Multispectral NeRF: modeling real scenes and synthetic remote sensing imagery

Arnau Marcos Almansa

February 8, 2024

Resum– Aquest projecte explora l'ús dels Neural Radiance Fields (NeRF) en el domini multiespectral. Construïm un model NeRF que admet imatges amb diverses bandes a part de RGB. Utilitzem una combinació d'imatges multiespectrals preses d'escenes reals i escenes sintètiques basades en dades de Sentinel-2. Les imatges sintètiques proporcionen un entorn controlat per entrenar i provar el model NeRF sense les particularitats de les imatges satèl·lit reals. El projecte també construeix el seu propi pipeline de renderització volumètrica i implementa xarxes neuronals per a la implementació de NeRF. Els resultats són comparables a la implementació original de NeRF. L'extensió al cas multiespectral ha estat directa, i es pot treballar amb totes les bandes amb un error baix. El projecte destaca el potencial que tenen aquestes tècniques en l'àmbit de la teledetecció.

Paraules clau– NeRF, imatge multispectral, imatge satelital, camps neuronals

Abstract– This project explores the use of Neural Radiance Fields (NeRF) in the multispectral domain. We build a NeRF model that supports multi-band images other than RGB. We use a combination of multispectral images taken from real scenes and synthetic scenes based on Sentinel-2 data. Synthetic imagery provides a controlled environment to train and test the NeRF model without the particularities of real satellite imagery. The project also builds its own volumetric rendering pipeline and implements neural networks for the NeRF implementation. The results are comparable to the original NeRF implementation. The extension to the multispectral case has been direct, and it is possible to work with all the bands with a low error. The project highlights the potential that these techniques have in the remote sensing field.

Keywords– NeRF, multispectral image, satelite image, neural fields

1 Introduction

GEospatial information is more relevant than ever in today's world. Modeling satellite and aerial images has become an essential activity for a wide set of applications, from predicting natural disasters to managing natural resources. Nowadays, many techniques exist for processing satellite and aerial images and being able to model the captured terrain. For instance, to obtain a Digital Surface Model (DSM) stereo products are often made, where two images are taken simultaneously and then processed to obtain the model.

Satellite images, in addition to being taken from points very far away from the ground, are also characterized by the fact that the satellites often carry special sensors, linear sensors that take advantage of the movement of the satellite itself and multispectral sensors that increase the information that can be obtained from their images. In this project, I will focus on this latter type of image, where multiple bands provide extra information contained in spectral variability.

On another note, in recent years, one of the techniques that has gained popularity when tasked with modeling a complex 3D scene is Neural Radiance Fields (NeRF) [7]. This technique allows the synthesizing of novel views of a scene and creates an implicit representation of said scene from multiview images. This technique has already been tested with satellite images, but it's still not widespread.

Some limitations still exist because of the fact of working with exclusively RGB images. For instance, when trying to measure crop health or soil moisture, bands other than RGB ones are more useful. Modeling the scene with images that

- E-mail de contacte: arnaumarcosalmansa@gmail.com, 1354223@uab.cat
- Menció realitzada: Computació
- Treball tutoritzat per: Felipe Lumbreras Ruiz (Dept. Ciències de la Computació)
- Curs 2023/24

contain more bands apart from the traditional RGB would provide a richer comprehension of the modeled scene.

Also, the current work being done in this field focuses on views with a high level of detail obtained from private images, while low-resolution images have not been given the same importance.

In this work, I will apply NeRF-type algorithms built from scratch, applied to multispectral images and free satellite images from Sentinel-2.

2 Objectives

The primary objective of this project is to develop a Neural Radiance Fields (NeRF) model with the capability to learn, and model terrains based on synthetic images. The objectives of this project encompass an exploration of the current state of the art in NeRF models applied to satellite images and applied to multispectral images. This exploration allows me to learn about the most recent techniques and practices in the field of NeRF.

This main objective of building a NeRF architecture for multispectral images will be achieved via a series of subobjectives. A considerable part of the project is the creation of the NeRF model. This creation is followed by validation and testing procedures using the original NeRF images. This phase is crucial for ensuring the model works, is accurate, and trains reliably. After this phase, the model must work and give results comparable to the original NeRF.

Later, the project delves into generating and using synthetic satellite and aerial images. Generating these images is an explicit goal of the project. These images provide a controlled environment to test the capabilities of the model.

Lastly, this project aims to expand the original NeRF model to work with multispectral images. This gives a richer comprehension of the modeled terrain.

3 Planning

From the objectives proposed in the previous section, I developed the plan for the project. The first part, and one of the most important, of the project is the exploration of the state of the art. This aims to provide me with a solid foundation and understanding of the technology and techniques I use in the rest of the project. Following this part, the plan includes a phase of gathering data needed to perform the basic NeRF experiments. Then the project continues with the largest phase, which is developing my own NeRF model. This phase includes developing the model and also validating and testing it. Then the project continues with me generating the synthetic data needed to perform the rest of the experiments. Finally, the project concludes with a phase consisting of expanding the NeRF model to support images with multiple bands. A version of this plan in a Gantt diagram was already handed over in a previous delivery and can be found in the dossier.

4 Methodology

The methodology for this project is based on the Kanban methodology. I create a list of all tasks needed for the completion of the project, store those tasks on a backlog, and perform the tasks in the order they are on the list.

I use Git through GitHub to have versioning of the code I develop. GitHub also allows me to work in parallel on different tasks when needed and enables me to work from different computers, e.g. from my given PC at CVC and from home. The Git repository is public [1] and is the dossier of this project.

Every week, from Monday to Thursday, I have a small meeting with my tutor to revise the progress of the project, solve doubts, and ask for help.

5 State of the art

As I mentioned in the introduction, this work expands the Neural Radiance Fields techniques to new modalities of images. First I begin with an overview of the original NeRF model: how it's structured, how it works, and some of its optimizations. Then I explain some applications of NeRF in two categories, the first one being NeRF applied to satellite imagery, and the second being NeRF applied to multispectral images.

5.1 NeRF

In recent years, a technique that has gained popularity to solve the problem of novel view synthesis of a scene is Neural Radiance Fields (NeRF). This technique consists of training a multilayered perceptron (MLP) from different images of the scene taken from known poses. From these different views, the MLP is capable of constructing an implicit 3D representation of the scene, and then we are able to query for the color in each point of the representation.

To achieve this, NeRF casts a ray for each pixel on the image. This ray has two *near* and *far* bounds, which define the distance at which the ray starts to cast and the distance at which the ray stops. This ray is sampled at different points along its length. To choose the sample points, it splits the ray into segments and chooses a random sampling point on each segment. That is to avoid overfitting the neural network to specific sampling points.

Each of these sample points is then passed to a multilayered perceptron (MLP) along with the view direction of the ray to which the point belongs. The inputs of the model end up being (x, y, z, d_1, d_2, d_3) where x, y, z represent the 3D point we want to query and d_1, d_2, d_3 represent the view direction. In Fig. 1 a simple representation of this can be seen.

