Universitá degli Studi di Padova

Department of Information Engineering

Master Thesis in ICT for Internet & Multimedia

Non-Line-of-Sight Imaging from iToF data

Student

Matteo Caligiuri

ID 2019283

Supervisor

Prof. Pietro Zanuttigh

Universitá degli Studi di Padova

Co-supervisor

Dr. Gianluca Agresti

Sony Europe B.V.

Co-supervisor

PhD Adriano Simonetto

Universitá degli Studi di Padova

Academic Year
2021/2022

Abstract

The project aims to perform a feasibility study on the field of Non-Line-of-Sight imaging using indirect Time of Flight and Deep Learning. Throughout the various chapter of this work, will be theoretically presented how Time of Flight cameras works and will be performed an in-depth analysis of `Mitsuba Renderer 2` (together with some of its forks). After that, a newly synthetic dataset will be designed and rendered from the ground up. Subsequently to these preliminary steps the *Fermat flow* approach to NLoS imaging will be presented and evaluated. To conclude this work we will introduce an innovative way to reach the desired goal exploiting a smart trick together with a Neural Network model.

Sommario

Lo scopo del progetto è quello di eseguire uno studio di fattibilità riguardo la percezione di scene in Non-Line-of-Sight usando un modello di Deep Learning assieme ad un sensore indirect Time of Flight. Attraverso i vari capitoli sarà inizialmente presentata una trattazione teorica su come le camere Time of Flight funzionano, dopodiché verrà eseguita una dettagliata analisi di `Mitsuba Renderer 2` (assieme ad alcuni suoi forks). Una volta fatto ciò verrà introdotto un nuovo dataset sintetico progettato apposta per questo specifico task. In seguito a questi passaggi preliminari verrà presentato e valutato il metodo del *Fermat flow* per percezione in NLoS. Per concludere verrà, infine, introdotta un'innovatia strategia capace di raggiungere lo scopo prefissato sfruttando un escamotage combinato con una rete neurale.

Contents

List of Figures

List of Acronyms

CGI	Computer-Generated Imagery
LoS	Line-of-Sight
ToF	Time of Flight
NLoS	Non-Line-of-Sight
FoV	Field of View
iToF	indirect Time of Flight
NN	Neural Network
dToF	direct Time of Flight
LiDAR	Light Detection And Ranging
BRDF	Bidirectional Reflectance Distribution Function
CV	Computer Vision
CM	Continuous Modulation
CWIM	Continuous Wave Intensity Modulation
PB	Pulse Based
SLP	Shuttered Light-Pulse
NIR	Near-InfraRed
SNR	Signal-to-Noise-Ratio
PMD	Polarized Mode Dispersion

IR	InfraRed
VGA	Video Graphics Array
MPI	Multi-Path Interference
MAE	Mean Absolute Error
EMD	Earth Mover's Distance
RTT	Round Trip Time
DL	Deep Learning
HDR	High Dynamic-Range
OS	Operating System
QVGA	Quarter Video Graphics Array
SPAD	Single-Photon Avalanche Diode
CNN	Convolutional Neural Network
MAE	Mean Absolute Error

1 Introduction

1.1 Aim of the project

Nowadays a huge amount of research interest is shifting towards the *3D imaging* field. This behavior is mainly justified by the fact that this topic can find endless usage in a wide variety of different applications e.g., scene reconstruction, autonomous driving, 3D modeling in Computer-Generated Imagery (CGI) and/or video games. That said, it is important to give a proper explanation of *3D imaging*. It is a process that, using a specific set of sensors, aims to capture the most possible 3D information about a given scene in the real world. This essentially means recovering all the depth information of the environment and in some cases also texture and material information, that can be used to completely and accurately rebuild the real-world scene inside a digital environment.

As said this topic is gathering a lot of attention, but the research is still not fully mature, for this reason, at the current time there exists a lot of different ways to perform *3D imaging*. In general, the various possible acquisition techniques can be subdivided into two main categories [1]:

- Passive system: uses the reflectance of the object and the illumination of the scene to retrieve the shape information (e.g., Stereo vision);
- Active system: requires an active light source that projects an electromagnetic signal onto the subject, then uses the reflection of this signal to retrieve depth information (e.g., ToF).

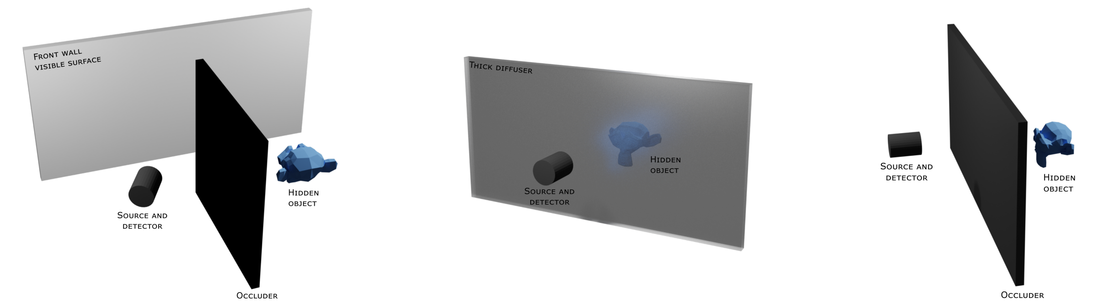
Of course none of the two presented approaches is perfect, each one of them has its strengths and weaknesses. In particular, the biggest limitation of passive systems is that they strongly require the scene under analysis to be well, and more importantly, uniformly illuminated[1]. Otherwise, the final reconstruction will probably end up having low quality. On the other end, the active system relies on its source of illumination and so, they do not suffer from such a limitation. This is a key aspect that makes technologies like ToF more powerful in a lot of scenarios since it is generally more robust and consistent across different environments. For this reason, the focus of the project is mainly concentrated on active systems.

Regardless of the used technology, the basic setup used to acquire 3D information from a real-world scene is composed of a sensor, or a pair of them (stereo system), pointed directly toward the subject to ensure that there is a clear path between the two entities. These settings fall under the name of Line-of-Sight (LoS) acquisition. This is the most common and diffuse approach in the *3D imaging* field since it is the most straightforward way to implement it. As a result, most of the efforts are, and have been, focused on the refinement of the sensors and software implied to reconstruct such scenarios.

Only in recent years, the scientific community has begun to explore novel ways of using *3D imaging* technology, in particular the active ones. One of the most interesting tasks that are possible to carry out is Non-Line-of-Sight (NLoS) imaging. This essentially means using a traditional acquisition system to capture a scene/object located outside of the direct FoV of the sensor. This category of imaging goes one step further concerning its LoS counterpart, analyzing the light scattered from multiple surfaces along indirect paths to reveal the 3D shape of the object located in the NLoS [2]. This added step introduces several new challenges to the reconstruction pipeline. In particular, under the NLoS condition only a few of the many recorded photons carry information related to the hidden object compared to the LoS case in which almost every photon is carrying useful information. Other than that it is important to keep in mind that the signal strength of multiply scattered light[2] decreases several orders of magnitude faster compared to the case of a single reflection. All of this makes the process of getting a robust detection and reconstruction, in a NLoS setup,

[1]In order to allow the system to be able to extract consistent features every object must retain the same color from all the points of view, for this reason, the illumination should be uniform

[2]The case of multiple scattering is extremely common in situations like the "look around the corner" (fig. 1.1a) were to have information about the hidden object each light ray has to be reflected at least three times to reach back to the sensor



(a) Looking around the corner

(b) Looking behind a diffuser

(c) Looking through a wall

Figure 1.1: Overview of the considered NLoS scenarios

much more difficult compared to the LoS case.
The NLoS imaging term groups under itself a lot of different scenarios, all of them linked by the fact that the target subject is outside of the line of sight of the used sensor. But there could be many different ways for an object to be hidden. For this reason, it is possible to distinguish three main categories of NLoS perception scenarios:

- Looking around the corner [3] (fig. 1.1a);
- Looking behind a diffuser [3] (fig. 1.1b);
- Looking through a wall [4] (fig. 1.1c).

All of these different scenarios have their related challenges. So for each one of them, it is required to use different technologies to succeed in the reconstruction of the hidden scene. In this work, the focus will be on the first category.

The goal of this project, as the title suggested, is to use an iToF sensor to perform NLoS *3D imaging* in a "look around the corner" setup. More precisely the aim is to build a pipeline that, given the raw iToF data in input, can retrieve the point cloud of an object located behind a corner, so outside of the direct FoV of the camera. To accomplish this task the idea is to develop a Neural Network (NN) model that takes in input the iToF data and gives as output the point cloud of the object in the NLoS scene. Other than that it is also necessary to develop and build a proper synthetic dataset that will be used to train the network as there are none suitable already available in the literature.

To ensure that the implemented approach can be fully employed in its field of application, it is crucial to keep in mind that the produced system has to work in real-time (or close to it) and also must be as small and as cheap as possible. All of these constraints will sum up to an already challenging task full of limitations. In this work, to deal with that, it has been decided to use an iToF sensor instead of dToF or Light Detection And Ranging (LiDAR). The iToF has been chosen mainly for three reasons: it is generally cheaper and smaller than its counterparts still guaranteeing a greater lateral resolution[3] [5]. This makes it seems more suitable for being integrated inside other devices like cars, robots, helmets, etc. Other than that to ensure the real-time requirements, as will be described in chapter 6 the NN model in charge of the 3D reconstruction has been designed to be as small and as light as possible. Another aspect to keep in mind is the fact that we aim to use an off-the-shelves device, not one designed and built ad-hoc for our experiments. This is due to the will of generating a solution that can easily be integrated into a lot of existing devices. Of course, all of these constraints will make the research and development phases harder.

1.2 Motivation behind the project and field of use

As it is possible to imagine, retrieving the point cloud of a subject located outside from the camera's direct line of sight is not straightforward. At the same time however being able to see the unseen could be extremely useful in many different scenarios like robotic vision, remote sensing, medical imaging, autonomous driving, military, and many other domains [2]. In all of these situations, the ability to see something that is hidden can greatly improve the quality and the degree of the information gathered from the environment. Just to give a simple example, it is possible to consider a car (provided with autonomous driving) stopped at an intersection. NLoS imaging allows such a car to know exactly what there is on each side of the intersection, also in areas that are hidden from its point of view. In this way, the autonomous system of the car can perfectly know the environment and potentially take the best decision on how to handle that intersection. Doing so the car is now able to perform an action knowing even more than what a human driver could have known, further reducing the possibility of an accident. Such reasoning could be extended to many more scenarios (e.g., an autonomous robot inside a factory). This gives an

[3]Lateral resolution = minimum distance that can be distinguished between two reflectors located perpendicular to the direction of the light beam. This corresponds to the minimum distance that can be perceived between two objects located adjacent to each other.

idea of how powerful NLoS imaging can become when it will be fully developed. Of course, since the development of this kind of technology is just in its early stages at this point, it is not possible to reach such performances, but we can still envision the future potential of this instrument. For this reason, working on its development is necessary to allow this promising tech to evolve toward its final stage.

2 Time of Flight cameras

2.1 Introduction

To properly understand the work that will be done through the following chapters and sections it is essential to have a fair good knowledge about Time of Flight (ToF) sensors. More precisely this chapter will aim to present how ToF works. After that will be covered how the two main variants of this technology (iToF and dToF) work. In particular, to fully understand the choices that will be made further in the project, the differences between the two implementations will be carefully discussed.

Apart from discussing how this type of sensor work it is also important to understand how it is possible to map the measurements produced by one kind of implementation to the other. This mapping is a key aspect that allows, to some extent, to interchange the two types of sensors increasing the overall possibilities of application of this technology. And, more importantly, in this way it is possible to put together the strengths of both approaches while minimizing the weaknesses to achieve better final performances.

Finally, the chapter will close by explaining the theory that is behind the NLoS perception based on ToF sensors. In other words will describe why, at least in theory, using such a sensor it is possible to properly retrieve the point cloud of an object located around a corner.

2.2 Time of Flight sensors

Time of Flight cameras represent a really interesting and efficient way to capture 3-dimensional information about a real-world environment in real-time.

