prof. Satoshi Ikehata

Academic year 2023-2024

Session 1st

Contents

List of Figures

List of Tables

Abstract

3D reconstruction is a fundamental task in computer vision that aims to generate accurate digital representations of real-world objects from 2D images. Traditional approaches, such as the *DUSt3R* method, rely on RGB images as input data, utilizing massive datasets to achieve significant results in reconstructing a wide variety of objects without the need of camera parameters information. Despite achieving excellent coverage of the original object's surface, these models still struggle to capture fine-level details. The proposed solution address this weakness by using a different kind of data for training the architecture. Normal maps are images that encode surface orientation vectors using colors and this work explores their potential as an alternative to traditional RGB-based datasets. The goal is to use the additional geometric information contained in normal maps to improve the performance of the *DUSt3R* method, enhancing 3D reconstruction accuracy for small surface details. While synthetic data is used for training, evaluation is conducted using real data, specifically from the *DiLiGenT-MV* dataset. Results indicate that when the model is trained with normal maps, the reconstruction accuracy of facial features, robe folds and other small surface details improves while maintaining good shape coverage of the entire object. These findings highlight the potential to overcome the limitations of previous approaches by incorporating richer geometric cues during model training. At the same time, opportunities for further improvements remain, such as modifying the training objective introducing normal-specific losses or using a combination of RGB and normal data for training.

Chapter 1

Introduction

In the field of *Computer Vision*, the process of capturing the shape and structure of real objects through computational methods is referred to as *3D Reconstruction*. Any related methodology involves interpreting 2D visual data, such as images or videos, to create a digital representation of the 3D world. These processes are of fundamental use in numerous applications, including virtual or augmented reality, medical imaging, and robotics. The ultimate goal of 3D reconstruction is to generate accurate and detailed models of the environment, enabling machines to perceive and interact with the world in three dimensions, much like humans do. *Multi-View Stereo (MVS)* is a computer vision technique used to reconstruct a 3D model of an object or scene from multiple 2D images taken from different viewpoints. It extends the concept of *stereo vision,* which typically uses two cameras, to multiple images in order to improve depth estimation and reconstruction accuracy. Traditional MVS approaches have primarily focused on using RGB images of a scene and face a key limitation: RGB images fail to accurately reconstruct fine level details of a surface leading to model with good shape coverage but suboptimal reconstruction accuracy. In recent years, with the availability of massive amounts of data, a shift in focus from architecture structure to data quality and quantity took place. It was noted that the quality and the amount of input data plays a more critical role than the choice of architecture, suggesting that high-quality

input can enhance 3D reconstruction results. A clear example of this can be observed with the *Dust3r* method [13], presented during the *2024 Conference on Computer Vision and Pattern Recognition*. The most remarkable result of this work is its ability to achieve good results in several 3D computer vision tasks without introducing architectural innovation. The network was able to show significant performance in the following tasks:

- **Absolute pose estimation**: the process of determining the position and orientation (pose) of a camera in a global coordinate system.
- **2D-2D pixel matching**: identifying and matching corresponding feature points between two or more images to establish geometric relationships.
- **Multi-view pose estimation**: estimating the camera pose across multiple views by analyzing correspondences between images.
- **Monocular depth estimation**: predicting scene depth from a single image.
- **Multi-view depth estimation**: estimating depth by analyzing multiple images of the same scene from different viewpoints.
- **3D reconstruction**: the process of generating a 3D model of a scene or object from 2D images, depth maps, or point clouds.

The contribution of *DUSt3R* is best described by the fact that the model was able to reach a good performance in the above downstream tasks without being explicitly fine tuned for them. The fact that no architectural innovation was introduced leads logically to focus on the data the model was trained on. The training set was comprised of multiple publicly available datasets, such as *Co3d* [10] and *Megadepth* [7] . This massive amount of data was able to help the architecture to generalize on many different domains, making it suitable for being used as a general purpose model. The aim of this dissertation is to

explore the possibility of using a new synthetic dataset, composed by images with a different information content than usual RGB ones, on the assumption that a successful method like *DUSt3R* would be able to extract meaningful information from it. The key idea is to leverage normal maps, a particular kind of images that encodes information about surface orientation vectors, to demonstrate that they contain information for 3D reconstruction and can potentially improve model performance. Normal maps contains valuable information about the orientation of object's surface, giving the model a chance to increase its performance on small level details, solving the weaknesses of previous approaches.

Chapter 2

Background

2.1 Camera Parameters

The transformation between a 3D scene and its 2D projection is governed by two types of camera parameters, extrinsic and intrinsic. The *intrinsic parameters* define the internal characteristics of the camera, which relate the 3D coordinates in camera frame to 2D pixel coordinates in the image domain. In practice, an intrinsic matrix maps 3D points in the camera coordinate system to 2D image points in pixel coordinates. The intrinsic matrix is typically represented as K, and can be expressed as:

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- f_x and f_y are the focal lengths (in pixels) along the x and y axes,
- c_x and c_y are the pixel coordinates of the principal point,
- γ is the skew coefficient.

The *extrinsic parameter*s describe the transformation between the camera's

coordinate system and the world coordinate system. They determine the camera's position and orientation in 3D space. The camera's extrinsic parameters can be represented by a single 3×4 transformation matrix:

$$[R \mid T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

where:

- R is a 3×3 rotation matrix that describes the orientation of the camera relative to the world coordinate system.
- T is a 3×1 translation vector that specifies the position of the camera in the world coordinate system.

Together, R and t define the rigid-body transformation from world coordinates to camera coordinates. The transformation is expressed as:

$$\mathbf{X_c} = R\mathbf{X_w} + T \tag{2.1}$$

where $\mathbf{X_c}$ and $\mathbf{X_w}$ are the points in camera and world coordinates, respectively. The overall transformation from world coordinates to image coordinates can be expressed as:

$$\mathbf{x} = K \begin{bmatrix} R & t \end{bmatrix} \mathbf{X_w} \tag{2.2}$$

2.2 Blender

Blender [8] is a popular computer modeling software widely used in both academic research and business for game development. In Blender, understanding the concepts of *global* and *local* coordinate systems is essential for accurately positioning objects, cameras, and vertices in 3D space. These reference

systems define how coordinates and transformations are interpreted, and they directly impact how vertices, objects, and rendered scenes are represented.

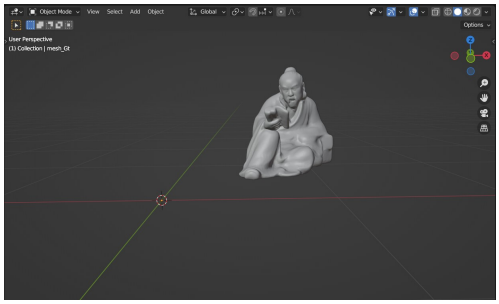

Figure 2.1: Object positioned in default Blender coordinate system

The *global reference system* (or world coordinate system) is the default coordinate system in Blender. It serves as a universal origin for positioning all objects, cameras, and lights in the 3D scene. It is comprised by the X, Y, Z axis and when something, like a vertex, is inserted somewhere in the scene, its position is defined with respect to the global origin. The *local reference system* (or object coordinate system) is specific to each individual object, including cameras. It is defined relative to the object's origin, which usually differs from the global origin. Every object in Blender has its own local X, Y, and Z axes that can be aligned, rotated, or translated independently of the global axes. As an example, if a vertex is positioned at $(1, 0, 0)$ in local coordinates the actual location of it is one unit along the local X-axis of the object to which the vertex belongs, regardless of where the object is in the global space.