These inputs are then passed via a positional encoder, which encodes them using $\sin$ and $\cos$ functions as in Eq. (1). Both sample position and view direction are encoded separately, the position with an $L = 10$ and the view direction with an $L = 4$. The reason for encoding both is that neural networks tend to be biased via low frequencies, and these encodings can help the network learn high-frequency details.

$$p(x) = (\sin(2^0\pi x), \cos(2^0\pi x) \ldots \sin(2^L\pi x), \cos(2^L\pi x)) \quad (1)$$

Once the inputs have been encoded, they are passed to the rest of the network to produce the output. This output,

[1] https://github.com/ArnauMarcosAlmansa/TFG

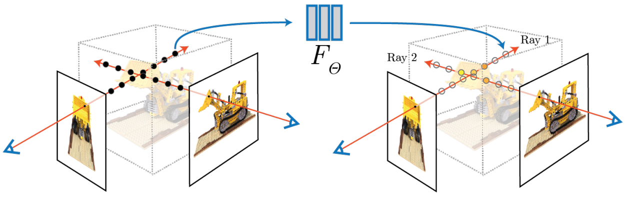


Fig. 1: Scheme of how the volume is sampled. For each pixel in the images, a ray is cast. For each ray, N sample points are selected that are passed to the MLP, producing the density and color of the point. *(source: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis).*

for each sampling point i of the ray, consists of an RGB color c_i and a density σ_i. These outputs are then integrated for each ray to obtain the final color of the pixel. The integration consists of the formulas on Eq.(2). This formula refers to the obtaining of the final color, $\hat{I}$ on the formula. The next formula, on Eq. (3), dictates how to compute the weight (aka importance) of the sampled point, that is, how much impact the sample has on the final output. The second formula allows computing the transmittance T_i of each sample, that is, the probability a ray of light coming from the sampling point would reach the camera without colliding first with another object and thus not reaching the camera. Finally, to obtain α_i, which is used to compute both the weight and the transmittance, it follows the last formula of (3), where σ is the density and d_i is the distance between sampling points.

$$\hat{I} = \sum_{i=1}^{N_s} w_i c_i, \tag{2}$$

where:

$$w_i = T_i \alpha_i, \;\; T_i = \prod_{j=0}^{i-1} (1 - \alpha_i), \;\; \alpha_i = 1 - e^{-\sigma_i d_i}. \tag{3}$$

5.2 NeRF applied to satellite images

NeRF techniques that began by being used on interior scenes or individual objects have been extended to a variety of situations, videos [9], outside scenes [6], a few training images [10], etc. In this work, I focus on satellite and multispectral images, where NeRF has begun generating good results.

Here I cite the main papers I have consulted and have served as inspiration for our specific implementation.

S-NeRF Shadow Neural Radiance Fields or S-NeRF [1] is an improvement over NeRF specifically designed to work on satellite images. What this model does is add another input that signifies the direction of the sun, and also adds two outputs that predict the RGB color of the sky based on the sun's direction and a weight that indicates the visibility of the sun at each sampling point. Thanks to these improvements, it is capable of more accurately predicting the albedo of the scene apart from the shadows cast by the sun and the ambient light.

Sat-NeRF Satellite Neural Radiance Field or Sat-NeRF [4] improves on previous attempts in two ways. First, it replaces the original pinhole camera model used by NeRF with a more sophisticated Rational Polynomial Camera (RPC) model, which more realistically represents a real satellite camera. Second, it adds extra inputs and an output to the network that indicates the probability that a sample point on the scene belongs to a transient object. This is because real satellite images often contain transient objects, such as cars or people, that appear or do not on different images. It borrows the latter improvement, to the transient objects, from NeRF in the Wild [6]. The extra inputs consist of a vector extracted from an embedding with a vector for each training image. This helps the network learn the changes in appearance that the scene displays for each different image. In the case of Sat-NeRF, these changes in appearance are caused by the transient objects.

EO-NeRF Earth Observation NeRF (Eo-NeRF) [5] is the most recent model. It integrates some of the steps that had to be performed on the images before using them to train inside the MLP, like correcting the camera model. But the biggest improvement that the system incorporates is that it is capable of generating the shadows from the geometry of the scene, instead of predicting them from the inputs of the network. It achieves this by casting rays from the points of the surface of the scene that is being rendered towards the sun. For each sun ray cast, it computes the transmittance of the surface point to the sun and multiplies it by the color of the surface point. That means that, if a surface point has a transmittance of 0 the sunlight does not reach it, and thus it is completely in the shadow.

5.3 Nerf applied to multispectral images

Originally, NeRF only works with RGB images, but it is interesting to add more spectrums to the images to gain a more complete understanding of the scene or of the total spectrum the camera is capable of capturing.

X-NeRF Cross-Spectral Neural Radiance Fields (X-NeRF) [8] is a system that manages to model a scene from images taken in various spectral bands. The biggest challenge this model had to overcome is that the various spectrums are not recorded using the same camera, but from different cameras attached to the same rig. Each camera has its own resolution and parameters. This makes it hard to robustly estimate the relative camera poses between the different spectrums and thus difficults the task of reconstructing the scene. COLMAP can only be used to estimate the relative positions of the same camera in different moments, but not the relative positions of cameras in different spectrums [8]. The solution they propose makes the neural network learn and estimate the relative poses of the cameras mounted on the rig.

Spec-NeRF Multi-spectral Neural Radiance Fields (Spec-NeRF) [3] is another model that allows working with different spectrums. In this case, instead of taking the images with different cameras, the images are taken with the same camera applying different filters to the lens. The camera poses can then be estimated using a single spectrum. This model not only learns the resulting RGB color directly, but it also learns a representation of the whole spectrum the camera is capable of capturing and uses that information to be able to reconstruct the final RGB values.

6 Development

In this section, I talk about the entire development of the project. The work done in this project consists of gathering the data needed to train and test the NeRF model, developing the NeRF model, training and testing it, generating the synthetic data for the experiments, and developing, training, and testing the expanded model that supports extra spectrums on the images.

6.1 Tools used

Here is a recollection of the tools I have used during this project. These have been essential and have helped me develop the entire project.

Python The main programming language I have used is Python. This is because it is a very commonly used language in machine learning, thanks to the availability of libraries like TensorFlow or PyTorch.

PyTorch This is the library I have used for creating the neural networks. The reason for using this library instead of Tensorflow is because it is the most popular alternative in the scientific community.

PyRender This library has allowed me to render the data created from the scenes I have built. It offers a complete Physical-Based Rendering (PBR) pipeline that allows rendering realistic scenes.

PyCharm The main IDE I have used for developing this project. As with all other IDEs, it allows me to view and edit the code and also comes with an integrated debugger, which has proved very useful during the development of the project.

Git and GitHub For version control, I have used Git and GitHub. This has allowed me to keep the code in sync while working from different PCs and to keep track of the changes I have made to the code.

LaTeX and Overleaf Lastly, I have used LaTeX and Overleaf to document the project and write the project monitoring reports and this final report.

6.2 Data gathering

The data I have gathered consists essentially of two sources:

- The original NeRF dataset.
- BigEarthNet Extended with Geographical and Environmental Data (BEN-GE).

Band	Description	Height and width (px)	Resolution (m/px)	λ (nm)
B01	Ultra Blue	20	60	443
B02	Blue	120	10	490
B03	Green	120	10	560
B04	Red	120	10	665
B05	Vis&NIR	60	20	705
B06	Vis&NIR	60	20	740
B07	Vis&NIR	60	20	783
B08	Vis&NIR	120	10	842
B8A	Vis&NIR	60	20	865
B09	SWIR	20	60	940
B11	SWIR	60	20	1610
B12	SWIR	60	20	2190

Table 1: Description of the Sentinel-2 image bands

The original NeRF dataset can be found on this repository[2]. I specifically have downloaded the `nerf_synthetic` data. As for BEN-GE, I have downloaded the Sentiel-2 patches that come with 12 bands of different resolutions. I have also downloaded the Digital Elevation Model (DEM) files that I will use as height maps when generating the synthetic scenes. I have downloaded the patches from the official website[3]. The DEM files come from this site[4].

As I have said, the downloaded Sentinel-2 patches consist of 12 images, one for each band. Each patch represents an area of 1.2 km². The color bands are bands B02, B03, and B04 (B, G and R respectively). Each of these three with a resolution of 10 m for each pixel. All bands, including B02, B03, and B04, are as specified in [2] and are described in Table 1.