A ToF sensor could be seen as a specialized digital camera sensor that, instead of capturing an RGB image of the observed scene, aims to capture a depth map of the same scene. If we look at ToF under this perspective we can easily understand that also for it the *Calibration* step represents an essential part of the pipeline that is used to obtain a usable output. Of course, a ToF camera is built differently from a traditional one and also uses more complex on-board technologies, which bring to the generation of completely different errors. For these reasons, the calibration procedure, in this case, is quite different from the standard one used in Computer Vision (CV) applications to extract RGB images. It is also important to notice that ToF cameras other than performing a more complex task use also a low-resolution sensor due to the more complex pixel architecture wrt RGB cameras. This lead to a strong reduction in the quality of the measurements and leads to a much more complex procedure that is required to extract the final output [6].

Sadly, due to the core approach used to capture the scene, ToF sensor is subject to a large number of measurement errors. In the latest years, it has been performed a wide number of investigations about this topic and it has been discovered that the ToF error sources could be of many different types. Some examples could be:

- camera parameters and properties,
- environment configuration,
- dependence from the Bidirectional Reflectance Distribution Function (BRDF) property of the measured scene[1],
- sensor hardware.

Note that for how ToF works, at every measurement instant, it provides simultaneously two different information: the depth and the amplitude images. The first one represents, of course, the measured depth map of the observed environment, while the latter corresponds to the amount of returning active light

[1]Since ToF are active sensors, the reflectance property of the objects influences the reflected light measured by the sensor affecting the Signal-to-Noise-Ratio (SNR)

signal. The amplitude image is also used as an indicator of the quality of the measurement.

2.2.1 ToF cameras working principles

It is important to keep in mind that exist different types of Time of Flight cameras that are based on different working principles. The two main approaches to building this type of camera are based on:

- Continuous Modulation (CM) approach also known as Continuous Wave Intensity Modulation (CWIM) characterize the iToF,
- Pulse Based (PB) approach also known as Shuttered Light-Pulse (SLP) characterize the dToF.

In the following, it will be explained how these two strategies work. Keep in mind that since CM is the most diffused one (and it is also the architecture used for the whole project), the coverage of the errors and possible corrections will be primarily focused on this specific ToF implementation.

2.2.2 Indirect Time of Flight (Continuous Modulation approach)

This kind of ToF implementation is also known as *indirect Time of Flight (iToF)* because the distance is not directly measured from the time of flight of the light ray, but instead, it is calculated indirectly starting from the phase shift ϕ.

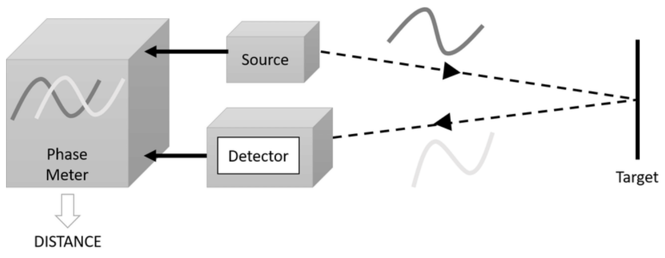


Figure 2.1: Schematized representation of a ToF camera implemented using the CM approach [7]

The Continuous Modulation approach (fig. 2.1) is based on the correlation of the emitted signal o_τ, shifted by an offset phase τ and the incident signal r resulting from the reflection of the active Near-InfraRed (NIR) light by the observed environment. More precisely theoretically it is assumed that the camera is composed of two separate elements: the emitter (or light source) and the sensor. Ideally, these two devices are colocated, of course, in a real device, it is not possible so it is necessary to perform some additional steps to compensate for the distance between the two elements. As it is possible to see in fig. 2.1 the source shoot an amplitude-modulated light signal toward the scene that will be reflected back to the detector. The latter will convolve the received signal with a square signal characterized by the same frequency f_M as the emitted one. From this convolution is possible to extract the phase shift $\Delta\phi$ between the two signals and that is used to compute the distance value.

ToF devices based on this principle used CWIM together with a correlation function $c(t)$ (directly implemented in the hardware using a shutter and a pixel sensitivity modulation process) to estimate the distance between the source of the light and the target.

$$c_\tau(t) = r(t) \otimes o_\tau(t) = \lim_{T\to\infty} \int_{-T/2}^{T/2} r(t) \cdot o_\tau(t) dt. \tag{2.1}$$

Note that the correlation function usually is computed at specific phase offset samples $\tau = 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$. The emitted and incident signals can both be expressed as cosinusoidal signals as follow:

$$o_\tau(t) = \cos((\omega_m + f_m) \cdot t), \qquad r(t) = I + A\cos(\omega_m t + \phi) \tag{2.2}$$

where:

- f_m represents the modulation frequency,
- $\omega_m = 2\pi f_m$ represents the angular frequency of f_m,
- I represents the offset of the signal,
- A represents the amplitude of the reflected signal,
- ϕ represents the phase shift, that is directly related to the distance of the target object from the sensor.

Considering the cosinusoidal nature of the two signals it is possible to use some trigonometrical relations to simplify (2.1) into:

$$c_\tau = \frac{A}{2}\cos(\tau + \phi) + I. \tag{2.3}$$

From (2.3) we can notice that we have three unknowns: A, ϕ and I. For this reason performing a single estimation of amplitude, distance, and offset is required to perform at least three different measurements. Usually ToF cameras acquire four consequent samples $S_i = c_\tau$ of the correlation function c at specific discrete phase offsets: $\tau = 0,\ \frac{\pi}{2},\ \pi,\ \frac{3\pi}{2}$. Of course, in case of noise, to increase the obtained SNR it is possible to increase the number of measurements[2]. From the three consequent measurements it is possible to extract the parameters of interest by exploiting the following equations:

$$\phi = \arctan\left(\frac{S_3 - S_1}{S_0 - S_2}\right), \tag{2.4}$$

$$I = \frac{1}{4} \cdot \sum_{i=0}^{3} S_i, \tag{2.5}$$

$$A = \frac{1}{2} \cdot \sqrt{(S_3 - S_1)^2 + (S_0 - S_2)^2}. \tag{2.6}$$

From that we can simply compute the target object distance d using the phase ϕ, the modulation frequency f_m and the speed of light $c \approx 3 \cdot 10^8 m/s$:

$$d = \frac{c}{4\pi f_m}\phi. \tag{2.7}$$

In the above computation, it has been performed quite an important approximation, since it has been considered that the sensor and the illumination module are placed in the same position, a physically impossible aspect.

This approach is quite simple but, when used, is important to keep in mind that since it is based on phase shift calculation, it allows to reliably measure only a range of distances within one unambiguous interval $[0,\ 2\pi]$. Since the allowed distances depend on f_m we can compute the maximum measurable distance as:

$$d_{max} = \frac{c}{2f_m}.\ ^{3} \tag{2.8}$$

[2]It is not possible to increase the number of measurements indefinitely since increasing the number of obtained samples also incorporates some additional errors, like motion blur

[3]The factor 2 is due to the fact that the light has to travel back and forth between the target object and the camera

2.2.3 Direct Time of Flight (Pulse Based approach)

The Pulse Based approach works by generating light of known dimension coupled with a fast shutter observation. More precisely in this case the camera will project a NIR light pulse of known duration and discretize the front of the reflected illumination. The discretization is performed before the return of the whole light pulse using a fast camera shutter. The resulting portion of the reflected light signal is the element that describes the observed object. Using this approach the depth of the observed scene is directly linked to the time delay of the received signal.

This time the depth of interest is related to the duration of the light pulse and the one of the shutters $(t_{pulse} + \delta_s)$[4]. So, theoretically, ignoring all the sources of error it is possible to compute the depth directly from:

$$d = \frac{c}{2 \cdot (t_{pulse} + \delta_s)} \tag{2.9}$$

The output produced by this kind of sensor is sensibly different from the ones produced by an iToF. In fact it returns for each sensor's pixel a *transient vector* (also known as *backscattering vector*). Analyzing this type of output it is possible to easily separate the *direct component* x_d from the *global component* x_g as we can see in fig. 2.2. The first peak represents the first returning reflection, so the closest object to the sensor, while the global component represents all the other incoming light rays.

This ToF implementation is also known as *direct Time of Flight (dToF)* since it directly retrieves the distance map from the time that the light ray spent going from the source to the object and back again to the sensor.

2.2.4 Differences

From sections 2.2.2 and 2.2.3 it is clear that both, iToF and dToF are based on the same principle: measuring depth based on the time traveled by the light from the source back to the sensor ($source \rightarrow object \rightarrow sensor$). Despite that, the two implementations are fundamentally different. Indeed, they produced two completely different outputs.

[4] t_{pulse} represents the duration of the time pulse while δ_s is the shutter time

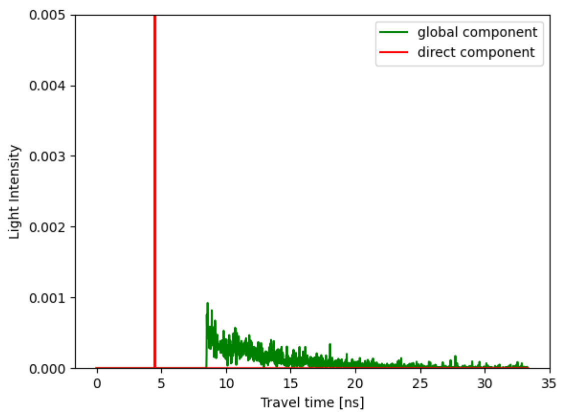
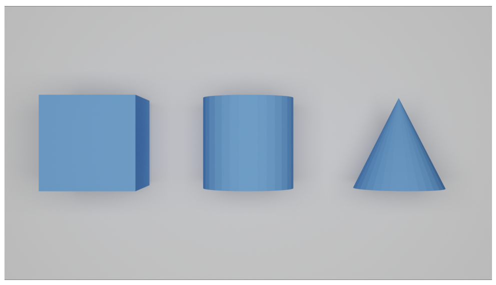
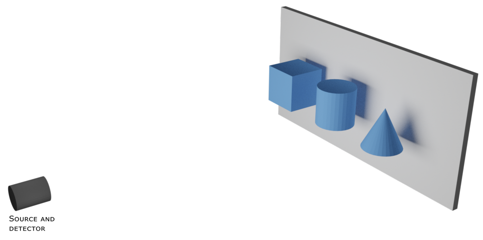


Figure 2.2: Example of a transient vector, where the red element represents the direct component while the green one represents the global component [5]

iToF gives in output an amplitude value A and a phase shifts ϕ for each pixel of the used sensor. From these data, as described in section 2.2.2 it is possible to compute the depth map of the scene by using eq. (2.7). On the other hand dToF for each pixel returns a *transient* vector. This element essentially contains information about how many photons reach back to the sensor in each time interval[5]. From this information, it is immediate to compute the depth map of the scene using eq. (2.9).



(a) Camera view of the scene



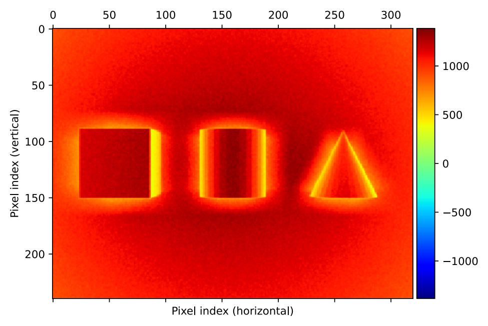
(b) Side view of the scene

Figure 2.3: Overview of the considered NLoS scenarios

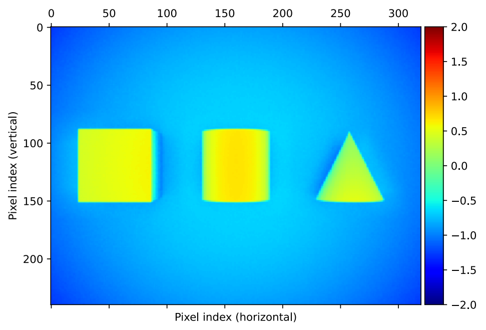
That said to better understand the differences between the two ToF approaches is better to analyze a practical example. So, considering the sample scene of fig. 2.3, in fig. 2.4 it is possible to see the two outputs obtained by using an iToF sensor with a frequency of $20MHz$. While in fig. 2.5 it is possible to see

[5]The number of time instants that the sensor can accurately distinguish determines the accuracy of the sensor.

the output produced by a dToF on the central pixel[6] of the same scene.