2.3 Normal maps

A normal map is a textured representation of an object that encodes surface normals using RGB colors. The color of the rendered surface encodes the information relative to the orientation vectors of the surface itself. Aligning normal maps images consists in ensuring that surface normal vectors are oriented correctly and consistently across different views, which are the source from where the normal maps are obtained. Alignment with respect to a certain

coordinate system is reflected by a change in surface's color. A normal map rendered in the global coordinate system will be observed with the same surface orientation vector from any viewpoint. For example, in the case of the left side of a cube, captured from two different cameras, if normal values are expressed in the *global* coordinate system surfaces orientation vectors have the same global direction and the observed surface color will remain the same. In contrast, if we are observing the same cube from a different perspective, but this time the normal values are expressed in *local* coordinate system, the orientation of the surfaces vectors, and subsequently the color appearance of the object, will differ between each local system. Given two local normal maps of an object, obtained from two cameras with a different orientation and local coordinate system, we will define the relative rotation and translation between Camera 1 and global coordinate system as R_1, t_1, while the relative rotation and translation for Camera 2 and the global coordinate system will be denoted as R_2 and t_2. The transformation that allows us to obtain global coordinate system values for each pixel is:

$$\mathbf{N}_{\text{GLB},i}(u,v) = R_i \cdot \mathbf{N}_{\text{LCL},i}(u,v) + T_i \tag{2.3}$$

Where:

- $\mathbf{N}_{\text{GLB},i}$ and $\mathbf{N}_{\text{LCL},i}$ are normal images rendered from the i-th camera with shape (H,W,3).
- R_i is the rotation matrix for the i-th camera.
- T_i: is the translation term for the i-th camera.

The inverse process, transforming a global normal map into local one, can be applied in the following way:

$$\mathbf{N}_{\text{LCL},i}(u,v) = R_i^T(\mathbf{N}_{\text{GLB},i}(u,v) - T_i) \tag{2.4}$$

Figure 2.2: First view of a global normal map

Figure 2.3: Second view of a global normal map

Figure 2.4: First view of a local normal map with identity rotation with respect to the world coordinate system

Figure 2.5: Second view of a local normal map, the color values are not completely coherent with the first view due to the application of camera rotation.

2.4 Point Clouds and Point Maps

A *point cloud* is a collection of points in a three-dimensional space. Each point is defined by its 3D coordinates (x, y, z), which specify its precise position in space. Point clouds have an unstructured nature: the points are not arranged in a regular grid or connected in any predefined way. This lack of structure allows point clouds to capture detailed geometric information, such as surface boundaries and fine features. The drawback is that processing a point cloud can be a complex and more transformation needs to be applied if a structured representation is needed. Furthermore, each point in the cloud may carry additional attributes like color, intensity or surface normal vector direction values. In contrast to point clouds, a *point map* is a structured representation in which the 3D points are organized in correspondence with the pixels of a 2D image. Each pixel in the *point map* is directly associated with a specific 3D point, preserving the spatial layout of the original image while keeping depth information. This structured mapping offers the advantage of a direct *Pixel To 3D point* association.

2.5 3D Unprojection

In computer vision, *3D unprojection* refers to the process of converting 2D image coordinates back into 3D world coordinates using depth information. Given a depth map $\mathbf{D}$ obtained from a camera with known intrinsic matrix, rotation matrix and translation vector, $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{T}$, for each pixel (u, v) the corresponding 3D point in camera coordinates is computed as:

$$\mathbf{p}_{\text{camera}} = D(u, v) \cdot \mathbf{K}^{-1} \cdot [u, v, 1]^T \tag{2.5}$$

where $D(u, v)$ is the depth value at that pixel location. The point is then transformed into world coordinates by means of the following transformation:

$$\mathbf{p}_{\text{world}} = \mathbf{R}^T(\mathbf{p}_{\text{camera}} - \mathbf{T}) \tag{2.6}$$

Depth values at infinity are discarded, and the resulting 3D points are stored in a structured *point map* that preserves spatial relationships. To obtain a point cloud from it we can just simply discard the structure of the point map object, flattening it in a 2D vector, where the first dimension represent the number of points in the cloud and the second dimension contains the X,Y and Z coordinates of each point.

2.6 Iterative Closest Point

Given a predicted and target point cloud, it is possible to employ the *Iterative Closest Point (ICP)* algorithm [17] to align them. ICP is a method for rigidly aligning two point clouds by iteratively minimizing the distance between corresponding points. Being $\mathcal{P}$ the predicted point cloud and $\mathcal{Q}$ the ground truth one, the following steps are applied:

- *Correspondence matching*: Find the closest points between $\mathcal{P}$ and $\mathcal{Q}$ using a nearest-neighbor search.
- *Transformation estimation*: Compute the optimal rigid transformation (rotation and translation) that minimizes the error between matched points.
- *Update*: Apply the transformation to $\mathcal{P}$ and iterate until convergence.

ICP stops iterating when one of the following conditions is met:

- The change in alignment error falls below a predefined threshold.
- The maximum number of iterations is reached.

Since point clouds generated from a 3D unprojection step may exhibit small spatial variations, ICP ensures that comparisons between predicted and ground truth point clouds are performed in a consistent reference frame.

This alignment step is crucial for obtaining meaningful reconstruction accuracy values, especially in cases where the compared point clouds are obtained through different procedures. For example, in the case of *DUSt3R*, the output point maps can be transformed into a point cloud but may be generated in an arbitrary coordinate system. Since the architecture does not receive any information about the input coordinate system, ICP is necessary to align the predicted point cloud with the reference ground truth ones.

2.7 Hemispherical sampling

Hemispherical sampling [2] is a technique used in computer graphics to distribute points uniformly over a hemisphere. In this case, it is applied to determine camera positions, in each view and scene, for ensuring uniform coverage across the upper hemisphere. The polar angle θ is determined using the inverse cosine of the square root of a uniform random variable r, ensuring a cosine-weighted distribution:

$$\theta = \arccos(\sqrt{r}), \quad \text{where } r \sim U(0,1) \tag{2.7}$$

The azimuthal angle ϕ is sampled uniformly from the interval $[0, 2\pi]$:

$$\phi = U(0, 2\pi) \tag{2.8}$$

The sampled angles are then converted into Cartesian coordinates to determine the camera position. Given a camera perspective distance d, the coordinates are computed as follows:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = d \begin{bmatrix} \cos(\theta)\sin(\phi) \\ \cos(\theta)\cos(\phi) \\ \sin(\theta) \end{bmatrix} \tag{2.9}$$

This approach ensures that the camera positions are distributed uniformly over the upper hemisphere.

2.8 Euler angles

Euler angles [16] are a set of three parameters used to describe the orientation of a rigid body in three-dimensional space. These angles represent a sequence of rotations around the principal axes of a coordinate system. The convention involves three consecutive rotations:

- *Yaw*: rotation around the vertical axis,
- *Pitch:* rotation around the horizontal axis,
- *Roll*: rotation around the axis perpendicular to the other two.

Chapter 3

Methodology

3.1 DUSt3R Architecture

DUSt3R (Dense and Unconstrained Stereo 3D Reconstruction) is an innovative framework designed to reconstruct 3D models from arbitrary image collections without requiring prior information about camera calibration or viewpoint positions. Traditional MVS methods must rely on known camera parameters to triangulate corresponding pixels in 3D space while *DUSt3R* operates directly on image content, estimating image depths and viewpoint poses to generate complete 3D reconstructions in the first view coordinate system.

3.1.1 Overall Design

Before feeding the data to the network, it is first passed through a pre-processing pipeline. All the data files are transformed into the correct format. Each data sample is structured in a main folder composed of:

- *Images subfolder*: contains camera poses, the intrinsic matrix, as well as images.
- *Depths subfolder*: with depth maps for each view.
- *Masks subfolder*: with masking information for each view.

During this step data augmentation techniques are applied , such as randomized change in resolution and RGB values. If necessary, images are resized or cropped to match the expected resolution, ensuring uniformity across training samples. Once preprocessed, the data is passed through a Transformer-based feature extractor, leveraging *CrocoNet* [14]. The input is first encoded and then processed by the decoder. Once the output is extracted from the decoder, it is fed into a regression head that functions as a downstream model. There is the possibility to use already implemented heads, namely:

- *Dense Prediction Transformer (DPT) [9]*: a transformer-based head optimized for dense output predictions.
- *Linear*: a simpler, fully connected head used for direct point cloud predictions.

These heads are directly trained to output a point map that represents the scene. The *Linear* head is simpler and allows the user to reduce memory consumption due to the lower number of parameters, while the *DPT* head is more powerful but much more computationally expensive. A series of post-processing steps and normalization approaches can be performed on the output. For a complete list of the features, refer to the *DUSt3R* paper.