The size of the DEM is also 120 × 120 px, with a resolution of 10 m per pixel.

6.3 Synthetic data generation

Most of the data I have used to train the models is synthetic. To generate the images, I have chosen a patch from BEN-GE to work with. To render the images, I have used the PyRender Python library. The images are rendered at a resolution of 800 × 800 pixels.

Obtaining the mesh To obtain the mesh, as in the geometry of the terrain, I have used the DEM as a height map. The DEM is quantized at a very high level, with gaps between neighboring pixels of 7 or 8. Because of that, to avoid very visible jumps in height when rendering, the DEM (Fig. 2) has to be preprocessed.

Originally, I upscaled the DEM to a size of 960×960 pixels, which is 8 times the original, and then applied a Gaussian blur with a kernel of 17×17 to smooth out the upscaled DEM. This reduced a little the visible jumps in height but did not reduce them greatly.

Later in the project, I changed the strategy to one that consists of progressively scaling the DEM and adding random noise. A more detailed explanation can be found in the

[2] https://drive.google.com/drive/folders/128yBriW1IG_3NJ5Rp7APSTZsJqdJdfc1

[3] https://bigearth.net/

[4] https://zenodo.org/records/8129350

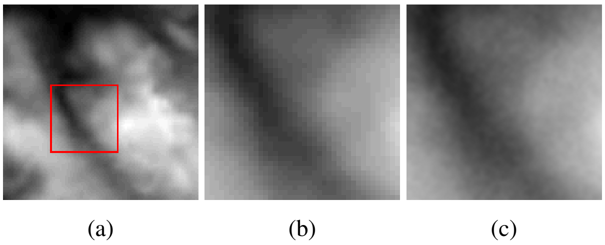


Fig. 2: (a): the original DEM obtained from the BEN-GE dataset, (b): a detail from the original DEM where the quantization can be appreciated in the jumps in values between some neighboring pixels, (c): a detail from the post-processed DEM, where the jumps between neighboring pixels can no longer be appreciated.

appendix. The result can be seen in Fig. 2(c). Once the DEM has been preprocessed, the program generates a set of 3D points, one for each pixel of the DEM. The x and y coordinates are based on the coordinates of the pixel on the DEM, they are normalized in the range -0.5 to 0.5. The z coordinate is based on the value of the pixel. After this, the program generates the triangles for the points and obtains the mesh that I will then texture and render. To do this, it iterates over the points in two passes: the first pass iterates over the points' rows and columns from the first one to the penultimate one, and builds the triangle using its $x + 1$ and $y + 1$ neighbors. then the second pass iterates over the rows and columns from the second to the last, and for each point generates the triangle using its $x - 1$ and $y - 1$ neighbors. This can be seen in Fig. 3. To generate the UVs, when generating the vertices, for each triangle vertex, it assigns a UV coordinate of the x coordinate divided by the width and the y coordinate divided by the height of the DEM.

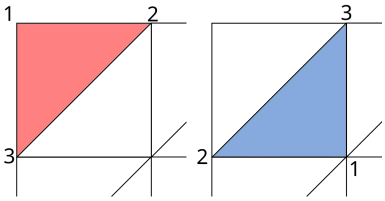


Fig. 3: The two passes the program makes to generate the triangles of a square. The numbers on the vertices indicate the order in which the point is considered for the triangle.

RGB texture To obtain an RGB texture, I fuse the three color bands (B02, B03, and B04, see Table 1) to generate the RGB texture in Fig. 4(a). I then apply the texture to color it. Once applied, I render the mesh and get the RGB images.

The result of the process is Fig. 4(b), where we can see the mesh generated from the DEM with the texture applied.

12 band texture To obtain the data for the experiments with more than the RGB bands, I change the strategy. Instead of combining the bands in RGB, I render the bands individually. I do this by triplicating each band, generating an RGB texture for each band where R, G, and B are the same, thus generating a grayscale texture. I then render the scene 12 times for each viewing position, one render for each band, and store the resulting images as grayscale. Beware that this is only to generate the data that will be used in the experiments, and is not how the NeRF model operates.

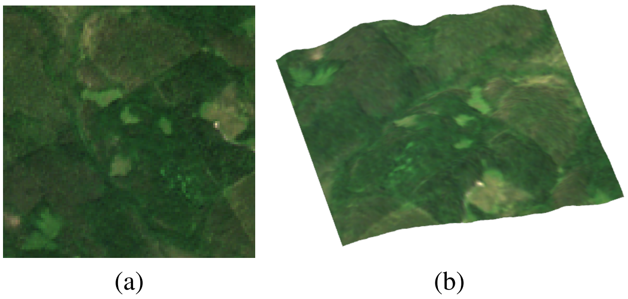


Fig. 4: (a): color patch image obtained by combining the B02 Blue, B03 Red and B04 Green bands from a Sentinel-2 patch. (b): generated 3D mesh resulting from combining the DEM and the RGB bands.

Positioning the camera The camera is positioned at d units of distance from the xy plane, along the z axis. I have tested various distances and the best results are obtained with the camera closer to the scene. Using greater distances yields bad results when later training the model. To obtain different views of the scene, the camera is moved randomly across x and y, and then pointing the camera at the origin of the world. An example of an image taken can be seen in Fig. 5.

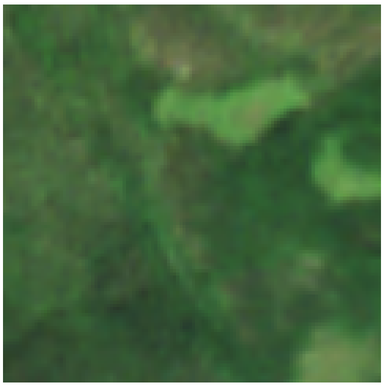


Fig. 5: Image rendered from the generated 3D mesh.

Generating shadows and specularities Later in the project, I added shadows to the scene in order to test how the model tries to fit a more complex lighting model. To achieve this, I have used PyRender's built-in shadow system, which automatically renders shadows on a scene by configuring the proper flags on the renderer. I use a directional light to simulate the sun. Some images are taken with the sun in a zenithal position, thus casting no shadows. Later, I would move the sun to an angle of 45 degrees along the y-axis to cast more pronounced shadows. I have also added cubes texturized with a pattern texture and with different metallic and rough factors to obtain objects with specular reflections on the scene. An image of the scene containing both shadows and cubes can be seen in Fig. 6.

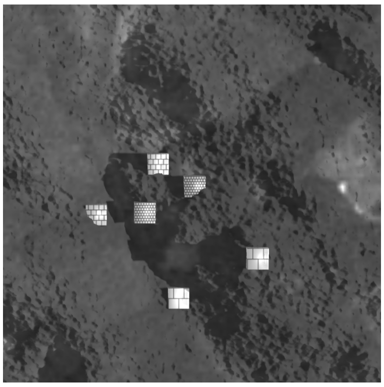


Fig. 6: Synthetic image of band B02 with shadows and cubes for specular reflections.

6.4 Real data generation

We have generated multispectral images from a real scene. For that, we placed a camera with a tripod and a wheel of filters in front of the lens. We took images with 8 different filters and 1 image with no filter. More of this data can be seen in Fig. 20 and Fig. 21 in the appendix. To obtain the camera poses, we reconstructed an RGB image from the bands and applied a sparse reconstruction using COLMAP.

6.5 Own NeRF model

To begin the work, I have developed my own NeRF model. I have done this to learn in-depth about this technology and to familiarize myself with the techniques that I have had to use in the rest of this project, like volume rendering and the peculiarities of how these kinds of models are trained.