(a) Measured amplitude (20MHz) [DN][7]



(b) Measured phi (20MHz) [$radians$]

Figure 2.4: Raw output from an iToF (based on the scene of fig. 2.3)

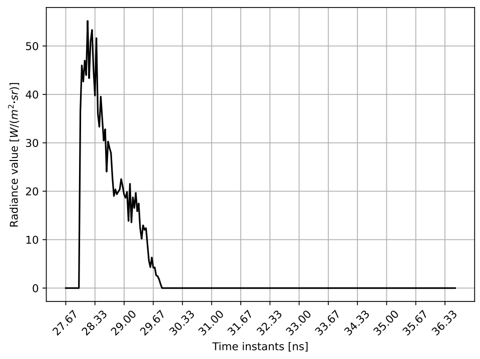


Figure 2.5: Raw output of a the central pixel, from a dToF (based on the scene of fig. 2.3)

From this comparison, it is clear that using an iToF sensor requires a lot more post-processing compared to the one needed using a dToF. At a first glance, this seems to be a huge advantage for the latter but it is also important to factor in the fact that in order to obtain high-quality results from a dToF, it is necessary to use a sensor with really high sensibility. Due to the complexity of the manufacturing process, this kind of sensor is still quite big and very expensive. Other than that it is important to point out that since iToF uses continuous phase its overall resolution is generally higher compared to dToF where the time delay is necessarily quantized. For these reasons, in many applications, the use of an iToF sensor is more suitable.

[6]It is represented only the central pixel because the dToF sensors produce a transient vector for each pixel and so representing the full image in just two dimensions was not possible

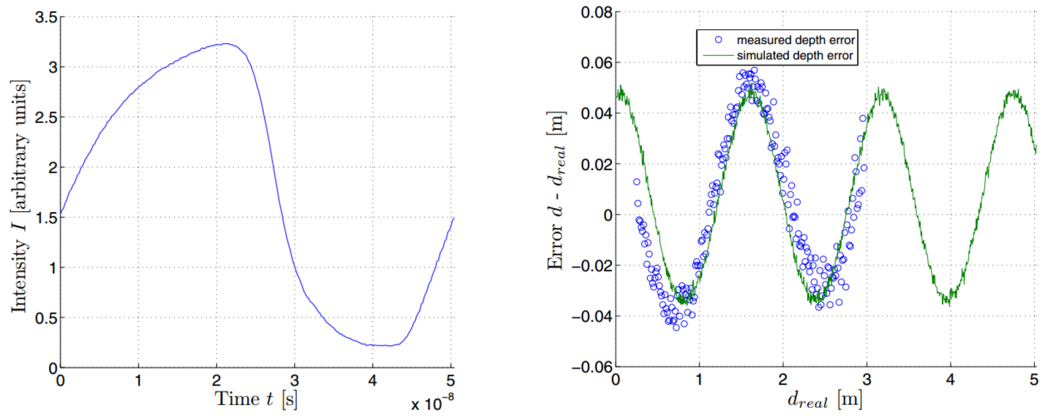
2.2.5 Common error types of a ToF sensor

This section will cover the most common types of error that can perturb a ToF measurement. Since for the rest of the project, it will be used just the iToF technology, the considered errors are mainly related to this specific type of implementation.

Before talking about errors let's talk about how the user can actually, directly influence the obtained measurements. The user has only two options to directly influence the measurement. It can change the integration time, which directly influences the Signal-to-Noise-Ratio (SNR) of the obtained signal and so the variance of the estimated distance. Or it can set different values of f_m in order to define the best compromise between range and noise.

After this small excursus, it is time to cover the most relevant error sources that affect an iToF sensors:

- Systematic distance error,
- Intensity-related distance error,
- Depth inhomogeneity,
- Motion artifacts,
- Multiple returns.



(a) Measured modulation of the Polarized Mode Dispersion (PMD) light source

(b) Mean depth deviation as a function of the real distance

Figure 2.6: Representation of the deviation from the perfect sine function [6]

Systematic distance error

In general systematic errors occur when the formulas used to model the physical imager do not perfectly match reality. In particular, in the iToF cameras there is a consistent difference between the actual modulation and correlation concerning the idealized version used for the calculations. More precisely this mismatch is caused by the presence of higher-order harmonics in the modulating light source that makes the model deviates from a perfect sine function (see fig. 2.6). So when computing the correlation in (2.1) we introduce a "*wiggling*" error which makes the estimated depth oscillate around the real value.

Intensity-related distance error

The measured distance is also greatly affected by the intensity-related distance error. This makes the estimated distance dependent on the total amount of light that reaches the sensor. For this reason, lower reflectivity objects appear closer to the camera (up to $3cm$ of error).

The causes of this problem are not known yet but may be located in some nonlinearity in the photon-to-electron conversion or the semiconductor hardware.

Depth inhomogeneity

The presence of a depth inhomogeneity leads to the so-called *flying pixels* phenomena. In order to visualize this problem, we need to consider a depth boundary with one foreground and one background object. This boundary might be mapped to a single sensor pixel. In this situation, that pixel is hit by a mixture of the background and the foreground light. Due to the nonlinearity in the depth computation and the phase ambiguity, the estimated depth at that point can assume any value of the camera's depth range. This problem occurs quite often due to the low resolution that the ToF sensors usually have.

Motion artifacts

As we have seen in section 2.2.2 the iToF cameras need to sample at least three consequent measurements in order to compute the depth value of each point. Ideally, these three measurements would be acquired simultaneously but, of course, this is not feasible. The non-simultaneity of the measurements makes it so that in dynamic scenes depth is estimated erroneously.

Multiple returns

When it section 2.2.2 it has been described how CWIM model works it has been made a big simplification. It has been assumed that the light returning to every pixel of the sensor, after the reflection on the scene, is coming from a single position in the environment. Unfortunately, this assumption holds only under perfectly ideal conditions so, in almost every real-world scenario, it is violated. This means that usually, multiple returns of light hit every single pixel of the ToF sensor leading to erroneous depth estimation. The most common source of multiple returns is known under the name of *Multi-Path Interference (MPI)*. This situation arises since light can travel multiple paths to intersect the target scene and the imaging pixel.

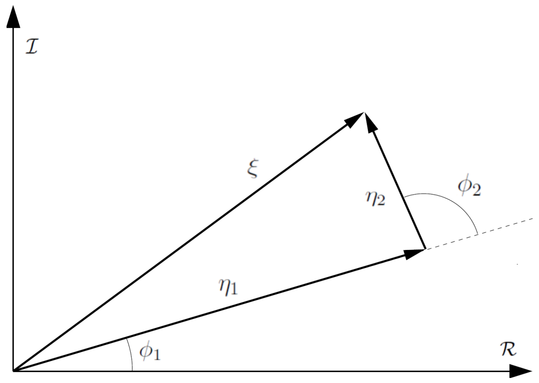


Figure 2.7: Phasor representation of the demodulated light. Representation in the case of only two returns, the primary one η_1 and a secondary one η_2. The resulting measurement corresponds to ξ [6]

In a ToF system the returning light is described by the amplitude A and the phase shifts ϕ. Usually, the demodulated light is commonly modeled as a complex phasor [8]:

$$\eta = Ae^{j\phi}.\ ^{8} \tag{2.10}$$

The above equation describes the situation where it is present a single return, when there are N multiple ones the sensors can measure only the combination

[8] $j = \sqrt{-1}$ is the imaginary part

of them:

$$\xi = \sum_{n=1}^{N} \eta_n = \sum_{n=1}^{N} A e^{j\phi}. \tag{2.11}$$

From all the returning phasor we can say that η_1 corresponds to the primary return, namely the one that corresponds to the ideal path of the light. This phasor usually is the brightest one.

Note that in some cases MPI can also occur intra-camera due to reflection and refraction of the light caused by the lens and the aperture.

Other types of errors

Overall ToF camera sensor is quite similar to the one used in traditional RGB cameras, for this reason, they suffer from the same errors that affect standard sensors. The most significant of them is related to the photon counting process inside the sensor. Since photons are detected only by a certain probability, during the conversion process is introduced some Poisson noise.

In ToF cameras this added noise has a much more severe influence on the output depth. This is because this type of device does not capture all the incoming illumination, but needs to isolate the active illumination from the background one. So any added noise will immediately reduce the final SNR, mainly if we consider that the non-linearity in the depth estimation will further amplify the noise.

2.2.6 Calibration

As it was anticipated in section 2.2 all digital cameras in order to work properly and, so, return a usable output, need to be calibrated before use. This is done through a calibration step. Time of Flight cameras make no exception. The following will briefly introduce how traditional camera calibration works and then it will discuss the calibration procedure for depth cameras.

Geometrical camera calibration process

The calibration process consists on determines a set of parameters (usually used in matrix form) required to retrieve the final output as close as possible to the correct one. It is possible to divide the needed data into two separate sets of parameters:

- INTRINSIC PARAMETERS: represent the camera-specific ones and determine the optical rays in camera-centered coordinates;
- EXTRINSIC PARAMETERS: determine the 3D position and orientation of the camera coordinate system in 3D world coordinates.

It is important to notice that it is extremely hard to extract the intrinsic parameters by simply inspecting the optical system. For this reason, it is necessary to estimate the extrinsic and intrinsic parameters altogether.

In order to calibrate a camera, it is necessary to use a planar 2D object that determines the world coordinates system (x, y coordinates span the plane while z coordinate spans the plane's normal). Then it is required to take a set of different calibration images while moving and tilting the calibration plane. Each image needs to correspond to a different camera pose (= different extrinsic parameters), but all the acquired pictures during the calibration procedure must link to the same intrinsic parameters. This aspect helps to disambiguate highly correlated data like the distance z and the focal length f. During the calibration, it is important to also take into account the Field of View (FoV) of the used camera since a narrow FoV brings a high correlation between extrinsic position and orientation and so to less precise calibration data.

Depth camera calibration process

From the brief introduction to calibration given in section 2.2.6, it is clear that calibrating a Time of Flight camera is a hard task. This is mainly due to three aspects:

- ToF sensors are forced by construction to have a small FoV in order to allow the emitted InfraRed (IR) light to reach the target with enough intensity;
- they are limited to really low image resolution smaller than half Video Graphics Array (VGA)[9];
- they do not capture the direct image, and so only the reflectance image can be used for calibration purposes.

The only benefit while calibrating a ToF camera is that, since it measures depth, it is possible to reliably disambiguate z from f and avoid the scale factor.

One of the best approaches to perform ToF camera calibration is to couple it in a rigid rig together with a traditional RGB camera and calibrate both devices

[9]VGA maximum resolution is 640×480 $[px]$

at the same time [6]. This helps to utilize the strength of each architecture compensating for their weaknesses. Keep in mind that since ToF sensors measure the time of flight along the light path, error calibration is better to be performed considering the radial distance.

2.2.7 Post-processing depth correction

Unfortunately not all the errors could be fixed during calibration, some of them are scene dependent and require a post-processing step applied after the depth measurement.
The following three errors fall into this category:

- flying pixels,
- motion artifacts,
- Multi-Path Interference.

In the following will be presented some state-of-the-art approaches to mitigate these issues.

Depth inhomogeneity

In order to reduce the flying pixels problem, it is possible to follow two different approaches, work directly on the 2D output of the ToF camera or work on the reconstructed 3D scene.

A rough and straightforward way to fight flying pixels is to apply a median filter to the 2D output and also a denoising pipeline [6]. But, in order to obtain a better, more accurate, and refined result it is necessary to utilize a more complex approach. The standard one is to identify the incriminated pixels (e.g., by confidence measures) and discard them. After that, the depth value of the removed pixels is estimated using information coming from the surrounding ones.

To further improve the flying pixels removal process it is possible to consider also the 3D geometrical data during the depth reconstruction of the missing pixel.

Motion compensation

Another issue affecting ToF cameras is represented by motion artifacts that mainly happen when the sensor is acquiring dynamic scenes. As it has already been explained in section 2.2.5 this artifact is a consequence of the non-simultaneity of the sequential measurements.

There are different ways to mitigate this type of issue. The simplest one consists of reducing the number of frames sequentially captured in order to produce a valid depth reconstruction. A more refined approach consists of, firstly detecting where erroneous regions (due to motion) are located and, only after that, they will be selectively corrected.

There is also an alternative approach to remove motion artifacts that directly estimate scene motion between sub-frames using *optical flow*.

Multipath correction

As it was discussed in section 2.2.5 the Multi-Path Interference (MPI) represent one of the biggest problems of ToF sensors and so contrasting it is an essential step. Fixing the MPI essentially consists of separating the measured complex phasor (ξ) into its main components (usually the most interesting one is the principal one, η_1). This is an underdetermined problem since it is available for only a single measurement but it is necessary to extract multiple elements. So in order to complete the task, it is required to have some kind of additional information, like some *a priori* one or the one coming from multiple measurements.