3.1.2 New Input Data

Unlike the standard *DUSt3R* framework, which receive RGB images as input, the version presented in this work receive instead normal maps. The dataset structure remains the same as in the original pipeline, except that normal maps replace RGB images in the images folder, and for each training sample we feed two views into the network. This means that we have two normal maps, two depth maps, two camera information files, and two masks. The first view is both a local and global normal as it was rendered with a camera aligned with the global coordinate system. In contrast, the second view normal map

is purely local. During inference, the network predicts depth and structure information in the first camera coordinate system, aligning the output point clouds accordingly. Additionally, the network incorporates a global alignment strategy to aggregate multiple output point maps into a common reference frame. This allows us to train *DUSt3R* using only two views while enabling it to process an arbitrary number of views during inference. This alignment procedure facilitates the integration of multiple images into a unified 3D model while also recovering pixel correspondences and camera poses.

3.1.3 Training Objective

The training objective is based on a 3D regression loss as well as a confidence-weighted loss, which is computed to refine predictions. The loss function is designed to enforce accurate 3D reconstruction and is expressed as:

$$\mathcal{L}_{\text{Regr3D}} = \|\mathbf{P}_{\text{pred}} - \mathbf{P}_{\text{gt}}\|_2 \tag{3.1}$$

where $\mathbf{P}_{\text{pred}}$ and $\mathbf{P}_{\text{gt}}$ represent the predicted and ground-truth 3D point maps in the first view camera reference system, and the aim is to minimize the Euclidean distance between them. The Confidence-Weighted Loss ($\mathcal{L}_\gamma$), instead, allows the model to place more importance on predictions with higher confidence and less on those with lower confidence, helping it to focus on more reliable areas of the input images. The actual implementation introduces a confidence-based weighting mechanism that adjusts the contributions to the total loss:

$$\mathcal{L}_\gamma = \mathcal{L}_{\text{Regr3D}} \cdot \gamma - \alpha \log(\gamma) \tag{3.2}$$

where:

- $\mathcal{L}_{\text{Regr3D}}$ is the per-pixel regression loss.
- γ represents the confidence map, where there is a value associated at

each pixel.

- α is a hyperparameter that controls confidence weighting.

The total loss is obtained as the sum of each individual confidence loss for each input view:

$$\mathcal{L}_{conf} = \mathcal{L}_{\gamma,1} + \mathcal{L}_{\gamma,2} \tag{3.3}$$

During training, the model also shows validation loss values, this is fundamental for understanding how the model is performing on an unseen set of data. For doing this, considering the raw output prediction of the network is unfeasible, because the scale and coordinate system are completely randomized by the *DUSt3R* inner processes. The authors address this problem by introducing validation loss invariant to shifts and scale. The *Regr3D_ShiftInv*, based on the 3D regression loss, removes sensitivity to depth shifts by normalizing point depths relative to the median scene depth. Each point maps is normalized as follows:

$$\mathbf{P}_{\text{normalized}} = (\mathbf{P}_x, \mathbf{P}_y, \mathbf{P}_z - \text{median}(\mathbf{P}_z)) \tag{3.4}$$

where:

- $\mathbf{P}$ represents the point map.
- $\mathbf{P}_z$ is the depth (or z-coordinate) of the point map.

The *Regr3D_ScaleInv* loss, instead, normalizes point maps to a common scale to prevent scale dependency:

$$\mathbf{P}_{\text{normalized}} = \frac{\mathbf{P}}{\text{scale}(\mathbf{P})} \tag{3.5}$$

where:

- $\text{scale}(\mathbf{P})$ calculates a scale factor for the 3D points.

These two losses are combined to obtain a final one invariant to both shift and scale, allowing for a fair comparison during validation.

3.2 MoGe

MoGe [12] is a method for estimating 3D geometry from a single image. Unlike traditional approaches that first estimate depth and then infer camera intrinsics, *MoGe* directly predicts 3D point maps. *MoGe*'s training objective includes a normal reconstruction function component, which is applied to the output point map in order to obtain a predicted normal map. This is especially useful in the case of *DUSt3R*, as it allows leveraging the output point maps to reconstruct the input normals and assess accuracy. The process of computing normal vectors from the point map involves calculating local surface differences and aggregating them using cross-product operations. Given a point map P, the differences are computed for each pixel coordinate (u,v) as follows:

$$\text{up}(u, v) = P(u - 1, v) - P(u, v) \tag{3.6}$$

$$\text{left}(u, v) = P(u, v - 1) - P(u, v) \tag{3.7}$$

$$\text{down}(u, v) = P(u + 1, v) - P(u, v) \tag{3.8}$$

$$\text{right}(u, v) = P(u, v + 1) - P(u, v) \tag{3.9}$$

Four normal vector components are predicted for each (u, v) coordinate:

$$\Big[\text{up} \times \text{left}, \quad \text{left} \times \text{down}, \quad \text{down} \times \text{right}, \quad \text{right} \times \text{up}\Big] \tag{3.10}$$

The final direction at each pixel coordinate is obtained as the weighted sum of these four components, given an input validity mask already present in the training data. The resulting normals are constrained to the range $[-1, 1]$ by normalizing them. The computation of the positive direction follows the convention of the *OpenGL* [5] coordinate system.

Chapter 4

Experimental Result

4.1 Data

4.1.1 Data Generation Setup

To create a large-scale training dataset a custom Blender-based data generation pipeline was used. The script was configured to produce around 50 000 image-geometry pairs, ensuring a diverse set of samples for robust model training. The rendering pipeline was set up to leverage GPU acceleration for efficient processing. Specifically, a *NVIDIA GeForce RTX 4090 GPU* was used for generating the data. To ensure realistic and diverse object rendering, high-quality 3D models, materials, and environment maps were sourced from the *Adobe Stock 3D asset library* [1]. The script dynamically loads these assets from the source directory, retrieving random objects, assigning a material to them as well as applying a random light source to the scene. Each generated scene is composed of multiple randomized objects with varying positions, orientations, and materials. For a complete overview of the generation parameters refer to **appendix A**. As explained in **Section 2.7**, the camera positions for rendering each view were obtained from the application of hemispherical sampling on the upper hemisphere. Using Blender, a constraint was applied to keep the camera facing the object in any position, ensuring that the object

was always observed. In the figure 4.1, the Euler rotation angles associated with each position can be observed. The skewing toward the center in all three distributions is due to the fact that the first camera was always positioned on the Z-axis, looking towards the global origin, resulting in a zero rotation for all three pitch, yaw, and roll angles in half of the total camera poses. For each scene only two views are generated, meaning two camera angles per sample.