Volume rendering I have created a volume rendering pipeline to be able to train the model. This pipeline consists of a camera that casts rays for each pixel of the viewport, taking into account the camera parameters both internal, like the focal, and external, like the camera pose. For each ray, the system selects N points to sample. The sampling point selection strategy is the same as the original NeRF, with random samples inside segments to avoid overfitting the MLP to concrete sample points.

Once the sample points are selected, they are passed onto a sampler (the MLP) which is a function that receives the sample point and the view direction and emits an RGB color and a density. These values are then integrated for each ray, following the formulas in Eq. (2) and Eq. (3).

This volume rendering technique is implemented entirely using PyTorch's tensors and operations.

Positional encoding I have also implemented the positional encoding according to Eq. (1), which decomposes each number in sines and cosines of frequencies of powers of 2 of the value. Originally, this was done by directly computing every value $2^n \cdot \pi \cdot x$ for each n where $0 \leq n < L$, then computing the sine and cosine for each value. This proved to be very slow and caused performance problems for the training and testing code. Later I tried to enhance the implementation by taking advantage of the $\sin$ and $\cos$ functions for the double of an angle. This proved to be faster, but not as fast as the original implementation. Finally, I replaced this implementation with the original NeRF paper's implementation, and that fixed the performance problems.

Model architecture The model architecture is effectively the same as the original NeRF. The network consists of 3 blocks and two positional encoders. The positional encoders encode the sample point and view direction using $L = 10$ and $L = 4$ respectively. Then the encoded position is passed as input to the first block. The first block is composed of 5 linear layers of width w, all layers with an activation on ReLU. The second block accepts the output of the first block plus the original encoded position again, following the skip connection pattern. This block is composed of 4 linear layers of width w, the last one being $w + 1$. The first two with ReLU activation and the latter two without activation between them. From the latter layer, we get the density output and we forward the rest of the block output to the third block. The third block receives the output from the second block plus the encoded view direction. This block consists of a $w/2$ width layer and a 3-width layer. The first layer has a ReLU activation and the last layer has no activation. From this last layer, we get the raw RGB output. This output needs to be post-processed before being used to render the final image. An overview of the network can be seen in 7. One important thing to note is that, except when stated the opposite, the width w of the network I have normally used is 256.

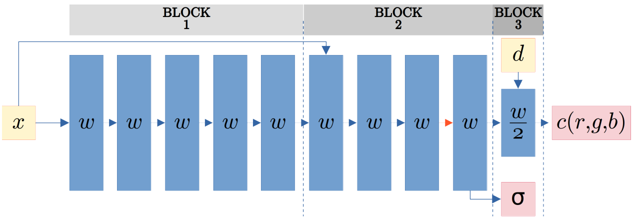


Fig. 7: The original network architecture of the NeRF model. The orange arrow indicates no activation.

6.6 Training with original NeRF images

I have trained and tested the model with the original NeRF images to see if my model worked. I have created a NerfDataset class in Python that is capable of loading the original NeRF Blender dataset. I have trained the original model with this dataset and obtained good results. The training takes 1 day on an RTX 4080, about 150.000 mini-batches go through the network. I will discuss the results in more depth in the results section of this document.

6.7 Training with own synthetic images

The image datasets I created consist of 20 training images, 20 validation images, and another 20 testing images. These images have been generated from the DEM and the different bands of the Sentinel-2 patch, as explained in section 6.3. For each image, I know the pose of the camera and its parameters. Thanks to knowing this information, when the images are loaded, some processing is done to obtain, for each pixel, the origin and direction of the ray that generates the pixel. Taking into account this, each entry in the dataset consists of the final color of the pixel, the origin of the camera, and the direction of the ray.

To train, I used a batch size of 4096 to avoid exceeding the 16 GB memory limit of the graphics card I trained this model on. I use the RAdam optimizer with a learning rate of 0.0005 and betas of 0.9 and 0.999. I also use 128 sample points for each ray. The loss that the training tries to minimize is a simple MSE loss.

Different camera distances I have done experiments, trying out putting the camera at different distances from the scene. This is to check if the distance affects the results of the NeRF. For this, I have generated 7 different datasets that place the camera at 5, 10, 20, 40, 80, 160, and 320 units of distance from the scene, where a unit in this case represents the length of a side of the original DEM, representing 1200 m of distance. A vital part of these experiments is to place the $near$ and far bounds of the rays correctly. If set incorrectly, this can negatively affect the results. For instance, if the $near$ and far bounds are set both too near or too far, the bounds might miss the scene completely, making the model learn an incorrect representation. Or, if the $near$ and far bounds are set too far from each other, most of the sampling points might end up outside the volume, making most of the sample points useless.

Another part I took into account is the view angle from the camera to the z axis. If I had just put the camera further without adjusting the view angle, as the cameras were further, the disparity between images would have been lower, thus presumably yielding worse results when training and testing the NeRF model. To avoid this, I have made sure that how far the camera moves along the x and y axis is proportional to how far the camera is along the z axis. The rule I have followed is that for each distance the camera is afar along the z axis, the camera can move along x and y 20% of that distance. This gives a maximum view angle to the z axis of 8.13 degrees at all distances or 16.26 degrees between images in opposing corners. A greater angle would risk that some images capture the white background outside the mesh. It's important to clarify that the camera moves along the x and y axis independently, inside the area of a square. This makes it so that the disparity for all distances can be the same. Additionally to all this, the camera is always pointed to the origin of the scene. A schema exemplifying how this is done can be found in Fig. 8. The results of these experiments will be shown in the results section.

6.8 Multispectral model

Once the regular NeRF model was working. I expanded it so it could work with more bands than the traditional RGB. The first thing I did was to develop a proof of concept and test it. Later, once the model proved to work, I built the complete model.

Model expansion The main changes to the model consist of adapting the volume rendering pipeline so it is capable of rendering an arbitrary amount of bands and changing the network output to more than 3 channels, as can be seen in Fig. 9.

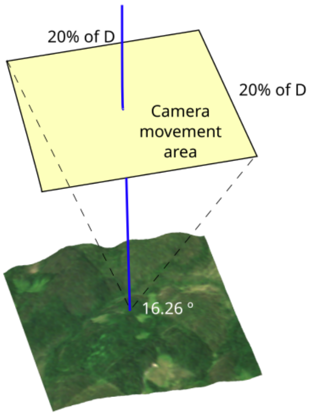


Fig. 8: A schema of how the camera is positioned at any distance from the origin along the z axis. The size of the square that represents the area the camera can move in is 20% of distance D, the distance from the camera to the xy plane at the origin.

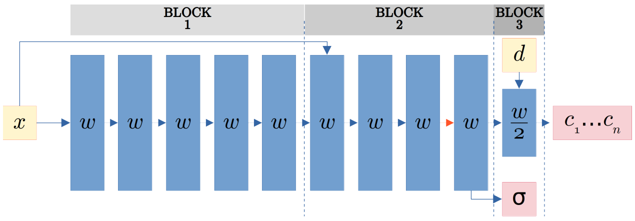


Fig. 9: Network architecture for the expanded model. Note that the only difference is the output of the color on the last layer. This architecture can output N different components for the color.

Data for proof of concept To test the model, as a proof of concept I worked with the original NeRF's blender dataset. To have more than 3 channels I generated 4 channels from the RGB as in Eq. (4).

$$C_1 = R, \quad C_2 = \frac{1}{2}R + \frac{1}{2}G,$$
$$C_3 = \frac{1}{2}B + \frac{1}{2}G, \quad C_4 = B. \tag{4}$$

To test the proof of concept, I downscaled the images to 80 × 80 to speed up the training.

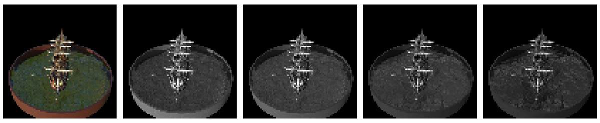


Fig. 10: Original image downscaled to 80×80, and the four generated channels C_1, C_2, C_3 y C_4.