2.3 Mapping between direct Time of Flight and indirect Time of Flight

As described in section 2.2.5, one of the biggest problems that affect the quality and the precision of the iToF output is the MPI. Essentially the emitted light will not perform a single reflection back to the sensor following a unique path. It will produce multiple reflections (both inside and outside the camera) and will reach the sensor at slightly different time instants, following multiple directions. This, often lead to erroneous depth estimation. In the iToF implementation the incoming light is represented as a phasor 2.10 and when multiple ones hit the sensor they will be combined in a single phasor that is different from the one of the direct reflection 2.11.

On the other side a dToF, for how it is built, it does not merge in a single phasor, the incoming light but, instead captures the intensity of light arriving at the sensor at each time instant. In this way isolating the different contributions due to the different times of arrival become quite easy. More precisely, using this type of camera it is possible to record the *transient vector* (an example could be seen in fig. 2.8) that allows avoiding any interference-related issues and at the same time provides additional insights on the scene geometry.

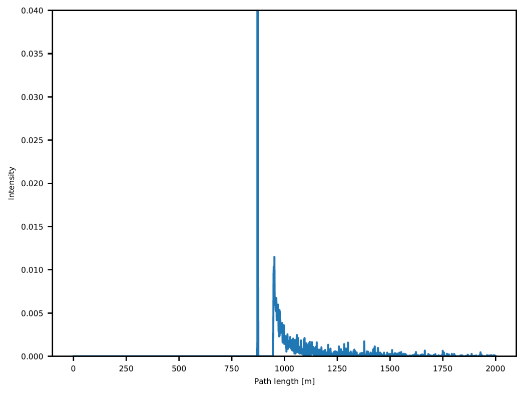


Figure 2.8: Example of a backscattering vector for a corner scene [9]

Due to the high cost and the reduced spatial resolution, dToF often are not the best choice. For this reason, could be useful a method able to convert an iToF output to a dToF one with minimum error. In this way, it will be possible to use a cheaper iToF to perform acquisition usually restrained only to dToF.

It is important to notice that the conversion from dToF to iToF is straightforward (use a lossless analytical model), while the opposite is much more complicated. In this work, we will consider the novel approach proposed in [5].

2.3.1 dToF to iToF

Converting the output from a dToF to the correspondent one of an iToF is quite a straightforward approach since it is possible to do it using a perfect analytical conversion process.

To do so it is necessary to consider the iToF measurements c_τ in its phasor notation as presented in [8]

$$c_\tau = Ae^{j\phi} = Ae^{j2\pi f_m \Delta t} \in \mathbb{C}^{10}, \tag{2.12}$$

where Δt is the Round Trip Time (RTT) of the light signal. This formulation can also be used to mathematically describe the MPI phenomena. Under this condition (common in real-world measurements), as described in section 2.2.5 the sensor will receive and integrate multiple signals coming from different and commonly longer paths. After this consideration, it is possible to rewrite eq. (2.12) as:

$$c_\tau = \int_{t_{min}}^{t_{max}} x(t)e^{j2\pi f_m t} dt, \tag{2.13}$$

where t_{min} and t_{max} represent respectively the minimum and the maximum time of flights considered, while $x(t)$ corresponds to the transient. Equation (2.13) could also be rewritten in its discretized version:

$$c_\tau = \sum_{t=t_{min}}^{t_{max}} x(t)e^{j2\pi f_m t}. \tag{2.14}$$

If we consider that in a standard iToF measurement multiple acquisition frequencies are used in order to obtain a reliable result it is possible to rewrite eq. (2.14) as:

$$\boldsymbol{c} = \boldsymbol{\Phi x}, \tag{2.15}$$

where $\boldsymbol{c}$ are the iToF measurements at multiple frequencies, $\boldsymbol{x}$ is the transient vector and $\boldsymbol{\Phi}$ is the measurement model of the iToF.

At this point, by a simple matrix multiplication, it is possible to map a dToF measurement $\boldsymbol{x}$ to the correspondent iToF one, $\boldsymbol{c}$ following eq. (2.15).

2.3.2 iToF to dToF

The conversion from iToF to dToF can be done using a linear model but it requires a complex acquisition process and, in any case, give poor results. The biggest problem is linked to the matrix $\boldsymbol{\Phi}$ that unfortunately is rectangular and non-invertible. This makes inverting eq. (2.15) impossible. So to perform this

[10]Note that to use this notation it is necessary to take aside the intensity component

conversion seems to be to use a deep learning approach.

The authors of [5] have proposed a very compact architecture exploiting the direct-global subdivision of transient information for the reconstruction of the transient information itself[11] starting from raw iToF data. More precisely they develop a deep learning architecture able to reduce the MPI from iToF data and also recover an approximation of the transient vector.

Implementation details

Before talking about the actual implementation strategy let's recall that a transient vector is composed of two distinct parts, the *direct component* x_d and the *global component* x_g. So, given a set of iToF measurements at different frequencies v we can decompose it using the scene impulse response x and the measurement model $\mathbf{\Phi}$ as follow:

$$v = x\mathbf{\Phi}. \tag{2.16}$$

We can now use the linearity of the model to further decompose (2.16) into:

$$v = x\mathbf{\Phi} = \mathbf{\Phi}(x_d + x_g) = v_d + v_g \tag{2.17}$$

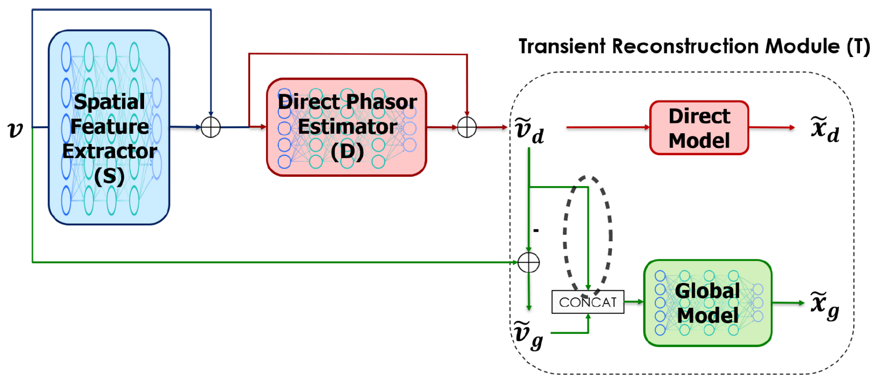
where v_d corresponds to the ideal iToF measurements, while v_g corresponds to the measurements of all the other reflections.

As anticipated in section 2.3.2 the proposed approach is completely based on the use of a deep learning model that takes into input the real and imaginary part of the raw iToF measurements v at different modulation frequencies. In particular, the designed model, as shown in fig. 2.9, is composed of three parts:

1. Spatial Feature Extractor (S) that aims to deal with temporal noise sources with zero mean (e.g., shot noise) and provide an encoded version of the spatial information to the following steps;
2. Direct Phasor Estimator (D) which perform the MPI denoising estimating v_d, the direct component of the raw phasor;
3. Transient Reconstruction Module (T) that has to reconstruct the transient representation (x_d and x_g), in order to do so it is composed by two sub-modules:

[11] The transient information other than error removal can also be used for NLoS perception or material recognition

- Direct model that reconstructs only the direct component $\boldsymbol{x}_d$ returning its magnitude E_d and its time position t_d;
- Global model which reconstructs only the global model $\boldsymbol{x}_g$ approximating it using a Weibull distribution:
$$\widetilde{\boldsymbol{x}}_g(t) = \boldsymbol{a}(t - \boldsymbol{b})^{\boldsymbol{k}-1} exp\left(-\frac{t - \boldsymbol{b}}{\boldsymbol{\lambda}}\right)^{\boldsymbol{k}} \tag{2.18}$$
where t ranges from 0 to T (maximum acceptable travel time), $\boldsymbol{a}$ takes care of the scale, $\boldsymbol{b}$ of the shift, $\boldsymbol{k}$ and $\boldsymbol{\lambda}$ of the shape.



$\boldsymbol{v}$
Spatial Feature Extractor (S)
Direct Phasor Estimator (D)
Transient Reconstruction Module (T)
$\widetilde{\boldsymbol{v}}_d$
Direct Model
$\widetilde{\boldsymbol{x}}_d$
-
CONCAT
Global Model
$\widetilde{\boldsymbol{x}}_g$
$\widetilde{\boldsymbol{v}}_g$

Figure 2.9: High level structure of the training architecture

Note that from the above-mentioned model it is possible to extract two different architectures:

- *SD* which corresponds to the full model and also to the best performing one, both on synthetic and real data;
- *D* which is an extremely lightweight architecture (only $3k$ parameters) composed by only the modules *D* and *T*, but still able to reach good performance nevertheless worse than the complete counterpart.

Regarding the loss function used to train the model, a Mean Absolute Error (MAE) has been used to guide the *Direct Phasor Estimator* while the *Global model* has been guided by an Earth Mover's Distance (EMD) one.

Performance of the model

Evaluating the performance of *SD* in the task of transient reconstruction it is possible to say that, as we can see in fig. 2.10, it can define the direct component quite good while it has some more problems in reconstructing the global component but it is still able to give a good general idea of it. Other than that it is important to notice that this transient reconstruction is much better than the one achieved by the only other approach, represented by iToF2dToF [10].

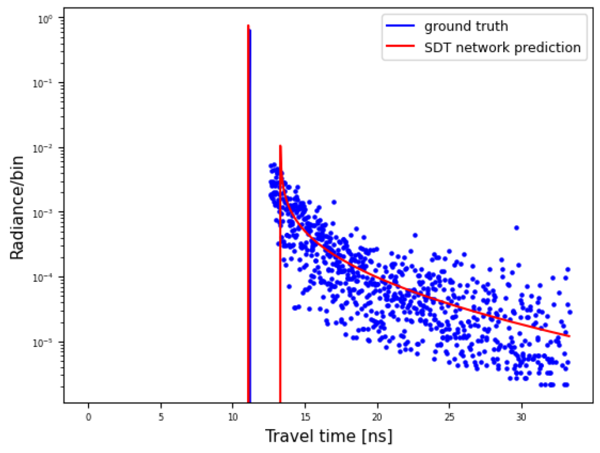


Figure 2.10: Example of transient vector reconstructed using the *SD* architecture (compared to the ground truth)

2.4 Non Line of Sight perception using Time of Flight

Regarding the task of Non-Line-of-Sight imaging using a ToF sensor, the general idea is to consider the output of a dToF and from that extract the *global component* x_g ignoring the direct one. This approach is justified by the fact that, if the goal is to extract information coming from an object located in NLoS for sure that element will not be the closest one to the sensor. For that reason, it is impossible that it ends up inside the *direct component* but it will be part of x_g.

From this general introduction it is clear that NLoS using ToF is limited to work only on two specific scenarios: look around the corner and look through a diffuser. This constraint is dictated by the fact that in order to gather information about the hidden object, the dToF requires that some reflection from that object came back to it. This excludes the "look behind a wall" scenario since in this situation no reflection from the hidden element may reach the sensor. For what concerns this specific project the goal will be to implement a NLoS imaging pipeline to retrieve information about an object located around a corner.

It is clear that to get some information from a NLoS scene it is essential for the direct/global separation of the ToF measurements. For this reason, in many cases, the most common ToF sensor used for this kind of application is the dToF from which it is easy to separately analyze the two components. In any case, also starting from x_g directly doesn't make the task easy. Indeed from the global component is still extremely difficult to isolate the information coming from the hidden object. Due to the extremely high complexity of this task, as stated in [2], at the moment most of the state-of-the-art approaches relay the extraction of useful data from x_g to a NN or Deep Learning (DL) model.

For that reason also the approach that will be proposed in this work will hand off the point cloud extraction process to a NN model.

3

Transient ray-tracing using Mitsuba Renderer 2

3.1 Introduction

As anticipated in section 2.4 to perform NLoS imaging the plan is to use a NN model. That directly implied the need for a related dataset on which it is possible to train the network. Since the considered topic is extremely difficult the best way to start approaching it is to first consider a simplified situation, and only in a later stage add more and more up until a real-world situation is reached. This work represents the beginning of the project, so it has been decided to start the training of the NN model on an ideal synthetic dataset without any error, apart from the MPI one. Unfortunately, in the literature, does not exist any pre-build synthetic dataset suited for the intended task. This made the need to generate a new dataset designed ad-hoc for this project.