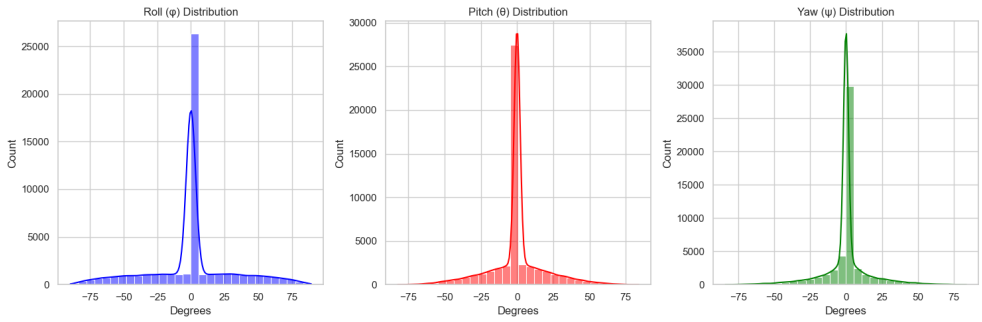

Figure 4.1: Distribution of rotation for Pitch, Yaw and Roll angles

4.1.2 Structure of generated data

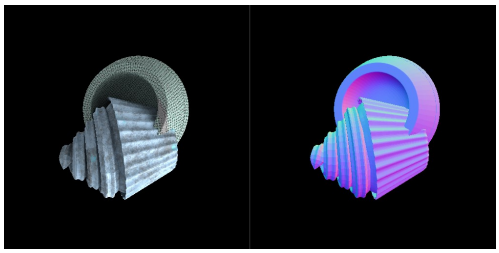

Figure 4.2: Random sample first view RGB and normal image

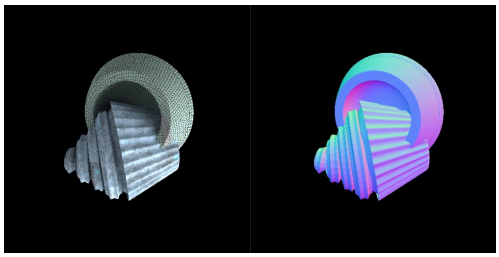

Figure 4.3: Random sample second view RGB and normal image

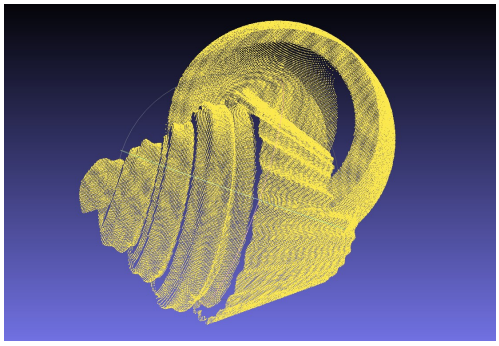

Figure 4.4: Merged ground truth point clouds of the scene

The generated data consists of rendered images, depth maps, and corresponding camera metadata, in particular we have:

- **First normal**: represents the normal map obtained from the first view of the scene. The first view is captured by a camera in Blender, perfectly aligned with the global coordinate system. Since the camera's local reference system matches the world reference system, the normal map values are effective both global and local.
- **Second normal**: represents a normal map captured from a second camera at a different angle. The values in this normal map are oriented with respect to the world reference system, which this time is not aligned with the camera local one, making the values only global.
- **First perspective projection matrix** : contains the rotation matrix, translation vector and intrinsic matrix. Since the first view camera is aligned with the world reference system, this rotation matrix is simply the identity matrix.
- **Second perspective projection matrix** : contains the rotation matrix, translation vector and intrinsics matrix for the second view.
- **RGB image of the first view**
- **RGB image of the second view**

- **Depth map of the first view**
- **Depth map of the second view**

4.1.3 Pre-processing

Three main pre-processing steps were applied to the generated data. The first step involved converting the second view from global to local coordinates using the camera information obtained from Blender, as described in **Section 2.3**. The second step involved applying the procedure outlined in **Section 3.1.1**, with some minor modifications. *Augmentation cropping* was deactivated as it introduced aliasing artifacts in the normal map. Additionally, the *color jittering* process was disabled. While this augmentation technique improves generalization for RGB images, it is less suitable for normal maps, where color encodes crucial geometric information. Altering these values could introduce inconsistencies during training, potentially impacting model performance. The third and final adjustment was reducing the precision of stored normal maps, changing from 16-bit to 8-bit precision. This modification was introduced for two main reasons:

- Lower precision reduces memory consumption and speeds up training time.
- Converting normal maps from TIFF to JPEG format was observed to introduce artifacts when using higher bit depths.

4.2 Training Objective

Two different loss functions were used during training. In the first case the confidence loss, as outlined in **Section 3.1.3**, is used. In the second case instead a combined loss is utilized: the idea is to apply a *MoGe*-inspired normal reconstruction loss to reconstruct a predicted normal map from the output

point map of *DUSt3R* as well as the confidence loss of the original training. The predicted normal map, obtained from the output point map of the model, is compared to the input normal map using the Euclidean distance between the corresponding normal vectors at each position. The reconstructed normal maps are predicted in global space and compared within the same reference system as the input. In the case of the first view, it can be both local and global, whereas in the second view, it is only local. Therefore, the comparison of the second predicted normal is performed after applying a change in the reference system from global to local. The comparison can be expressed as:

$$\mathcal{L}_{\text{Normal}} = \|\mathbf{N}_{\text{pred}} - \mathbf{N}_{\text{gt}}\|_2 \tag{4.1}$$

where:

- $\mathbf{N}_{\text{pred}}$ is the predicted normal map in the same reference frame as the input one.
- $\mathbf{N}_{\text{gt}}$ is the ground truth input normal map.

This process is repeated for both views, ensuring the normal map prediction is compared for each camera perspective. The final equation is:

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{Conf}} + \lambda \cdot \mathcal{L}_{\text{Normal1}} + \lambda \cdot \mathcal{L}_{\text{Normal2}} \tag{4.2}$$

where:

- $\mathcal{L}_{\text{Conf}}$ is the confidence loss, as defined earlier.
- $\mathcal{L}_{\text{Normal}}$ is the normal reconstruction loss, calculated as the Euclidean distance between the predicted and ground truth normal maps.
- λ is a hyperparameter that balances the relative importance of the two losses.

The implementation allows for flexibility in choosing whether to use only the confidence loss or the combined loss, as well as specifying the values of the weight parameters.

4.3 Evaluation Dataset

A suitable candidate dataset for the evaluation of our version of *DUSt3R* was identified as the *DiLiGenT-MV* dataset [6]. It extends the original single-view *DiLiGenT* [11] dataset to a multi-view setting, consisting of images of five objects with complex *Bidirectional Reflectance Distribution Functions (BRDFs)* [15], captured from 20 different viewpoints. The BRDF describes how light is reflected at a surface, defining the relationship between the incident and reflected light directions. It plays a crucial role in accurately modeling the appearance of real-world materials. For each view, 96 calibrated point light sources are used, ensuring controlled and consistent lighting conditions. Some example of samples taken from the the dataset can be observed from figure 4.5 to 4.8.

Figure 4.5: Ground truth normal for a random view of the bear object

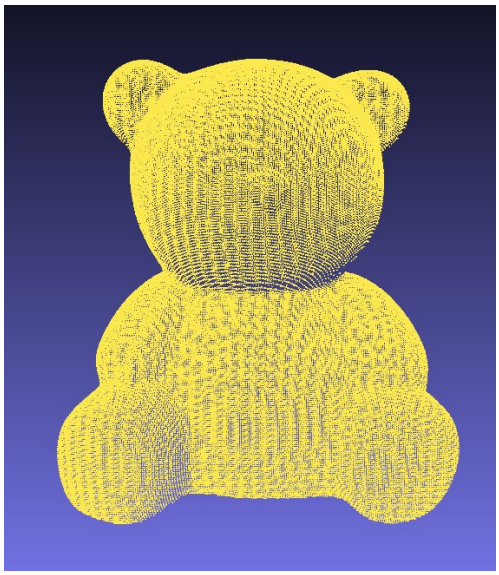

Figure 4.6: Ground truth point cloud for a random view of the bear object

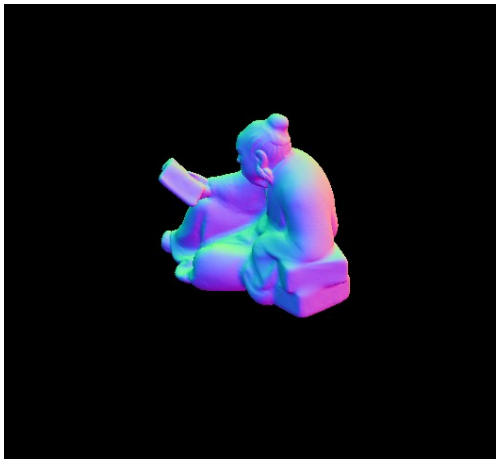

Figure 4.7: Ground truth point cloud for a random view of the reading object

Figure 4.8: Ground truth normal for a random view of the reading object

Additionally, the dataset includes ground truth shape information, allowing for a quantitative evaluation of 3D reconstruction accuracy. The decision to use this dataset was guided by the necessity of having ground truth for both normal maps and RGB images. This is because the baseline model, to which the final model will be compared, was trained with RGB images, making it unfair to feed it with normal maps directly. Furthermore, the dataset provides camera calibration and pose information. Depth maps are not directly given in the dataset, but they are essential for ground truth point cloud generation. To obtain the missing depth maps, the original *DiLiGenT-MV* setup was reconstructed in Blender, and depth images were rendered from each camera viewpoint for every object. Using these depth images, 3D unprojection procedure described in **Section 2.5** was applied to generate ground truth point clouds for each view.