Data for the complete model For the complete model, I used my own synthetic and real multispectral data, which I generated as I explained in the 6.3 subsection.

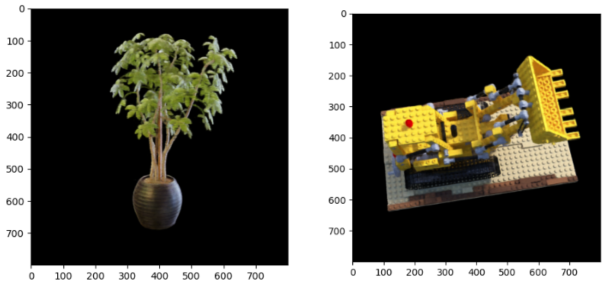
7 Results

Here I explain the results I have obtained for the different experiments I have made. First, I will explain the results

I have obtained when trying the model with the original Blender NeRF dataset. Then the experiments from different distances using our own RGB images, and finally the rest of the experiments I have done with the multispectral model.

7.1 NeRF with blender dataset

I have trained the model using the Ficus and Lego image collections from the original NeRF dataset.



(a) (b)

Fig. 11: (a): Ficus from the blender dataset rendered using the NeRF model, (b): Lego excavator from the same dataset renderer using the NeRF model.

For the Ficus dataset, the validation MSE is 0.0029 and the PSNR is 25.38 dB. As for the Lego dataset, the validation MSE is 0.0018 and the PSNR is 27.45 dB. Rendered images for both these experiments can be seen in Fig. 11.

These results are close to the original NeRF results. The training took about 1 day, but I consider that more training time could improve the model's results.

From a trained MLP it is possible to obtain depth maps using the Eq. (5). This formula obtains the depth of the scene as viewed from the camera, where T_i is the transmittance at a sample point i, α_i is the transparency and t_i is the distance between the point and the origin of the camera. An example of depth can be seen in Fig. 12.

$$d(r) = \sum_{i=1}^{N} T_i \alpha_i t_i. \tag{5}$$

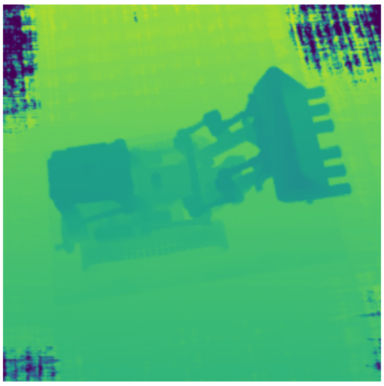


Fig. 12: Depth map extracted from the trained NeRF model.

7.2 NeRF with own synthetic data

I have trained the model using RGB images of the scene from different distances. For each distance, I have trained the model 3 times. The results are displayed in Table 2.

D	Near-Far	$\overline{\text{MSE}}$	$\overline{\text{Depth MSE}}$
5	1-6	**0.000013** $\pm 3.5 \cdot 10^{-7}$	$0.000252 \pm 2.6 \cdot 10^{-4}$
10	5-11	$0.000018 \pm 9.6 \cdot 10^{-7}$	**0.000083** $\pm 3.0 \cdot 10^{-5}$
20	15-21	$0.000018 \pm 6.8 \cdot 10^{-7}$	$0.000113 \pm 6.4 \cdot 10^{-5}$
40	35-41	$0.000020 \pm 4.1 \cdot 10^{-7}$	$0.000116 \pm 6.0 \cdot 10^{-5}$
80	75-82	$0.000023 \pm 3.7 \cdot 10^{-6}$	$0.002301 \pm 1.4 \cdot 10^{-3}$
160	155-165	$0.000030 \pm 1.2 \cdot 10^{-6}$	$0.060041 \pm 6.2 \cdot 10^{-2}$

Table 2: Results for different distances (D). The MSE and Depth MSE represent the average and the standard deviation of 3 experiments for each distance.

The plot in Fig.13 shows the tendencies of both MSE and depth MSE in a more clear way. The conclusion we can take from it is that decreasing the distance yields better results, and increasing the distance of the camera to the scene yields worse results. The results for the 320 distance are significantly worse than the others, so they have been omitted from the plots. These results are surprising to me because I think I have set the right conditions so that this doesn't happen. The near and far are properly set and, when moving the camera further away, the view angles are kept the same so that when viewed from any distance the disparity between images can be the same. With this configuration, the only possible explanation for these results is that the distance from the object is the cause of the degradation of the results, especially when computing the depth.

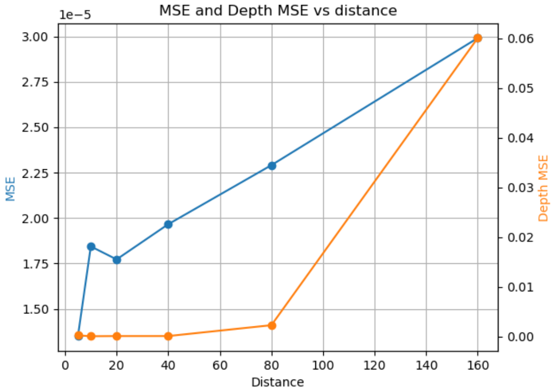


Fig. 13: The MSE and Depth MSE against the distance a which the images were taken.

7.3 Multispectral NeRF (MSNeRF)

This work featured in this section is focused on using NeRF with multispectral images. This is training and testing the model with images with more than the 3 RGB bands. First I will explain the results of a proof of concept model I developed, and then I will explain the results of training with synthetic data and real data.

7.3.1 MSNeRF proof of concept

I have trained a version of the model that supports 4 channels instead of 3 RGB. The test training and test images were based on the Ship images from the original Blender NeRF dataset. These images have been downscaled to 80 × 80, which is a very low resolution so that the training

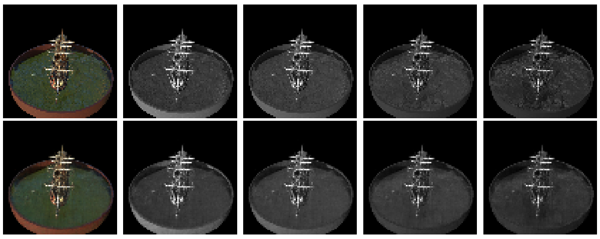
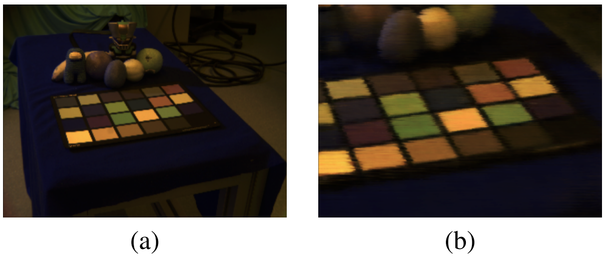


Fig. 14: Comparison of the original bands (TOP) against the reconstructed bands (BOTTOM).

would be fast. The resulting MSE of the test images against the predicted images is 0.0031, and the PSNR is 25.09 dB. A visual example of the results can be seen in Fig. 14. This has proved that a NeRF model with more than 3 channels is possible to make, even though it is just s proof of concept.

7.3.2 MSNeRF real scene

As another experiment, I have trained the NeRF model using the real multispectral images we captured. For this experiment, I selected 57 images for the training dataset and 2 for the test dataset. The results aren't as good as with the satellite images. When reconstructing the images, the resulting image is blurrier than the original, as can be seen in Fig. 15. The resulting test MSE for this experiment is 0.0325. I think this happens because real data is not as perfect as synthetic data, it contains noises and defects, and my NeRF model or the dataloaders or the renderer might not be completely prepared to deal with this data. Nonetheless, the results are acceptable and the different parts of the scene can be recognized in the reconstructions.