The precise structure of the dataset will be described in detail in chapter 4. In any case, to build a dataset that simulates dToF measurement of a 3D scene representing a "*look around the corner*" scenario (same setup of fig. 1.1a) it is required to first build the 3D scenes and then perform the "measurements" to obtain the transient information. To accomplish such a task it is required the use of a rendering system that both builds the scene and also acquires the transient data. In order to do so it is necessary to use a rendering environment able to fulfill four essential requirements:

- being able to render the 3D scene with different camera/object poses and materials,
- render everything following the law of physics as close as possible,
- allows for some kind of automation in order to speed up the generation process,
- being able to estimate realistic transient information from the considered scene.

After some evaluations, it has been decided that the best solution for our needs was `Mituba Renderer 2` [11].

3.1.1 Mitsuba Renderer 2

`Mitsuba 2` is an open-source research-oriented rendering system written in C++. More precisely it consists of a set of libraries and plugins that implement a wide range of different functionalities, ranging from material and light sources to complete rendering algorithms. The goal of `Mitsuba 2` is to provide researchers in the advanced sensing and Computer Vision fields with relatively easy and lightweight renderer environments deeply focused on realism. More precisely the goal of the authors was to develop a complete solution that generates realistic renderings using physical light transport simulations and at the same time keeps the overall complexity and computational load at an acceptable level. In this way, researchers can focus on their project without losing time on implementing the rendering pipeline.

For our specific use, these premises was perfect, since our goal is to focus most of our efforts on the NLoS detection instead that on the rendering of the dataset. Other than that, this specific renderer fulfills three out of the four mandatory requirements presented in section 3.1. Regarding the requirement of the transient computation, the standard version of `Mitsuba 2` was not able to perform such a task. Thankfully there exist two forks of the renderer, `mitsuba2-transient` [12] and `mitsuba2-transient-nlos` [13] that add this exact functionality to the core version of the software.

3.2 Mitsuba2-transient and mitsuba2-transient-nlos

Both these two forks of the core `Mitsuba 2` aim to add the functionality of transient vector estimation from a given 3D scene. `Mitsuba2-transient-nlos` also add the possibility to perform some NLoS acquisitions. More precisely the latter is based on the first one, it represents a follow-up of the original project started by Jorge Garcia Pueyo. For that reason, both of them share the same core modifications that allow the implementation of transient rendering but `mitsuba2-transient-nlos` contains some additional features and many bug fixes. The main contribution of [12] is the introduction of two novel plugins, `transientpath` and `streakhdrfilm` that utilize the functionalities of the core `Mitsuba 2` to render also the transient data.

It is important to briefly explain how these `Mitsuba 2` forks build the dToF like reconstruction. In order to obtain the desired output, it is necessary to first design the 3D scene, defining properties and position/orientation for each object, camera, and light source of the scene. In our specific case, this step was performed using `Blender 2.83`. Once the scene is fully defined using the `Mitsuba Blender Add-on` [14] it is possible to directly export it in the format required by the renderer (general information inside an `.xml` file and all the meshes as `.obj` files). Since this plugin is not compatible with the modified version of `Mitsuba` it is necessary to perform some easy modifications to the `.xml` file in order to make it compatible with `streakhdrfilm`. Once all of these steps are concluded simply running `Mitsuba` on the final file it is possible to obtain the transient information related to the given scene.

The obtained output requires some post-processing since it is not formatted in a standard and easy-to-use form. More precisely for each scene `Mitsuba` return a list of `.exr`[1] images. Each one of them stores the transient information of each row of pixels of the considered sensor. Starting from This kind of output it is necessary to load all the `.exr` in order to store them in a more easy-to-use format, and more importantly, reshape them to fit inside a single file containing the transient information pixel by pixel instead of row by row. In our specific case, the data were converted from `.exr` to a `numpy` array.

[1]EXR is a raster image stored in the OpenEXR format, that is an High Dynamic-Range (HDR) image format [15], essential to store all the information relative to the scene without losing any data due to compression

3.3 In depth tests on the various version of Mitsuba Renderer 2

Since the considered forks of `Mitsuba 2` presented in section 3.2 are not officially supported by the `Mitsuba` team and represent personal projects or thesis projects it was essential to verify their accuracy as the first step of the project. This represented a fundamental step in the development of our work since all the following will depends on the dataset generated using `Mitsuba`, so any error of the renderer will be reflected in all the following stages and possibly could become worst at every step.

Originally the idea was to use as render environment `mitsuba2-transient` as it just adds to the core version only the transient reconstruction without any additional features. In this way, the possibility of having errors caused by additional features should be minimized. For that reason, the validation phase was designed to test this specific version. The renderer was compiled and built (following the official guide) on `Ubuntu 20.04.4 LTS` as it is the recommended Operating System (OS) by the author of [12]. In order to validate all the key aspects of the renderer, it was decided to perform the following four tests:

- check if the distance decay follows the expected law,
- understand if some temporal bin quantization is applied,
- check if the cross-section decay follows the expected law,
- compares the transient reconstruction with the RGB render.

Note that for all the following tests the resolution of the sensor was set to be VGA, $640 \times 480\ px$.

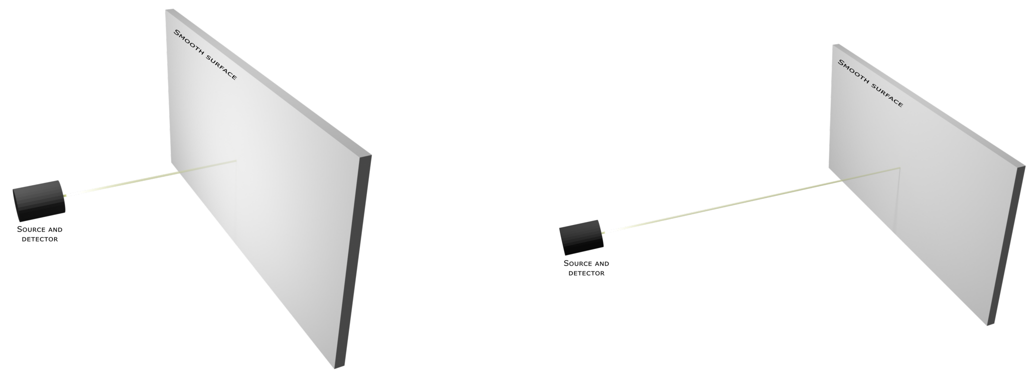
3.3.1 Testing of mitsuba2-transient

Distance decay

If the transient computation is implemented correctly it must hold that, illuminating a smooth and flat surface located perpendicularly to the sensor, the radiance measured on the central pixel (the one directly in front of the sensor) decays as:

$$\frac{1}{d^2} \cdot r_{max}, \tag{3.1}$$

where d is the distance between the sensor and the surface, while r_{max} is the maximum measured radiance value.



(a) Setup with $d = 2m$ (b) Setup with $d = 4m$

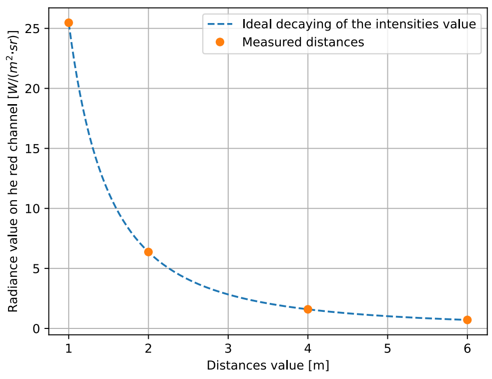
Figure 3.1: Example of two test scenes for the distance decay validation

In order to validate this aspect, we designed a testing procedure consisting of multiple consecutive acquisitions. First of all, it was necessary to design the scenes used to get the measurements. The idea was to use a big white surface[2] located in a fixed position in front of the sensor and perform several measurements each time moving the sensor far away from the surface. In fig. 3.1 it is possible to see two sample scenes.

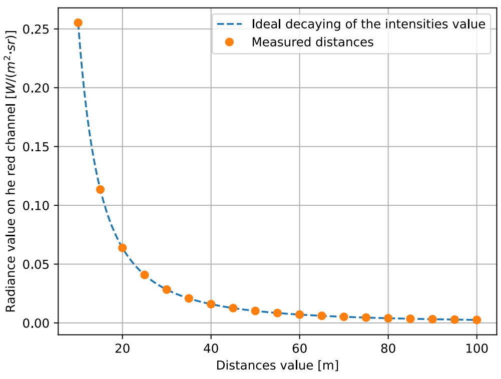
The final evaluation was performed on a total of 23 different scenes split into two groups: the first group considers $d \in [1,\ 6]m$ with a step of $1m$, while for the second one, $d \in [5,\ 100]m$ with a step of $5m$. The test was not performed in a single group with the same step size due to a bug of `Mitsuba` that has bad memory management and consequently, there are some limitations on the value of the number of temporal bins used related to their size. Since to reach $d = 100m$ it was necessary to increase the number of temporal bins used by the sensor, in order to avoid the memory problem it was also necessary to increase the bin size, forcing us to split the test into two groups. For the first group, the bin size was set to $1cm$ while for the second one to $20cm$.

As it is possible to see in fig. 3.2 it is clear that `mitsuba2-transient` passes

[2]This surface should be completely diffuse, perfectly white and must cover the complete FoV of the sensor in order to avoid any kind of alteration



(a) Setup with $d \in [1,\ 6]$



(b) Setup with $d \in [5,\ 100]$

Figure 3.2: Distance decay test results

all the tests regarding the decay of the radiance relative to the distance.

Temporal bin quantization

During the testing phase presented in the previous section, we noticed a constant offset in the measured distances compared to the expected (real) value. This issue could be a consequence of a quantization performed by the back-end of `Mitsuba` that groups all the ray falling into the same bin and give them the same value. In order to verify this hypothesis we considered the same scene setup used before but this time d varies in $[1,\ 30]mm$. Other than that in this case the sensibility of the sensor is set to $1cm$[3], in order to put the renderer in the worst-case scenario.

As we can see in fig. 3.3 `Mitsuba 2` actually perform quantization to the data, and to be precise perform a *ceil*. This is not a big deal since, knowing the issue, it is simply necessary to compute the constant offset (only dependent on the bin size) and subtract it directly from the computed distance values (computed following eq. (2.9) starting from the raw measurements).

Cross-section decay

In order to be sure that the renderer is accurate it is not enough to check that the radiance decay as $\frac{1}{d^2}$. It is also essential, as can be evinced by the theory

[3]The sensibility of the sensor is determined by the bin size used

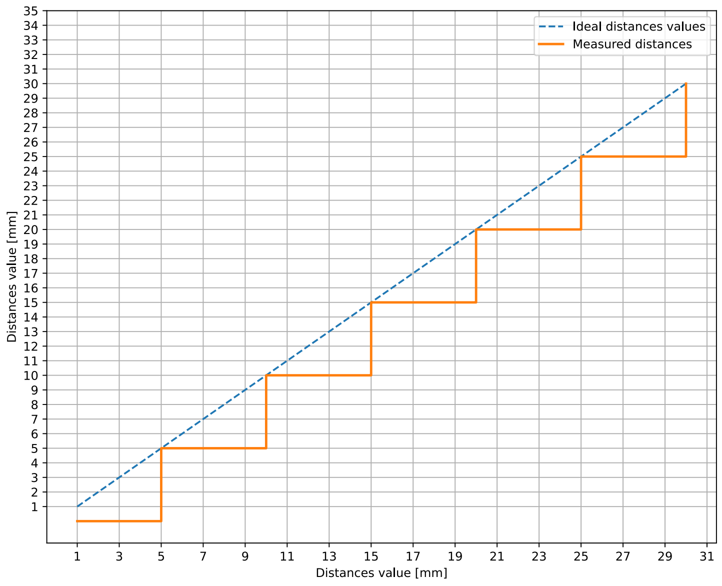


Figure 3.3: Quantization test results

presented in section 2.2.2, to verify that on the cross-section[4] of the FoV of the sensor the radiance decay as

$$cos^3(\theta) \cdot r_{max}, \tag{3.2}$$

where θ represents the angle between the optical axes[5] and the observed pixel.