4.3.1 Data Preparation

The custom evaluation dataset, based on *DiLiGenT* files has a specific structure. Each sample in the evaluation dataset have the following elements:

- Depth maps for each view
- GT point cloud (a combination of individual point clouds from both views)
- Normal maps for each view
- Foreground masks for each view
- RGB image for each view

For each scene, 8 samples are extracted from *DiLiGenT*. Each sample is obtained with a pair of views with a difference of at most two angles. This means that we can have view 1 and view 3 sample but not view 1 and view 4.

4.4 Evaluation Procedure

The evaluation procedure involves comparing the original *DUSt3R* model with a modified version that incorporates the changes described in the experimental setup. The two models are as follows:

- **Baseline**: the original *DUSt3R* model, pretrained using the final checkpoint from the official training. Specifically, the *ViTLarge_BaseDecoder_dpt* checkpoint from the official *DUSt3R* GitHub repository is used. This baseline model takes RGB images from the evaluation dataset as input.
- **Modified version (N-DUSt3R)**: a version of *DUSt3R* that includes the modifications described in the experimental setup section. Instead of RGB images, this version takes normal maps from each view as input.

Once the output point maps are generated by each model, two post-processing steps are applied. In the first step, the predicted point maps are converted into point clouds and normalized based on the scale factor difference relative to the ground truth. The scale factor is computed using an axis-aligned bounding box, which is obtained using the corresponding function from the Open3D [3] library. An axis-aligned bounding box is computed for each point cloud, defined as the smallest rectangular box, aligned with the coordinate axes, that fully encloses all the points in the cloud. The size of a bounding box refers to its largest dimension, which could be its width, height, or depth. By comparing the largest dimension of the ground truth bounding box with that of the predicted bounding box, a scale factor is determined. The predicted point cloud values are then scaled based on the computed scale factor. In the second step, the *ICP* algorithm, as described in **Section 2.6**, is applied to align the predicted point clouds with the ground truth.

The resulting point clouds are compared using *Chamfer Distance* [4] which is a commonly used metric for measuring the similarity between two point

clouds. Given two sets of points, P (predicted) and Q (ground truth), the Chamfer Distance is defined as:

$$d_{\text{Chamfer}}(P, Q) = \sum_{p \in P} \min_{q \in Q} \|p - q\|_2^2 + \sum_{q \in Q} \min_{p \in P} \|q - p\|_2^2 \tag{4.3}$$

This metric calculates the sum of squared distances from each point in one set to the nearest point in the other set, providing a bidirectional measure of similarity. A lower Chamfer Distance indicates a closer geometric match between the two point clouds.

4.5 Experiments and Results

A series of experiments were carried out to explore potential research directions. These research directions aimed to investigate the quality of the model output under different training conditions. They can be summarized as follows:

- **Over fitting analysis**
- **Analysis on resolution influence**
- **Analysis on loss function choice**
- **Final training**

For each training configuration refer to **appendix B**. The final training was conducted using four *NVidia H100* GPUs.

4.5.1 Overfitting Analysis

In this case, *DUSt3R* was trained and evaluated on a dataset containing only one sample. The primary goal of this experiment was to confirm that the modifications to the model, the data loader, and the input data were correctly implemented. If the model was unable to learn from even a single sample, it

would indicate an underlying issue in the training pipeline. The overfitting test was conducted on a simple cube, whose views can be observed in figure 4.9 and 4.10. Be noted that since the views don't cover the full shape, only the observed part can be reconstructed, leading to an incomplete cube. In figure 4.11 and 4.12, the reconstruction visual result can be observed, respectively from the *N-DUSt3R* model and baseline *DUSt3R*.

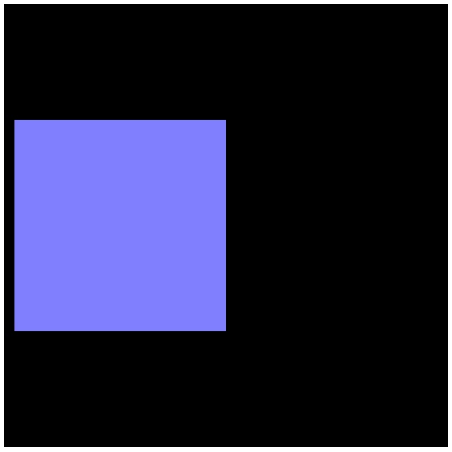

Figure 4.9: First view of the cube

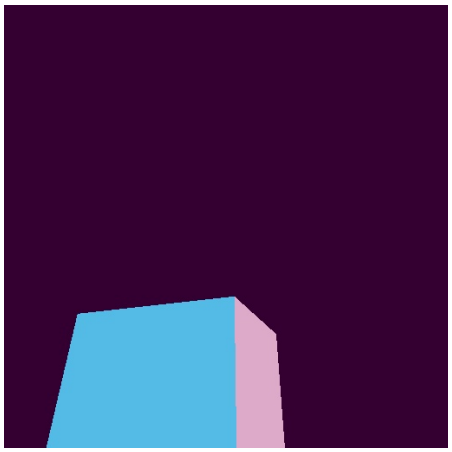

Figure 4.10: Second view of the cube

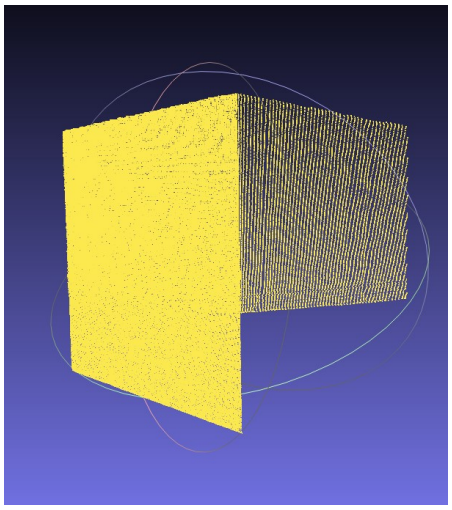

Figure 4.11: Reconstructed cube from overfitted model

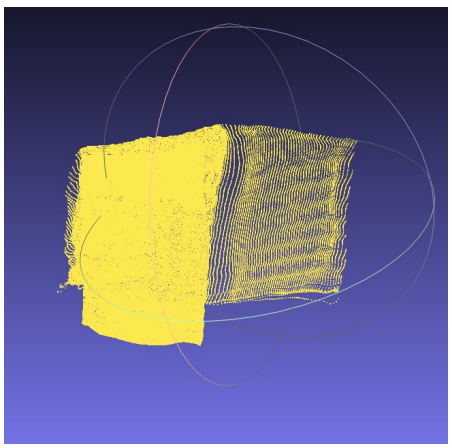

Figure 4.12: Reconstructed cube from baseline model

4.5.2 Resolution Influence

This series of training experiments revealed that the target resolution enforced by the data loader has a significant impact on the model's performance, even in the overfitting case. The target resolution should be divisible by 16, as *DUSt3R* processes images in 16×16 patches. In general, higher image resolutions lead to increased memory consumption, longer training times, and improved reconstruction accuracy. The objective was to determine the optimal balance between these factors, and the final choice led to a 512 resolution. The following images, from figure 4.13 to 4.15, illustrate the visual change in output's quality with respect to resolution.