(a) (b)

Fig. 15: (a): original image of the real scenario, composite of the 3 first bands. (b): detail of the reconstruction of the image using NeRF, also a composite of the 3 bands, it's possible to see the defects of the reconstruction.

7.3.3 MSNeRF synthetic and satellite

I have experimented with this model to determine the network's necessary width to represent the scene. The width of the network refers to the amount of neurons each layer has, except for the last layer, which has half the width. For the base architecture, this width is 256. I have done these experiments because the complexity of the model determines how complex the scene can be. The results can be seen in 3. The table, apart from containing the MSE and depth MSE, also contains the reconstruction time for one image on a GTX Titan X. It can be seen that the thinner the network, the faster it reconstructs images.

Width	MSE	Depth MSE	Reconstruction time per image
16	0.000501	0.004041	**4.2 s**
32	0.000190	0.000240	5.6 s
64	0.000086	0.000302	8.0s
128	0.000031	0.000048	10.8 s
256	0.000017	0.000086	22.0 s
512	**0.000016**	**0.000019**	62.7 s
1024	0.000017	0.000018	250.2 s

Table 3: Results for the multispectral NeRF depending on the width of the network.

In the plot Fig. 16 I show the tendencies of both MSEs when the width changes.

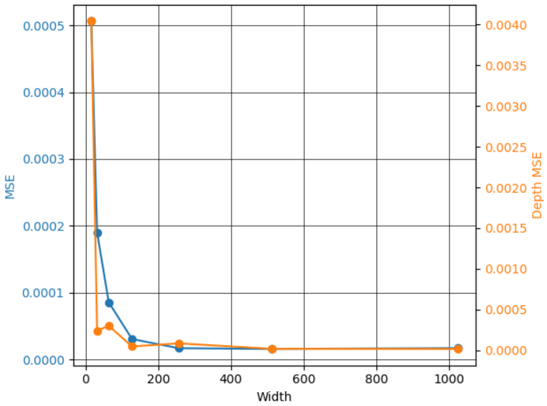
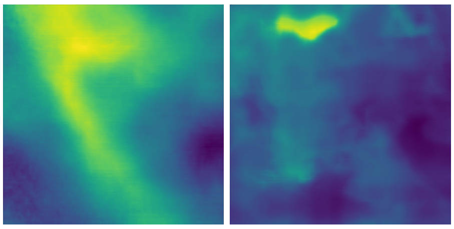


Fig. 16: The MSE and Depth MSE against the width of the network.

From the plot, we can understand that when the network is sufficiently wide, it can accurately model the terrain. But when it is too thin, it cannot model correctly the scene, which we can see as a higher MSE and depth MSE. In Fig. 17(a), we can see the depth perceived by a 256-width network. It's visually similar to part of the original DEM in Fig. 2. But if the network is too thin, it doesn't learn the depth properly, as in Fig. 17(b), which doesn't resemble the original DEM in any way.



(a) (b)

Fig. 17: (a): depth prediction using a 256-width network resembling the original DEM, (b): depth prediction using a 16-width network, which doesn't resemble the original DEM.

I have also experimented with a shorter network, which is based on the original network with layers 2, 3, and 7 removed for a more compact architecture while keeping the width of 256. The model gives an MSE of 0.000044 and a depth MSE of 0.000238, and needs 15.8 seconds to reconstruct an image. These results are close to the original net, proving that the net can be optimized in size without affecting much of the results.

7.3.4 Experiments with more complex lighting

For the last experiments, I have used the multispectral dataset generated with shadows and cubes.

Shadows	Cubes	MSE	Depth MSE
No	No	**0.000010**	**0.000023**
Yes	No	0.000135	0.000083
No	Yes	0.000109	0.000042
Yes	Yes	0.000385	0.000246

Table 4: Results for the multispectral NeRF depending on whether shadows and cubes are present or not.

From Table 4 it can be understood that both shadows and the presence of cubes that generate specular reflections negatively affect the performance of the model. It appears as if the shadows affect more negatively than the cubes, but it's not entirely clear. Also, it seems that combining both yields even worse results. Even when giving the worst results, when qualitatively comparing the original images with the reconstructed images, no apparent differences can be seen. An example of this is in Fig. 18 in the appendix.

8 Conclusions

With the current work, I have achieved to develop a NeRF model that gives similar results to the original NeRF. I have also adapted this model to work with multispectral images, which was one of our main objectives. This shows how NeRF is promising when trying to model scenes from satellite or aerial images with more bands than RGB.

The main challenges I have faced come from the complexity of these types of models, the amount of parameters they have, and the sensitivity of the results to the parameters. I have had to learn a lot about NeRF before achieving the first working version of the model. The adaptation of the model to work on multispectral images was also an interesting challenge. On the one hand, having had to generate the necessary synthetic data, and on the other hand, having built a multispectral dataset from a real scene. Finally, modifying the network and the renderer to support multiple bands.

The specific conclusions of the experiments give us that it is important to adjust the $near$ and far bounds to the size of the scene. The camera distances should not be too far away from the scene, no more than 80 times the scene size. This implies that it will adapt better to aerial than satellite images. As expected, working with real data gives worse results than synthesized data, although they are still acceptable. We could improve the reconstruction time by reducing the network (width and depth) while obtaining good results. Finally, introducing light conditions, shadows, and specular behavior reduces the performance.

In the future, I consider that it would be interesting to try and use a model close to EO-NeRF with the synthetic data. This would approach the work done to the latest techniques in the field. I also consider that it would be interesting to leave NeRF behind and try to explore the world of multispectral Gaussian Splatting.

Acknowledgments

Thanks to the CVC for providing me with the resources to make this work possible, and especially thanks to Felipe Lumbreras Ruiz for being my tutor and mentoring and guiding me through this project.

References

[1] Dawa Derksen and Dario Izzo. Shadow neural radiance fields for multi-view satellite photogrammetry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1152–1161, 2021.

[2] GISGeography. Sentinel 2 bands and combinations, 2023. Last accessed on January 12, 2023.

[3] Jiabao Li, Yuqi Li, Ciliang Sun, Chong Wang, and Jinhui Xiang. Spec-nerf: Multi-spectral neural radiance fields. *arXiv preprint arXiv:2310.12987*, 2023.

[4] Roger Marí, Gabriele Facciolo, and Thibaud Ehret. Sat-nerf: Learning multi-view satellite photogrammetry with transient objects and shadow modeling using rpc cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1311–1321, 2022.

[5] Roger Marí, Gabriele Facciolo, and Thibaud Ehret. Multi-date earth observation nerf: The detail is in the shadows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2034–2044, 2023.

[6] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[7] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[8] Matteo Poggi, Pierluigi Zama Ramirez, Fabio Tosi, Samuele Salti, Stefano Mattoccia, and Luigi Di Stefano. Cross-spectral neural radiance fields. In *2022 International Conference on 3D Vision (3DV)*, pages 606–616. IEEE, 2022.

[9] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.

[10] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

APPENDIX

A.1 Strategy for preprocessing the DEM

This is the process I have followed to reduce the quantization of the DEM. The strategy consists of applying uniform noise from -1 to 1, then upscaling the original DEM ×2. Then applying noise from -0.25 to 0.25 and upscaling again. Then applying noise from -0.0625 to 0.0625 and upscaling again. With this process, the visible jumps are not completely eliminated but are greatly reduced.