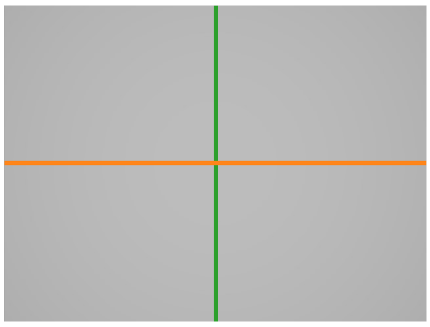


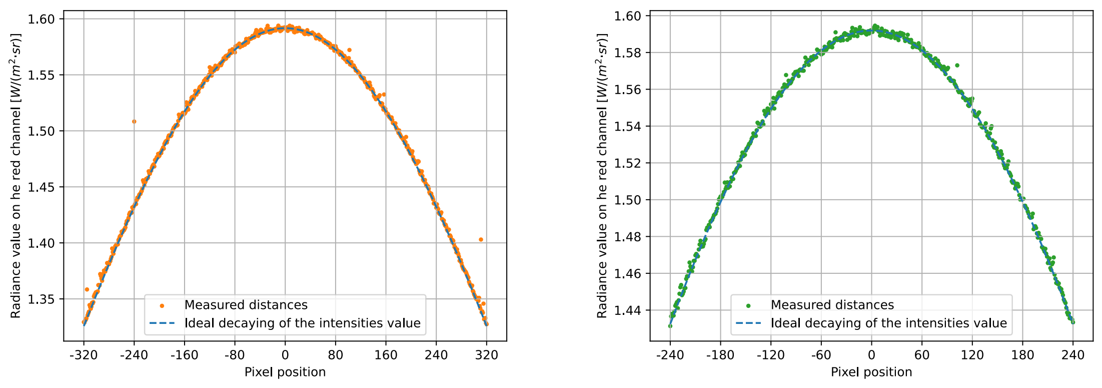
Figure 3.4: Representation of the cross-section over the FoV of the sensor

To validate the accuracy of `Mitsuba` also on this aspect we used the same setup presented in fig. 3.1, but this time d was fixed to $4m$.

From fig. 3.5 it is possible to verify that, apart from few outliers, the renderer

[4]The cross-section corresponds to all the sensor pixels located on the principal row and column

[5]The optical axes is an imaginary line passing through the sensor and perpendicular to it



(a) Principal row decay (b) Principal column decay

Figure 3.5: Cross section decay test results

follows the law of eq. (3.2). So also this test was a success.

Comparison of the transient reconstruction

Another essential check that must be performed consists of verifying that the standard RGB image of the scene is equal to the one obtained by summing the transient data (pixel by pixel), over the time dimension and normalized by the total number of bins, of the same image. If the two images don't match it means that there is some problem in the handling of the transient information. Since the considered scene is the same the final result must be also the same.

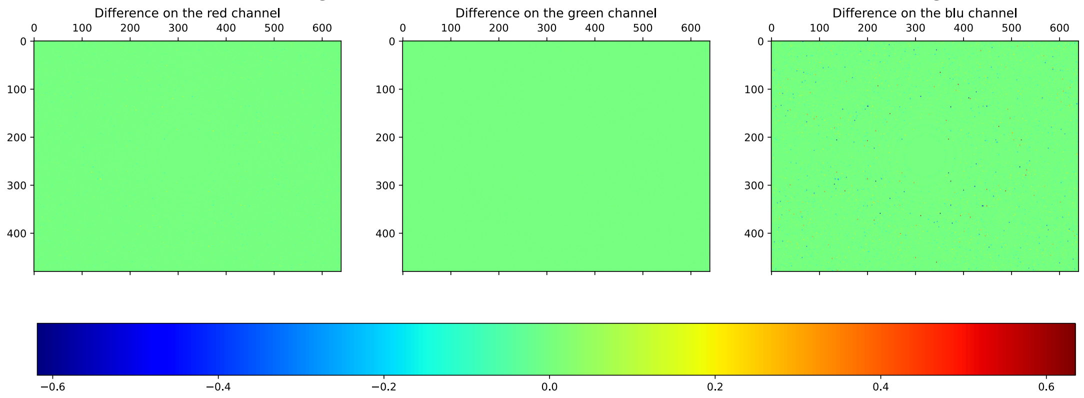
To test this aspect we have designed a test performed on two different scenes in order to be sure that the result is scene-independent. The first, and also the simplest, setup was based on the one of fig. 3.1, so the sensor was looking directly towards a white wall. For the second scene, we complicated a bit the environment making the sensor look at a Cornell Box[6].

As regards the white wall scene it is possible to see from fig. 3.6 that the two different renders are essentially the same both visually and quantitatively (fig. 3.6c). The only thing to point out is that on the blue channel there are some outliers. This small inconsistency could be due to the rendering noise, and for this reason, can be ignored. This scene is really simple since there is only one big surface in front of the sensor there are no indirect reflections. The only reflection reaching the sensor is the direct one coming from the wall. For this reason, the transient vector of each pixel is characterized by a single peak (e.g., fig. 3.7), the

[6]The cornel box scene used is the `cbox-point-light` from [16]

(a) RGB image

(b) Transient image



(c) Differnece between the two images

Figure 3.6: Comparison between the standard RGB render and the one obtained by the transient of the white wall scene

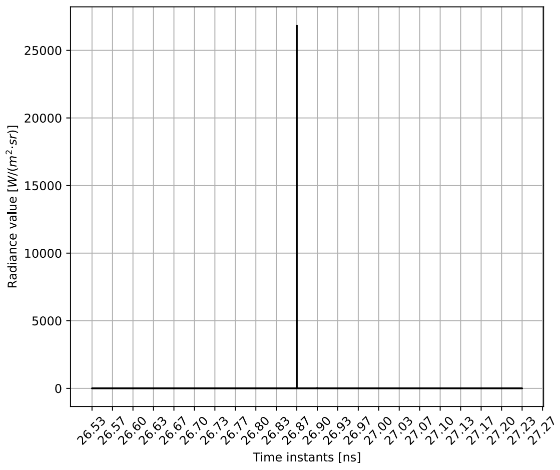
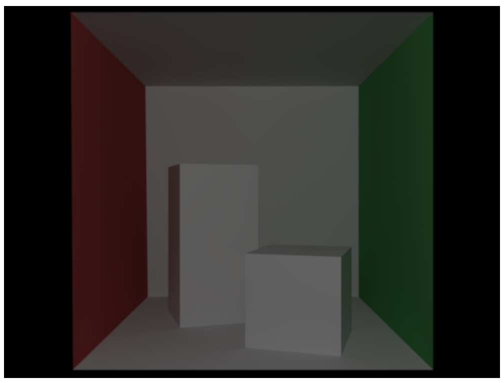
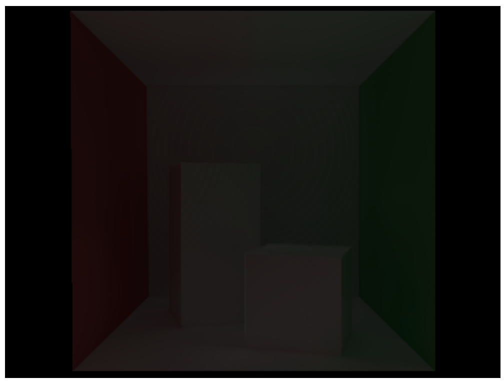


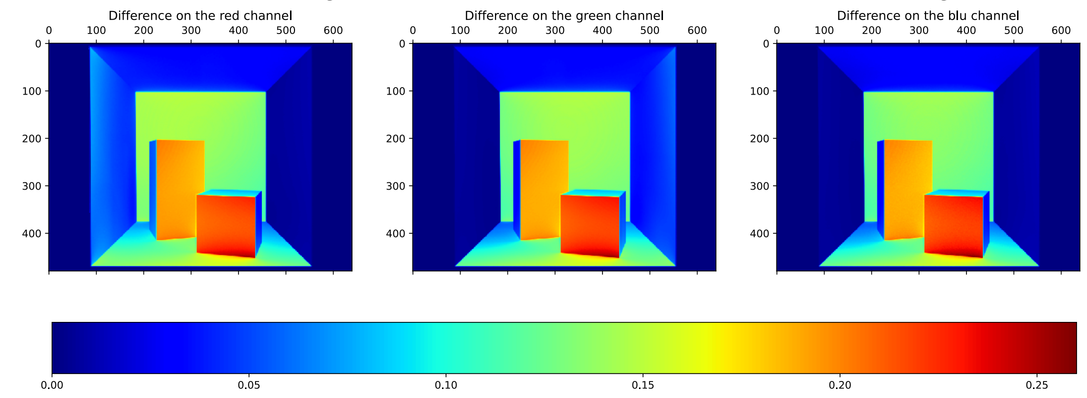
Figure 3.7: Representation of the transient vector of the central pixel (white wall)

one from the wall.

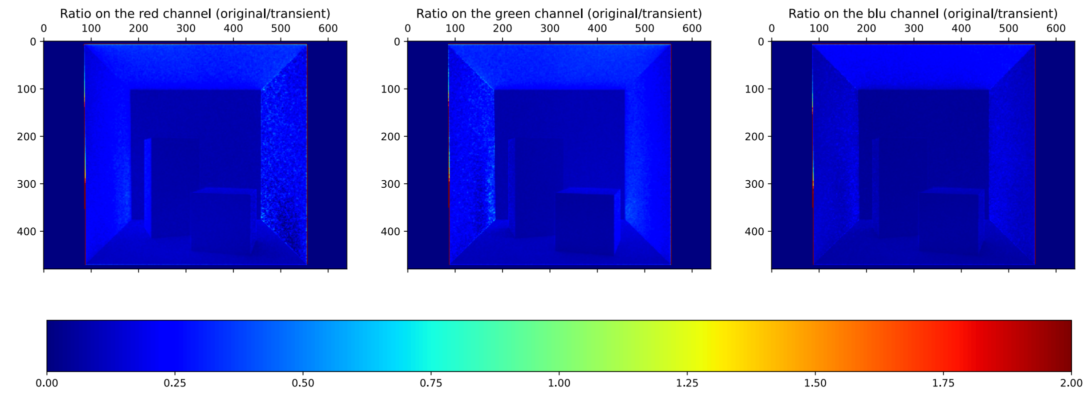
To be sure that the just tested scenario wasn't a success only because there were no multiple reflections and so, no global component, the same test was


(a) RGB image


(b) Transient image



(c) Differnece between the two images



(d) Ratio between the two images

Figure 3.8: Comparison between the standard RGB render and the one obtained by the transient of the Cornell box scene

performed also on a Cornell box where the light source was colocated with the sensor (instead of putting it on the ceil of the box as standard). This time, as we can see from fig. 3.8, the two produced pictures, visually look completely different. Analyzing the differences, channel by channel (fig. 3.8c), it seems that the two reconstructions differ only by a scaling factor. This behavior could be justified if the renderer, while estimating the transient vector, performs a normalization different from the standard one[7]. At that point, it would be simply necessary to rescale the image produced using the transient information using the correct scale factor. To check if this could be the case we look at the implementation of `mitsuba2-transient` to find which kind of normalization has been used. Unfortunately, it seems that the implementation of the transient estimator uses the expected normalization. This means that there is some kind of bug in the renderer that ruins the estimation. Indeed, looking at fig. 3.8d, it is possible to notice that also the ratio between the two images is not constant, so a scaling factor will not fix the issue.