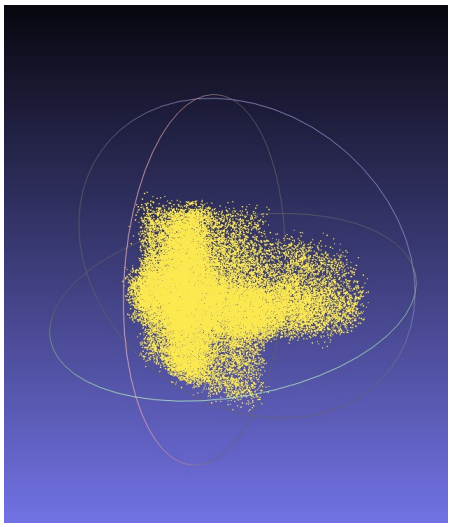

Figure 4.13: Output quality of overfitted model with 224 x 224 input image resolution

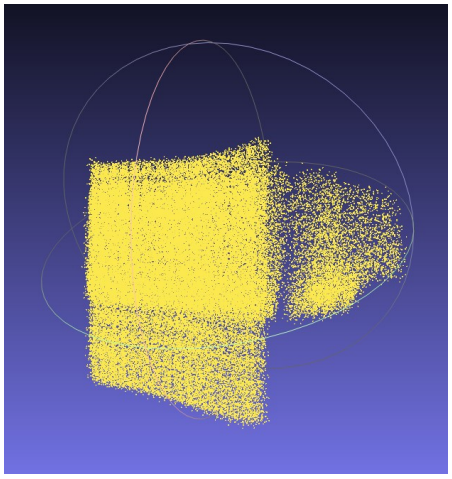

Figure 4.14: Output quality of overfitted model with 400 x 400 input image resolution

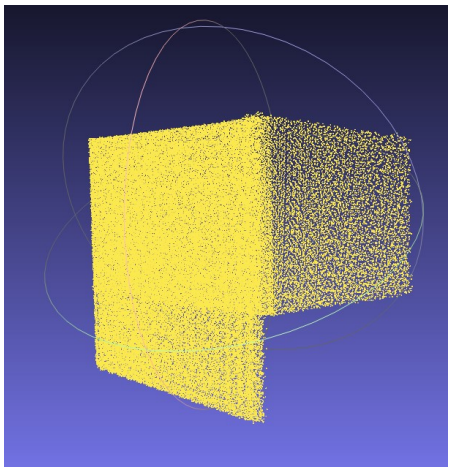

Figure 4.15: Output quality of overfitted model with 512 x 512 input image resolution

4.5.3 Loss Function Choice

As it can be observed from figure 4.16 and 4.17, the combined loss function convergence speed is around 4 times slower than the simple confidence loss. This was observed in every subsequent training and rendered more difficult the use of the combined loss due to increased training time.

Figure 4.16: Iteration convergence speed in the case of simple confidence loss

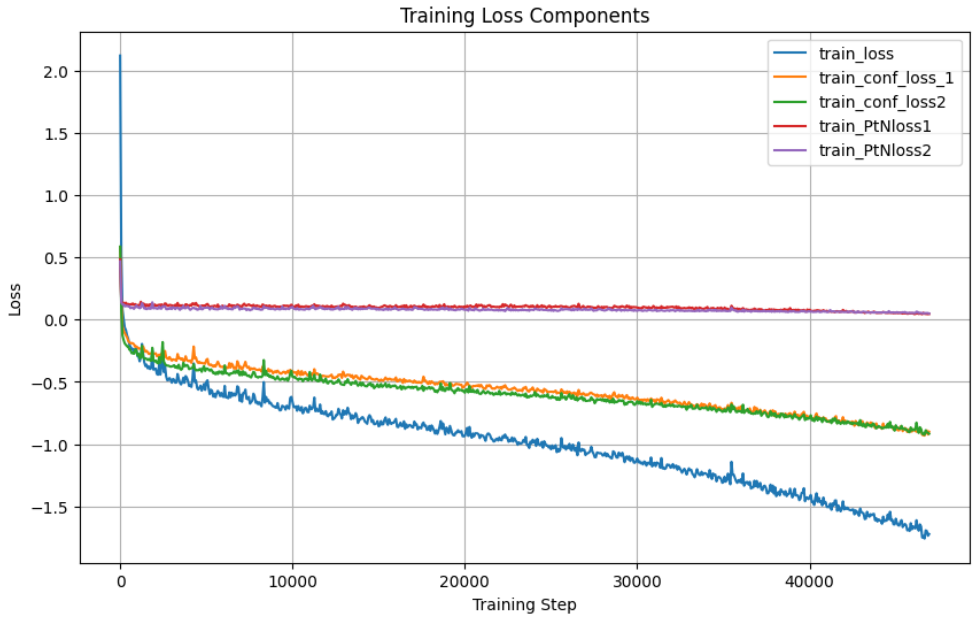

Figure 4.17: Iteration convergence speed in the case of combined normal loss

However, another issue emerged during the final testing phase of the model. From 4.17 it can be noted that the normal loss decrease is extremely slow and noisy: even if training appeared successful, during debugging of the loss function, a subtle inaccuracy was observed in the reconstruction of normal maps from point maps. Ideally, using ground truth point maps should produce a reconstruction nearly identical to the input image. However, when analyzing the disparity between the predicted normals and the GT normals, it became evident that the R and G channels behaved as expected, while the B channel exhibited an unexpected disparity that could not be attributed to simple numerical inaccuracies. An example of this behavior can be observed in figure 4.18, 4.19 and 4.20 The unclear effect of this phenomenon to the reconstruction result led to abandon the use of the combined loss for the final training.

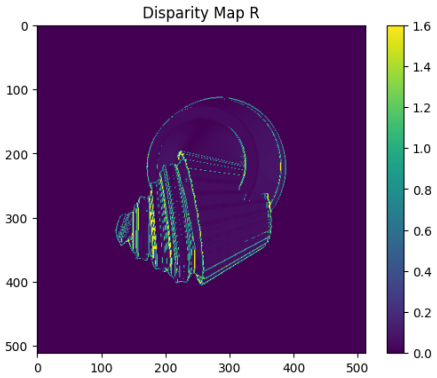

Figure 4.18: R channel disparity

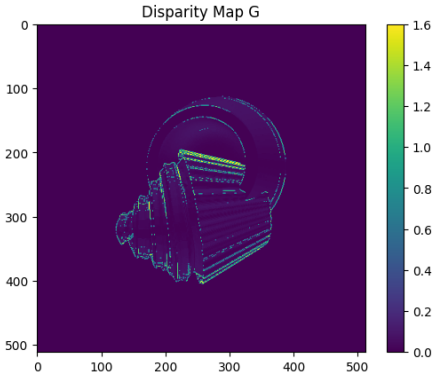

Figure 4.19: G channel disparity

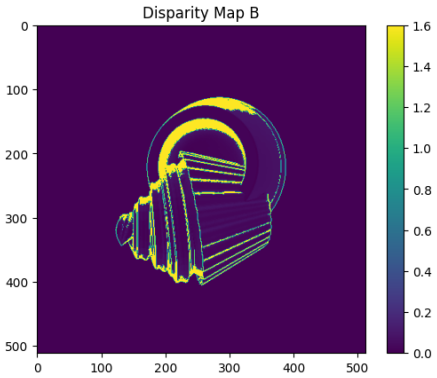

Figure 4.20: B channel disparity

4.6 Final Training

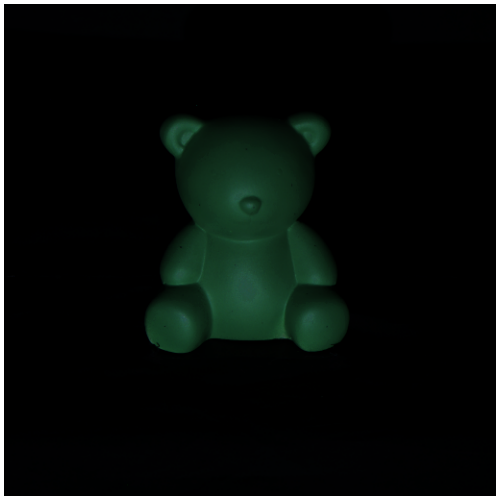

Figure 4.21: Bear

Figure 4.22: Cow

Figure 4.23: Reading

Figure 4.24: Pot

Figure 4.25: Buddha

In this section, the results of the final training are presented. For the testing images, refer to figures from 4.21 top 4.25. The *baseline* is a challenging architecture to surpass, however *N-DUSt3R* achieves comparable performance in most cases. Specifically, the baseline excels when handling complex shape objects because it can incorporate prior knowledge derived from the extensive variety of data it has seen during training. When the object features numerous spikes and intricate details, the information content increases, allowing the baseline to effectively reconstruct most views with high quality. However, this advantage diminishes when applied to particularly smooth objects, such as the cow model. The cow object is predominantly smooth, with minimal details other than the head and upper back. As a result, the baseline output quality is reduced when this informative details are missing. In contrast, *N-DUSt3R* excels in this scenario, as it can leverage on the information content of normal maps, which is significant even in the case of smooth surface because of color encoding. By observing table 4.3 it can be seen that *N-DUSt3R* beats the baseline in any situation. This can be clearly observed visually in image 4.26, 4.27 and 4.28.