A.2 Some visual results for an experiment with shadows and cubes

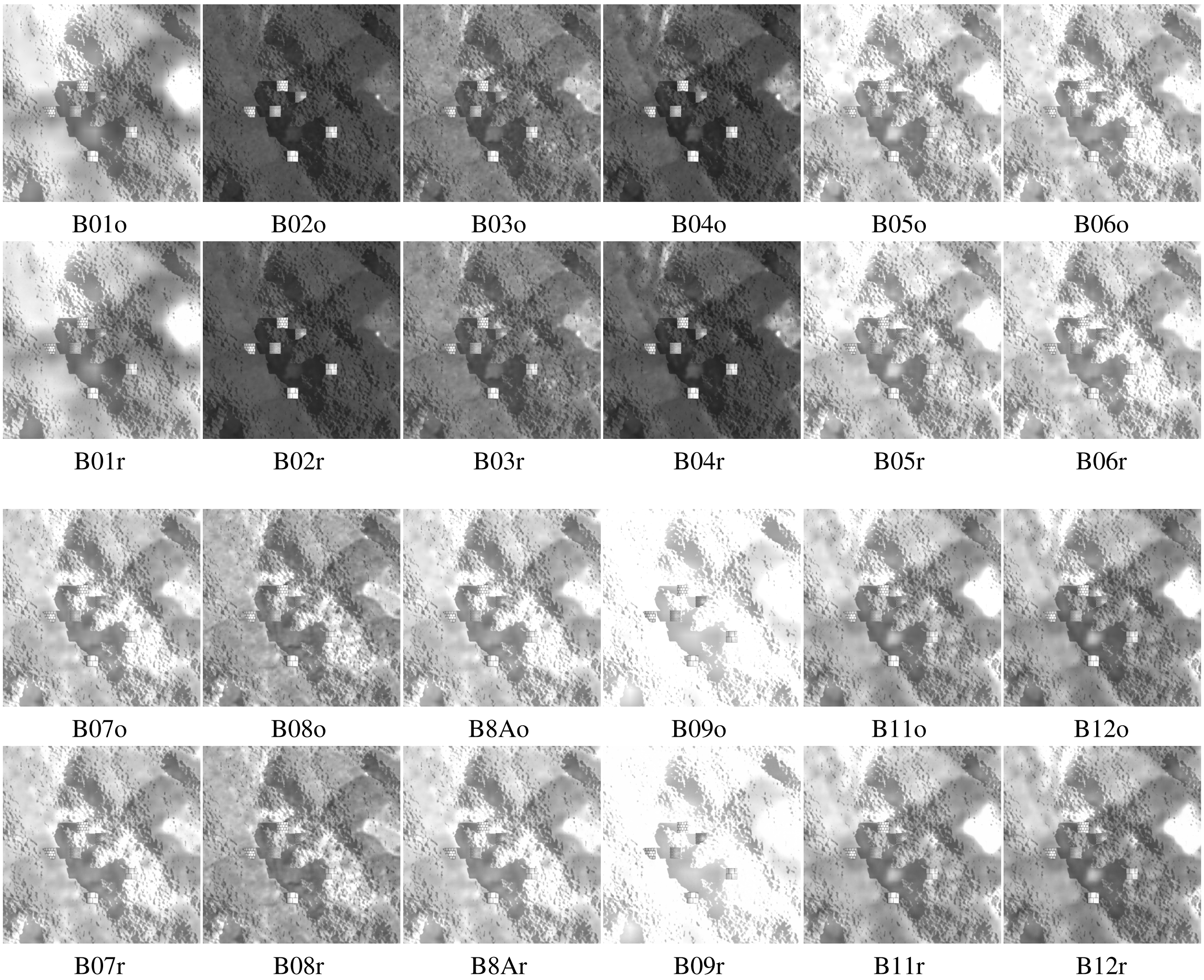


Fig. 18: In the upper part bands B01 to B06, BXXo means original band, while BXXr means reconstructed band. Bands B07 to B12 in the lower part.

There are hardly any differences between the reconstructed bands and the original ones, as shown by the low reconstruction error shown in Table 4.

A.3 Observation about ship depth estimation

In the experiment for the proof of concept multispectral model, an interesting thing can be observed. Even though the images are of a resolution of only 80 × 80, the model is capable of estimating depth, as can be seen in Fig. 19.

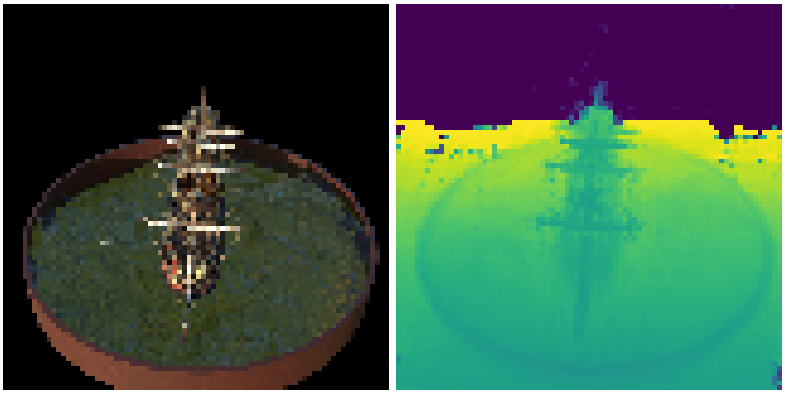
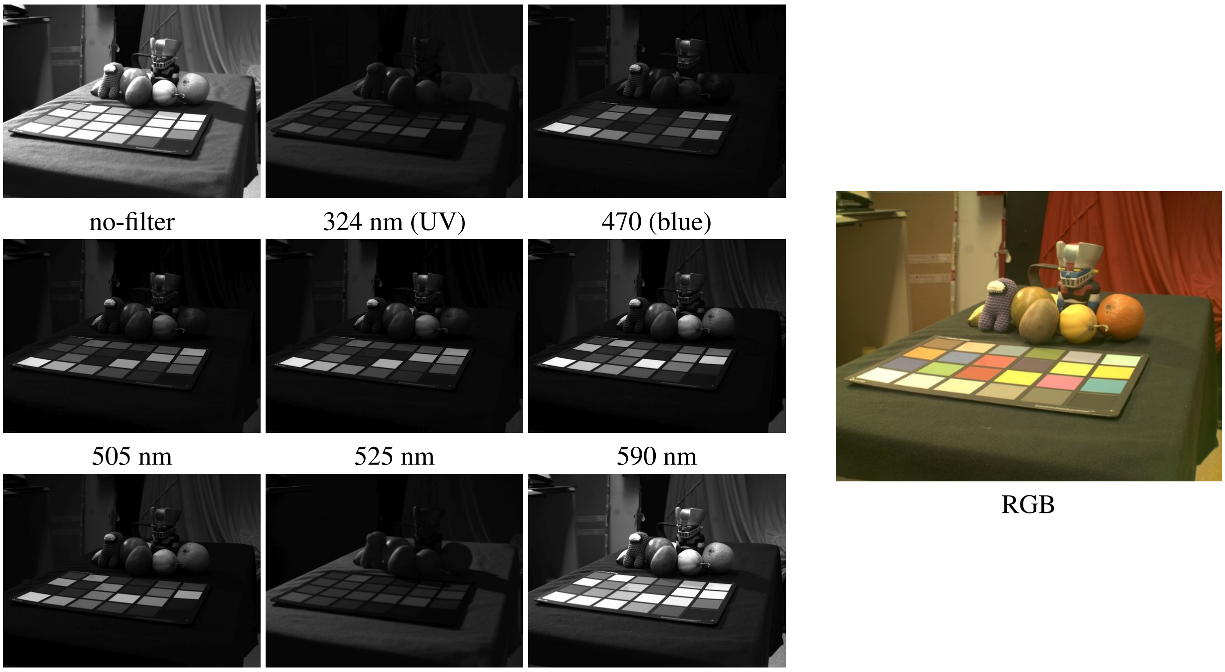


Fig. 19: The depth map of the ship model at 80 × 80.