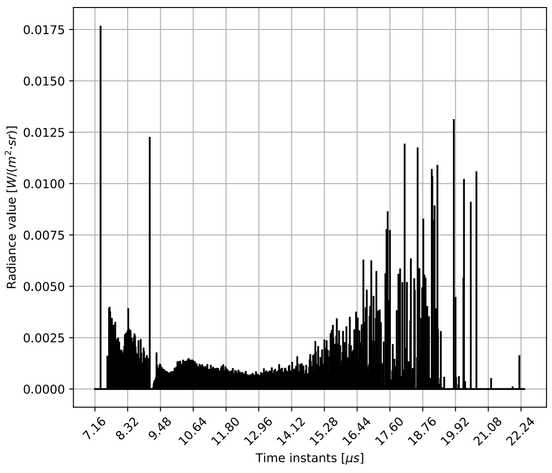


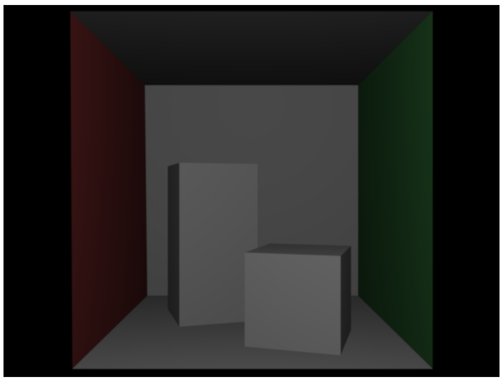
Figure 3.9: Representation of the transient vector of the central pixel (Cornell box)

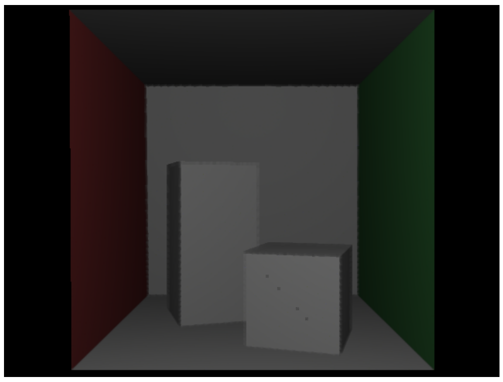
The fact that this specific fork of `Mitsuba` has a problem in handling transient information is even more clear observing fig. 3.9. Given the structure of the scene, the expected transient vector shape should be characterized by huge peaks in the beginning (due to the direct reflections) followed by smaller and smaller peaks generated by undirect and multiple reflections that are for sure weaker. Despite that, the produced transient is completely different from the expectation since the tail, instead of decreasing is increasing over time.

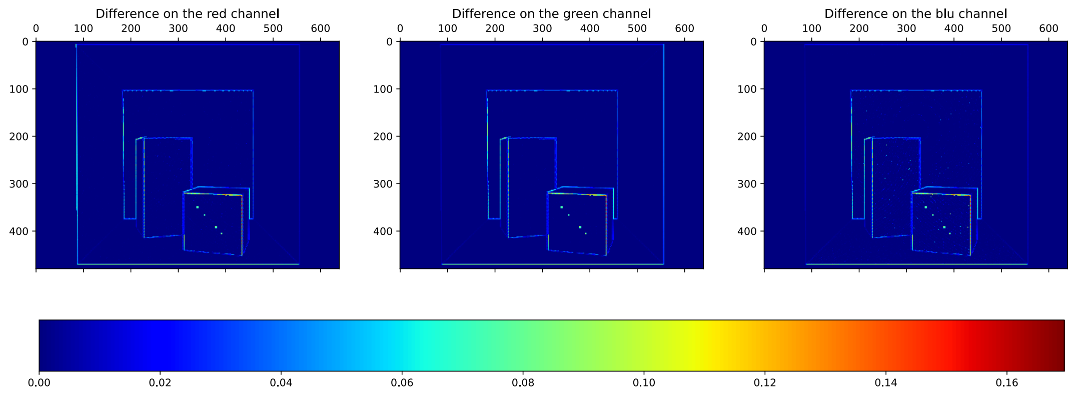
To ensure that the described problem is generated by the indirect reflection we re-rendered the same Cornel box scene forcing `Mitsuba` to consider only direct reflection. As it is possible to see in fig. 3.10 in this case the problem is

[7]The standard normalization in this situation correspond to sum all the incoming beam of light inside the same temporal bin and dividing it by the number of rays

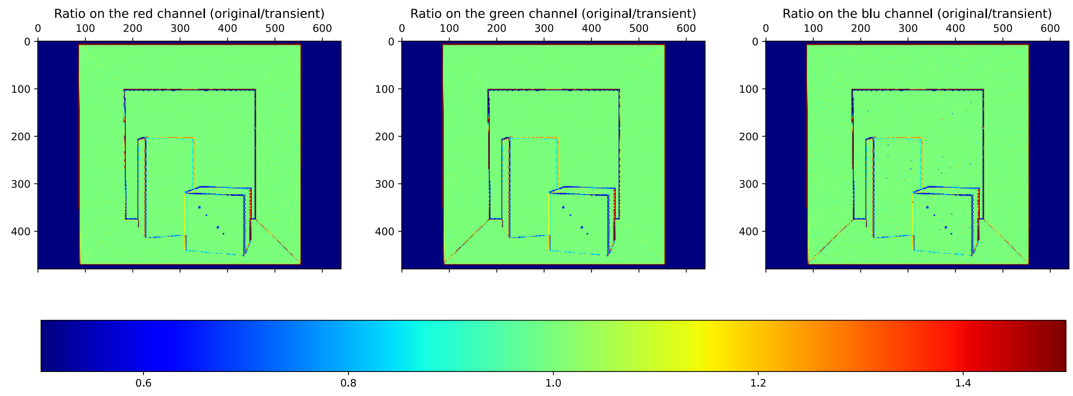
much less severe but on the edges of the object persist a lot of inconsistencies.


(a) RGB image


(b) Transient image



(c) Differnece between the two images



(d) Ratio between the two images

Figure 3.10: Comparison between the standard RGB render and the one obtained by the transient of the Cornell box (single reflection) scene

3.3.2 Testing of mitsuba2-transient-nlos

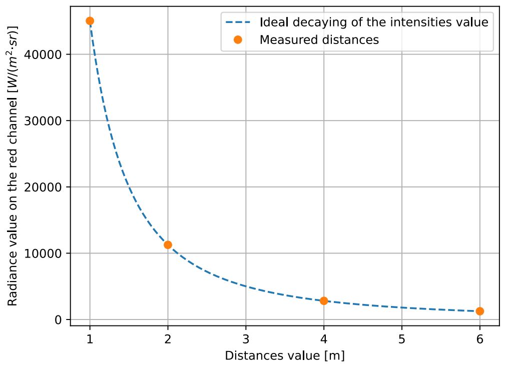
After the analysis performed in section 3.3.1 it was clear that `mitsuba2--transient` suffers from severe issues related to transient computation. For that reason has been decided to switch to its parallel fork `mitsuba2-transient-nlos` [13]. This alternative version is based on the previous one but aims to have fewer bugs and some additional functionalities regarding NLoS imaging (none of these additional features are useful for the scope of this project[8]).

Of course before definitively switching it is mandatory to perform again the same tests performed in section 3.3.1 in order to ensure that all the errors of `mitsuba2-transient` have been fixed.

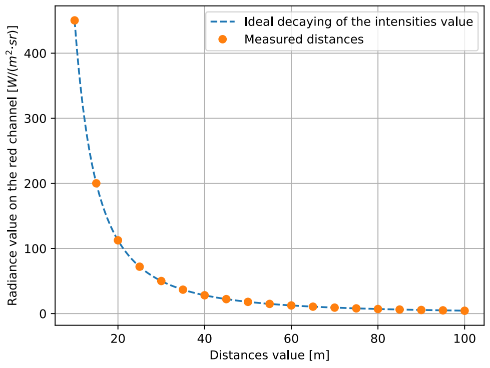
Before starting the testing session it is important to point out that this version of the renderer fixes the memory issue that limits the maximum number of temporal bins given their size. But, unfortunately, introduce a new bug that limits the sensor resolution to be at maximum Quarter Video Graphics Array (QVGA)[9], so $320 \times 240\ px$. If a greater resolution is used half of the produced transient information results corrupted and completely unusable. For this reason, all the renders that will be produced from now on in this paper are limited to QVGA, in order to avoid corruption of the output.

Distance decay

As already stated in the previous section measuring the transient in a scene like the one in fig. 3.1 the radiance values should follow eq. (3.1).



(a) Setup with $d \in [1,\ 6]$



(b) Setup with $d \in [5,\ 100]$

Figure 3.11: Distance decay test results

Looking at fig. 3.11 it is clear that as for `mitsuba2-transient` also `mitsuba-`

[8]The additional features regarding NLoS imaging are not useful since the method is not fully implemented and is based on a different approach from the one that we plan to use

[9]This limitation is only related to transient rendering, the standard RGB render (handled by standard `Mitsuba 2` function) works fine at every resolution

`2-transient-nlos` perfectly follows this law.

Temporal bin quantization

Since the core implementation of the transient estimation procedure (`transientpath` and `streakhdrfilm`) are substantially the same as the one used in the former version `mitsuba2-transient-nlos` it is expected that the quantization process is exactly the same. Indeed as it is possible to see in fig. 3.12 also in this case the output is preceded by a *ceil* function.

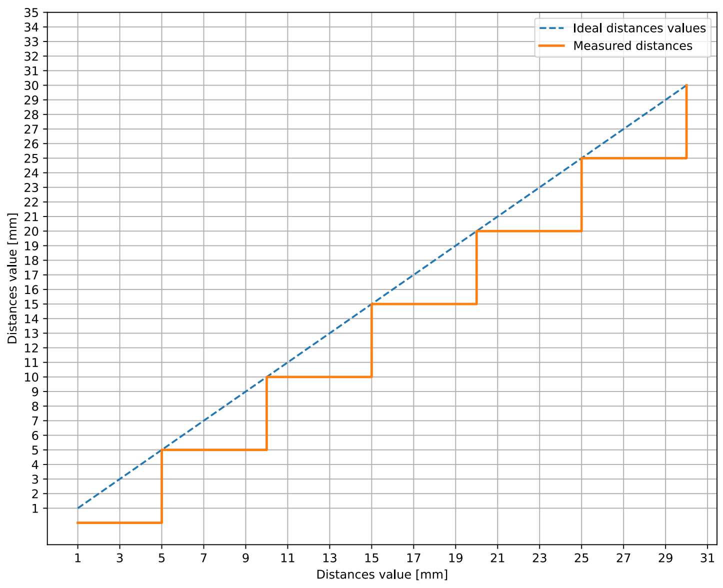
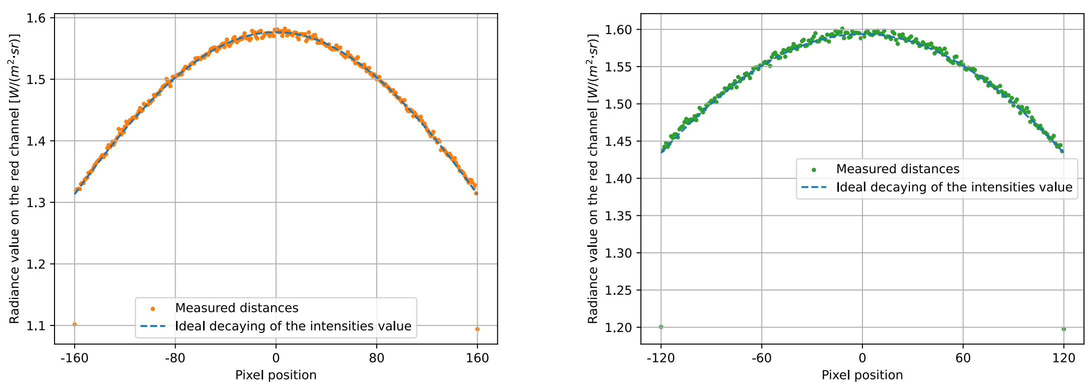


Figure 3.12: Quantization test results

Cross-section decay

Performing again the tests of the decay of the measured radiance over the cross-section of the sensor is very important since it is one of the key aspects ensuring that the simulated dToF properly reflects a real device.

Analyzing fig. 3.13 it is possible to see that also this renderer perfectly follows the law described by eq. (3.2). In particular, if we compare these results with the one presented in fig. 3.5 it is possible to see that with the new renderer the result are even cleaner. There are just two outliers on the principal raw and two on the principal column. These outliers are located at the extreme borders of the sensor so, probably their existence is a consequence of some discontinuities in that regions.