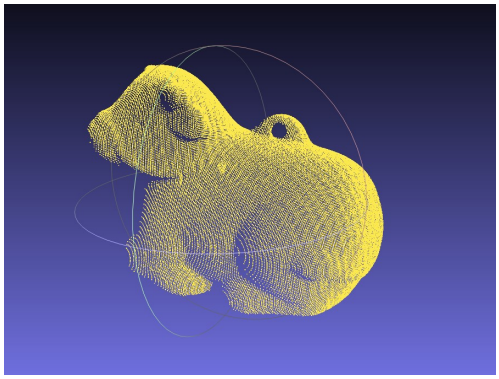

Figure 4.26: Cow object ground truth for sample 2

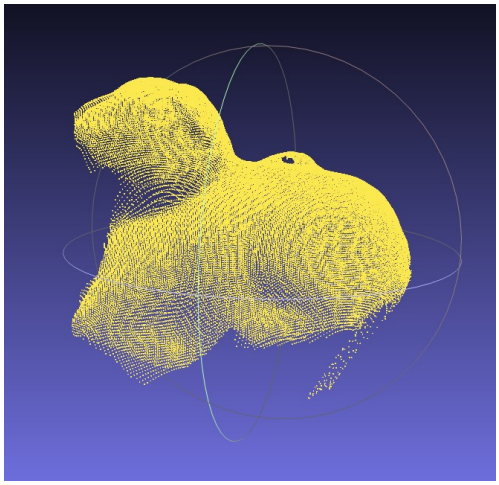

Figure 4.27: Baseline model prediction

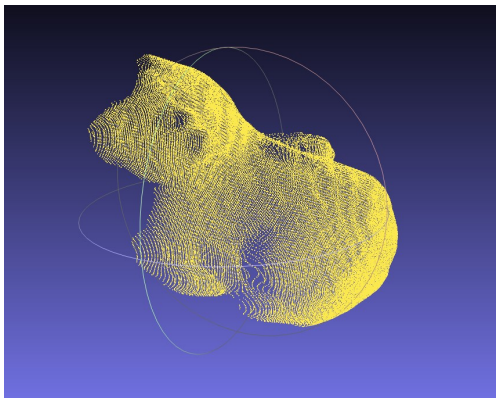

Figure 4.28: N-DUSt3R model prediction

It must be noted that *N-DUSt3R* was trained with synthetic data, and was evaluated on real data making it clear that the training data was diverse enough to make the model generalize. Additionally the model sacrifices some of the encapsulated knowledge from RGB images, from which instead the baseline benefits from, in order to learn new reconstruction patterns. Another important observation is that the level of detail in the reconstruction is lower in the baseline, even when the Chamfer distance is smaller than that of *N-DUSt3R*. This is particularly evident in the case of Buddha, a highly complex object. By examining the values in Sample 2 of the *Buddha* object table, we notice that in theory the baseline appears to perform much better than *N-DUSt3R*, since it has a lower Chamfer distance value. This example highlights a key limitation of models trained on RGB input images, as they often struggle to capture fine details effectively. Furthermore, it demonstrates that relying solely on quantitative metrics such as Chamfer distance is insufficient. Visual inspection is also essential for accurately assessing performance. The results can be observed in image 4.29, 4.30 and 4.31.

Figure 4.29: Buddha object ground truth for sample 2

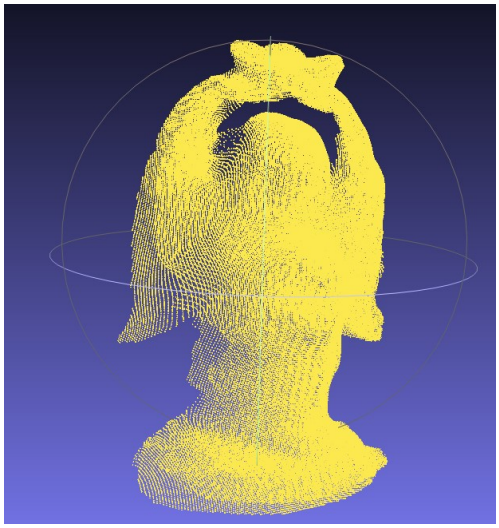

Figure 4.30: Baseline model prediction

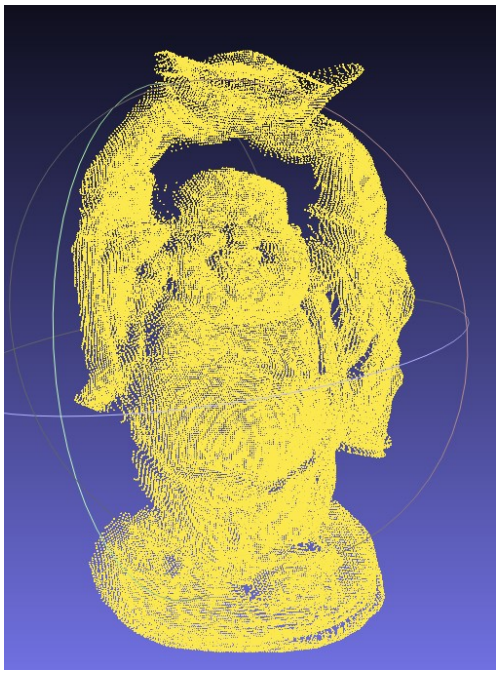

Figure 4.31: N-DUSt3R model prediction

The above behavior is repeated for any detailed view, such as in the ones related to the *reading* object. The performance of both the *baseline* and *N-DUSt3R* model is reported for each scene and each sample in tables 4.1 to 4.5. A star symbol next to a Chamfer distance value indicates that the predicted reconstruction quality was already quite low, making it difficult for the ICP algorithm to align it with the ground truth, further increasing the real Chamfer distance with alignment error.

Exhibit	Baseline	N-DUSt3R
1	**0.0451**	0.0679
2	**0.0858**	0.1028
3	**0.0824**	0.1149
4	0.0961	**0.0890**
5	0.0917	**0.0751**
6	**0.0647**	0.0724
7	**0.0552**	0.0811
8	**0.0332**	0.0693

Table 4.1: Chamfer distance comparison for Bear object

Exhibit	Baseline	N-DUSt3R
1	**0.0324**	0.1042*
2	**0.0303**	0.0621
3	**0.0398**	0.0715
4	**0.0299**	0.0578
5	0.1130	**0.1077**
6	**0.0486**	0.0685
7	**0.0529**	0.0565
8	0.0452	**0.0435**

Table 4.2: Chamfer distance comparison for Buddha object

Exhibit	Baseline	N-DUSt3R
1	0.0569	**0.0164**
2	0.0287	**0.0167**
3	0.0362	**0.0162**
4	0.0656	**0.0251**
5	0.0340	**0.0234**
6	0.0849	**0.0709**
7	0.0470	**0.0451**
8	0.0249	**0.0185**

Table 4.3: Chamfer distance comparison for Cow object

Exhibit	Baseline	N-DUSt3R
1	**0.0275**	0.0339
2	0.1875	**0.0637**
3	**0.0534**	0.0888
4	0.1091*	**0.0870**
5	**0.0573**	0.0928
6	**0.0758**	0.1127*
7	**0.0628**	0.2089*
8	**0.0367**	0.1178*

Table 4.4: Chamfer distance comparison for Pot2 object

Exhibit	Baseline	N-DUSt3R
1	0.1773*	**0.0577**
2	**0.1078**	0.1500
3	**0.0503**	0.1265*
4	0.0441	**0.0283**
5	0.0391	**0.0275**
6	0.1452*	**0.0377**
7	0.1653*	**0.0477**
8	**0.0839**	0.0859

Table 4.5: Chamfer distance comparison for Reading object

Chapter 5

Conclusion

As expected, a versatile architecture like *DUSt3R* successfully integrated normal map information from the newly introduced synthetic dataset. This outcome was anticipated, given that *DUSt3R* model size and training procedure are well-documented and have not exhibited any fundamental flaws. The results highlight a strong baseline performance that remains challenging to surpass with the modified model. However, the new model still achieves competitive results, demonstrating its potential. There remains significant room for future research and improvements. One key observation during training was a slight residual shift error in the normal reconstruction loss. This issue led to the eventual abandonment of the combined loss function, as its effects on training remained unclear. Addressing this problem in future work could allow for the correct reintegration of the combined loss function into the *DUSt3R* pipeline, potentially enhancing performance. Another avenue for improvement involves expanding the training dataset. It is uncertain whether integrating larger datasets, similar to the original *DUSt3R* training set, would yield improved results. However, with proper preprocessing of these datasets, it may be possible to extract the necessary normal information and enhance the model's generalization capabilities. Another possibility would be to use a joint representation of RGB images and normal maps as input to the model, enabling it to learn representations for both data modalities' informational

content simultaneously. Additionally, architectural modifications to *DUSt3R* could be explored, leveraging advancements in deep learning architectures that emerge each year. However, such modifications present challenges, as their impact on training dynamics and overall performance would be uncertain and require extensive experimentation.