A.4 Mutispectral NeRF with real data

In this part, we will show in more detail the data captured and used in this experiment. We have acquired a static scene at the basement of the CVC. The scene consists of a set of figures and fruits with a MacBeth Colorchecker on top of a table. The images are taken from around this scene from 10 different angles, 5 different heights, and 9 bands, in total 50 multispectral images. Then an extra set of nine different multispectral images. Total 59 images. The multispectral information has been acquired with a 5MP mono camera (Basler ace acA2500-14gm) and a wheel of filters. These filters are from MidOpt (`https://midopt.com/` (BP324-27 (UV), BP470-27 (blue), BP505-27 (cyan), BP525-27 (light green), BP590-27 (orange), BP635-27 (light red), BP850-27 (NIR), BP550-27 (visible)))



no-filter

324 nm (UV)

470 (blue)

505 nm

525 nm

590 nm

635 nm (dark red)

850 nm (NIR)

visible band

RGB

Fig. 20: On the left, a set of images captured for the second camera height. The first one is a no-filter image that corresponds to the panchromatic image, seven images are acquired in different bands pass filters (bp) with central wavelength in the figures. The last image corresponds to a wide bandpass related to the visible spectrum. At the right part RGB composite image form bands: blue 470 nm, light green 525 nm, light red 635 nm.

We can see the preliminary results of Gaussian Splatting in Fig. 22. Future work may try to extend this algorithm to multispectral images.

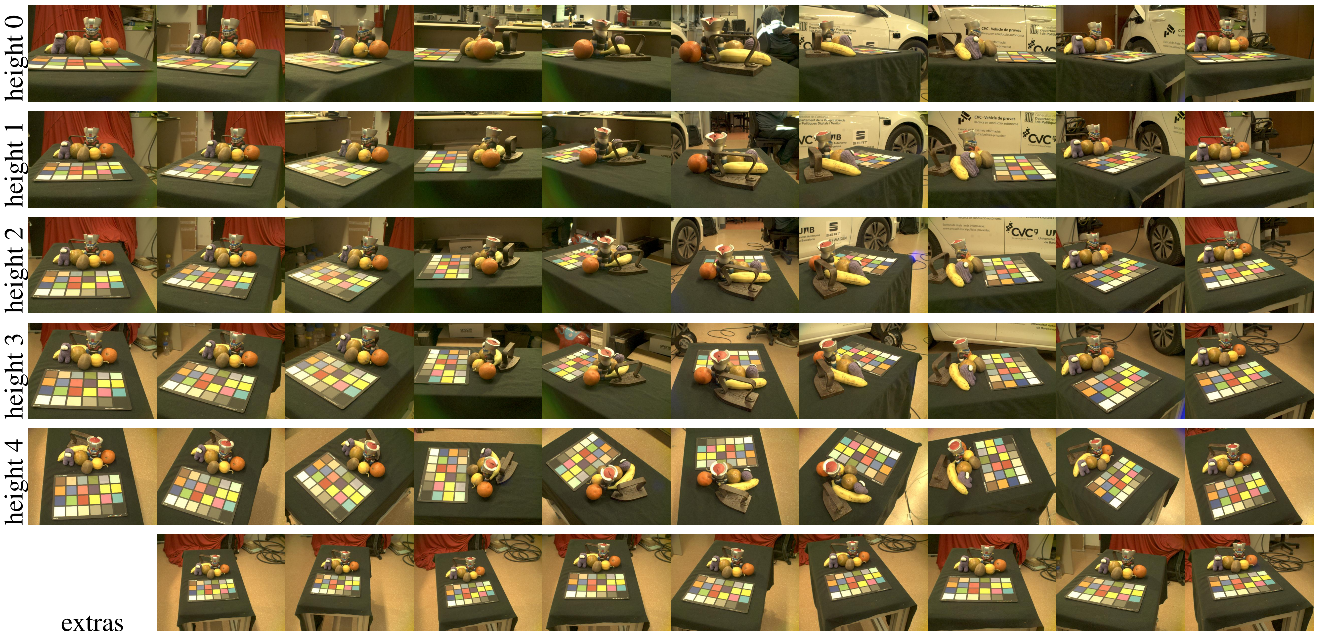


Fig. 21: The different views for the RGB composite. There are the same number of views for each monochromatic band (not shown in the figure) that are the images involved in the NeRF reconstruction.

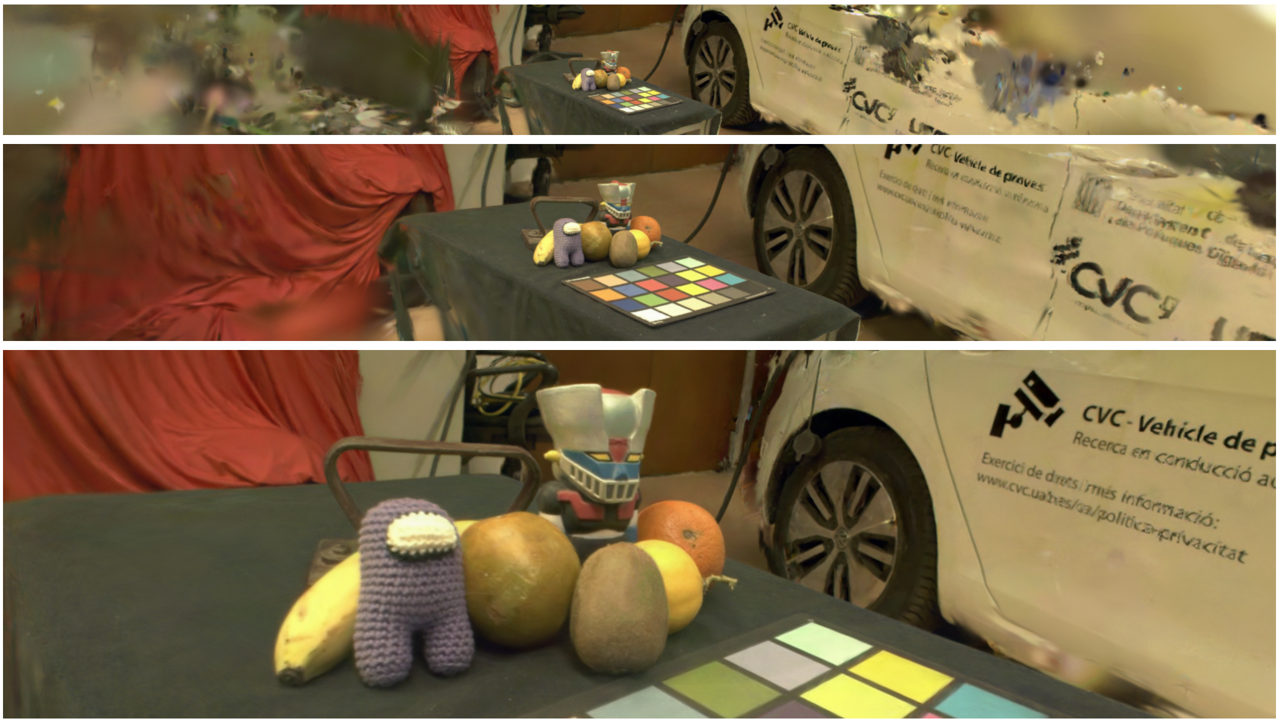


Fig. 22: Gaussian Splatting of the sequence of RGB images. Generated with LUMA (`https://lumalabs.ai`). Full link to the reconstruction: `https://lumalabs.ai/capture/ef40026c-97d9-4cbf-bc8f-472e51c071ad`.