Appendix A

Rendering Parameters

- **Number of views:** 2 per scene
- **Number of objects:** 2 per scene
- **Lens focal length range:** 20mm - 60mm (randomized)
- **Shading mode:** flat shading
- **Rendering device:** NVIDIA GeForce RTX 4090
- **Rendering engine:** Cycles
- **Image resolution:** 512x512 pixels
- **Scene lighting:** environment-based illumination using HDRI maps
- **Object rotation:** random for each local axis
- **Measurement unit:** Metric
- **Resolution:** (512, 512)
- **Frame rate:** 24
- **Start frame:** 1
- **End frame:** 250

- **Camera type:** Perspective
- **Focal length:** 25.0
- **Sensor size:** 35.0
- **Field of view:** 1.2214518785476685
- **World background color:** Color (r=0.0500, g=0.0500, b=0.0500)
- **Output normal format:** TIFF
- **Output depth format:** exr
- **Light count:** 1
- **Material count:** 2

Appendix B

Training Configurations

B.1 Baseline DUSt3R

For the baseline, the official DUSt3R training hyperparameters and the available DPT checkpoint was used.

B.2 Overfitting DUSt3R with Confidence Loss

The training with confidence loss was performed using the following configuration:

- **Model architecture**: `AsymmetricCroCo3DStereo`
- **Training criterion**: `ConfLoss(Regr3D(L21, norm_mode='avg_dis'), alpha=0.2)`
- **Testing criterion**: `Regr3D_ScaleShiftInv(L21, gt_scale=True)`
- **Pretrained model**: CroCo_V2_ViTLarge_BaseDecoder.pth
- **Learning rate (lr)**: `0.0001`
- **Minimum learning rate (min_lr)**: `1e-06`
- **Warmup epochs**: `0`

- **Total epochs**: `40`
- **Batch size**: `1`
- **Accumulation iterations (accum_iter)**: `1`

B.3 Overfitting DUSt3R with Combined Loss

The training with combined loss was performed using the following configuration:

- **Model architecture**: `AsymmetricCroCo3DStereo`
- **Training criterion**: `Normal_loss(ConfLoss(Regr3D(L21, norm_mode='avg_dis'), alpha=0.2), PtNloss(L21), beta=0.7)`
- **Testing criterion**: `Regr3D_ScaleShiftInv(L21, gt_scale=True)`
- **Pretrained model**: CroCo_V2_ViTLarge_BaseDecoder.pth
- **Learning rate (lr)**: `0.0001`
- **Minimum learning rate (min_lr)**: `1e-06`
- **Warmup epochs**: `0`
- **Total epochs**: `40`
- **Batch size**: `1`
- **Accumulation iterations (accum_iter)**: `1`

B.4 Final Configuration

For the final run, based on insights from previous trials, the parameters were slightly adjusted to achieve better performance. The final configuration is as follows:

- **Model architecture**: `AsymmetricCroCo3DStereo`
- **Training criterion**: `ConfLoss(Regr3D(L21, norm_mode='avg_dis'), alpha=0.2)`
- **Testing criterion**: `Regr3D_ScaleShiftInv(L21, gt_scale=True)`
- **Pretrained model**: DUSt3R_ViTLarge_BaseDecoder_512_dpt.pth
- **Learning rate (lr)**: `0.0001`
- **Minimum learning rate (min_lr)**: `1e-06`
- **Warmup epochs**: `10`
- **Total epochs**: `1000`
- **Batch size**: `4`
- **Accumulation iterations (accum_iter)**: `8`

Bibliography

[1] Adobe. 3d assets | adobe stock. 2025. URL: https://stock.adobe.com/3d-assets.

[2] B. Ameye. Sampling the hemisphere. 2025. URL: https://ameye.dev/notes/sampling-the-hemisphere/.

[3] O. Contributors. Open3d: a modern library for 3d data processing. https://www.open3d.org/, 2021.

[4] F. Williams. Point Cloud Utils - Shape Metrics, 2025. URL: https://fwilliams.info/point-cloud-utils/sections/shape_metrics/.

[5] Khronos Group. The industry's foundation for high performance graphics. 2025. URL: https://www.opengl.org/.

[6] M. Li, Z. Zhou, Z. Wu, B. Shi, C. Diao, and P. Tan. Multi-view photometric stereo: a robust solution and benchmark dataset for spatially varying isotropic materials. *IEEE Transactions on Image Processing (TIP)*, 29(1):4159–4173, 2020. DOI: 10.1109/TIP.2020.2986725. URL: https://ieeexplore.ieee.org/document/9171987.

[7] Z. Li and N. Snavely. Megadepth: learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. URL: https://www.cs.cornell.edu/projects/megadepth/.

[8] Pokedstudio. Blender organization. 2025. URL: https://www.blender.org/.

[9] R. Ranftl, A. Bochkovskiy, and V. Koltun. Vision transformers for dense prediction. *arXiv preprint arXiv:2103.13413*, 2021. DOI: 10.48550/arXiv.2103.13413. URL: https://arxiv.org/abs/2103.13413.

[10] J. Reizenstein, R. Shapovalov, P. Henzler, L. Sbordone, P. Labatut, and D. Novotny. Common objects in 3d: large-scale learning and evaluation of real-life 3d category reconstruction. In *International Conference on Computer Vision (ICCV)*, 2021. URL: https://arxiv.org/abs/2109.00512.

[11] B. Shi, Z. Mo, Z. Wu, D. Duan, S.-K. Yeung, and P. Tan. A benchmark dataset and evaluation for non-lambertian and uncalibrated photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(2):271–284, 2019. DOI: 10.1109/TPAMI.2018.2848850. URL: https://ieeexplore.ieee.org/document/8433520.

[12] R. Wang, S. Xu, C. Dai, J. Xiang, Y. Deng, X. Tong, and J. Yang. Moge: unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision, 2024. DOI: 10.48550/arXiv.2410.19115. arXiv: 2410.19115v1 [cs.CV]. URL: https://arxiv.org/html/2410.19115v1. License: CC BY 4.0.

[13] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud. Dust3r: geometric 3d vision made easy, 2023. DOI: 10.48550/arXiv.2312.14132. arXiv: 2312.14132 [cs.CV]. URL: https://arxiv.org/abs/2312.14132.

[14] P. Weinzaepfel, V. Leroy, T. Lucas, R. Brégier, Y. Cabon, V. Arora, L. Antsfeld, B. Chidlovskii, G. Csurka, and J. Revaud. Croco: self-supervised pre-training for 3d vision tasks by cross-view completion,

2022. DOI: 10.48550/arXiv.2210.10716. arXiv: 2210.10716 [cs.CV]. URL: https://arxiv.org/abs/2210.10716.

[15] Wikipedia contributors. Bidirectional reflectance distribution function — Wikipedia, The Free Encyclopedia, 2025. URL: https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function. [Online; accessed March 16, 2025].

[16] Wikipedia contributors. Euler angles. 2025. URL: https://en.wikipedia.org/wiki/Euler_angles.

[17] Wikipedia contributors. Iterative Closest Point — Wikipedia, The Free Encyclopedia, 2025. URL: https://en.wikipedia.org/wiki/Iterative_closest_point.

Acknowledgements