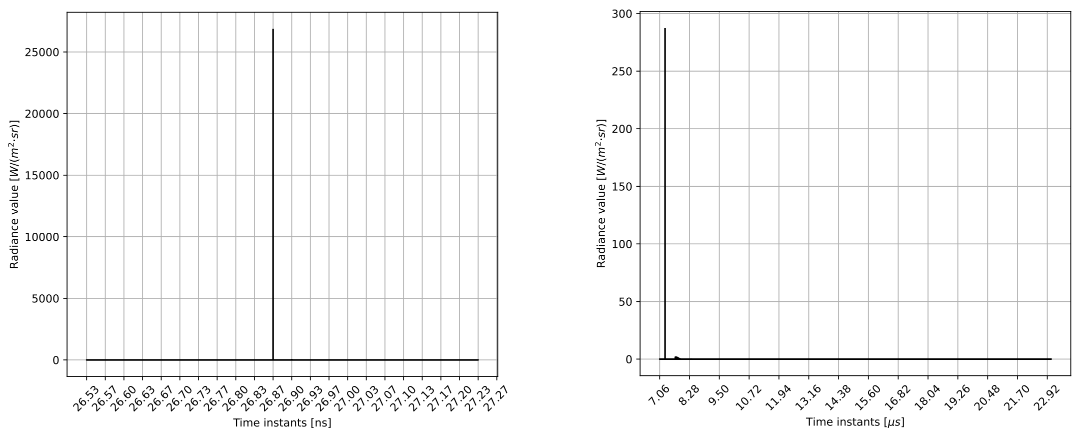


(a) Principal row decay (b) Principal column decay

Figure 3.13: Cross section decay test results

Comparison of the transient reconstruction

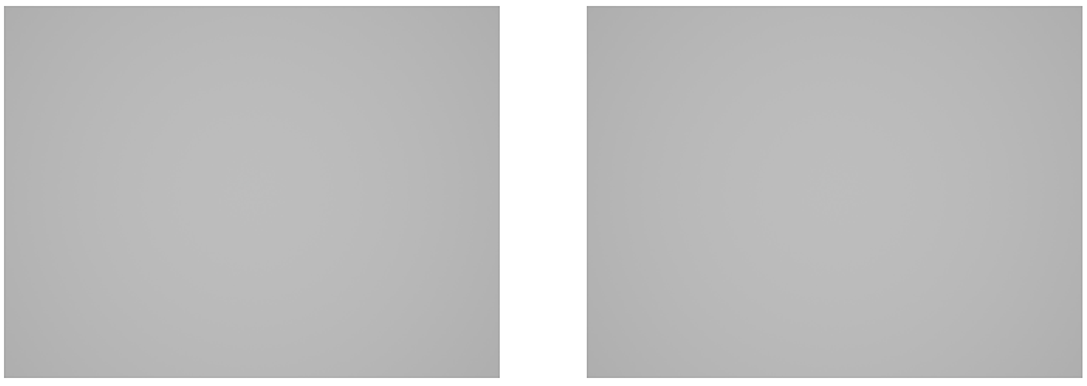
The last remaining test is probably the most important one. All the previous validation was already a success also for `mitsuba2-transient`. `Mitsuba-transient-nlos` seems to perform a little better on all of them but it remains to be proven if it can succeed where its predecessor failed. For that reason, the comparison test between the RGB renders and the one obtained by the transient information represents a turning point that defines if `Mitsuba 2` could be used or not for dToF simulation.



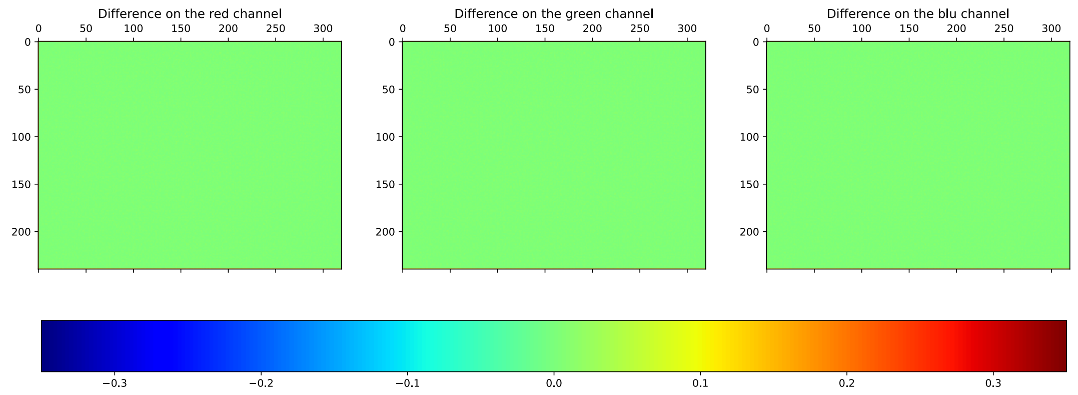
(a) White wall scene (b) Cornell scene

Figure 3.14: Representation of the transient vector of the central pixel

In the following, it will be presented the results of the same tests done using `mitsuba2-transient`, this time with the new version. From fig. 3.14 it is possible to notice that this time the transient vector of both the white wall scene and of the Cornell box one follows the expected behaviors. They both have a huge peak characterizing the direct reflection followed by a decreasing tail that contains all the information related to the indirect reflections.



(a) RGB image (b) Transient image

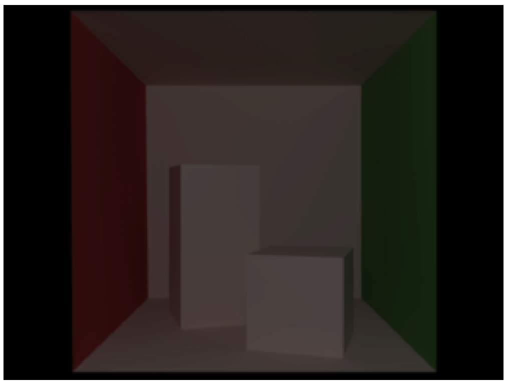


(c) Differnece between the two images

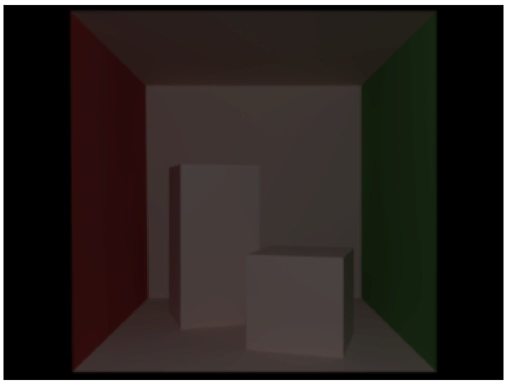
Figure 3.15: Comparison between the standard RGB render and the one obtained by the transient of the white wall scene

The success of the transient graphs test proves that in this fork of `Mitsuba 2` the transient computation procedure has actually been updated in order to be more consistent and realistic. This aspect is confirmed by the render comparison presented in fig. 3.15 where, again, the RGB render is identical to the one obtained starting from the transient. This time the results are even better compared to the one obtained by `mitsuba2-transient` since on the blue channel there are no more artifacts.

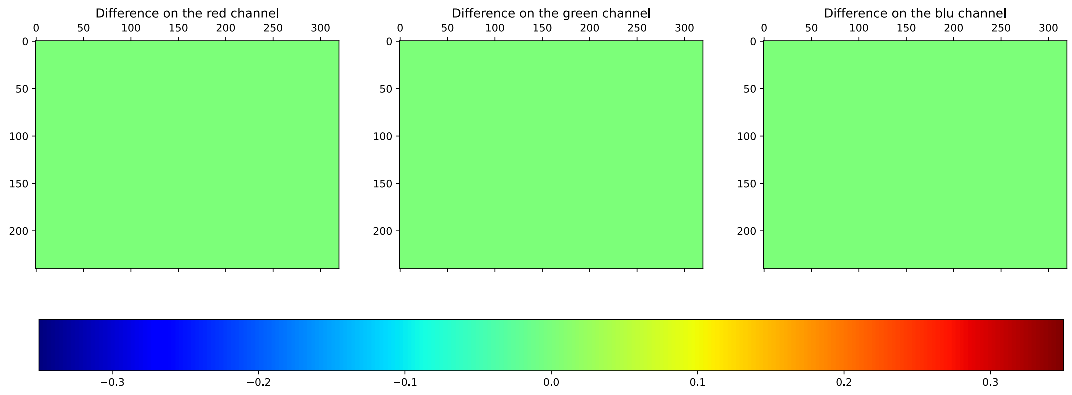
Finally comparing the renders of the Cornell box scene (fig. 3.16) it is possible to see that the two images are practically the same both visually and quantitatively. This time both the difference and the ratio, computed channel by channel,



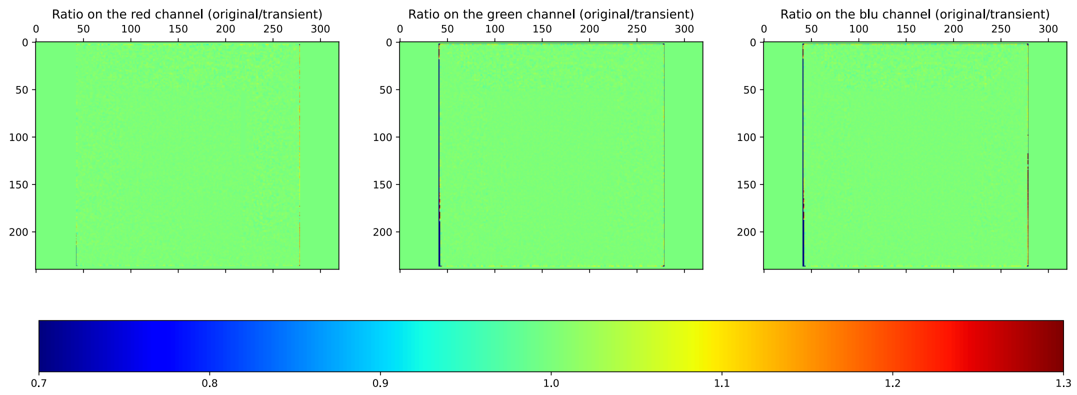
(a) RGB image



(b) Transient image



(c) Differnece between the two images



(d) Ratio between the two images

Figure 3.16: Comparison between the standard RGB render and the one obtained by the transient of the Cornell box scene

follow the expected behaviors (with just some noise on the edges)[10].

Given that `mitsuba2-transient-nlos` using a QVGA resolution succeeded in all the tests, it will be used to generate the dataset.

[10]This type of error is probably a consequence of the antialiasing applied to the RGB render

4 Dataset

4.1 Introduction

As extensively described in chapter 1, since the idea is to design and implement a NN model, a fundamental step of the work is to use a proper dataset to train and test the network. As already stated in section 3.1, in the literature there is no ready-to-use solution that perfectly fits the needs of this project.

After some evaluation of the possible strategies to solve this issue, we decided that the best way to proceed would have been to generate the dataset that we need. In this way, we would be able to control every single parameter of the data, and more importantly, there would be no compromises on the quality of the data.

After these considerations, we concluded that in order to carry out the project we need a set of scenes of the type "*look around corner*" characterized by:

- having the sensor located and oriented in different positions/directions,
- being ideal and so characterized by only MPI errors but no sensor or shot noise,
- having only simple but variable geometries as a hidden object that can assume different positions and rotations,
- having a front wall of different materials,
- having a middle wall of perfectly black and absorbing material (in his way it doesn't reflect any light emitted by the sensor),
- having no other elements in the scene, like floor, ceiling, etc.

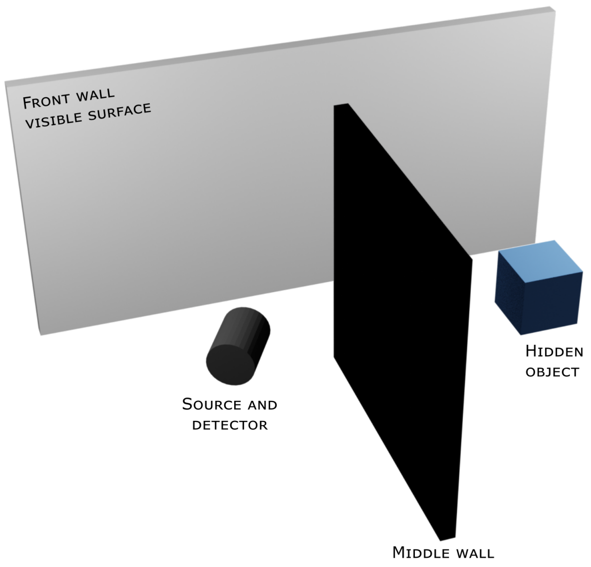


Figure 4.1: Sample dataset scene

Figure 4.1 represent a good summary of a generic scene contained in our dataset.

Of course, all the scenes have been rendered using `mitsuba2-transient--nlos` as if they have been acquired by a dToF sensor. For this reason, all the scenes are stored as transient vectors.

4.2 Structure of the dateset

To ensure that the dataset covers all the scenarios that we need and, at the same time, has enough elements to properly train a NN, it has been decided to generate a dataset composed of 1344 scenes, characterized by:

- image resolution: 320×240 (QVGA),
- number of samples for each render: $1 \cdot 10^6$,
- number of temporal bins: 2000,
- temporal bin size: $0.01m$,
- type of illumination: full field,
- sensor and projector horizontal fov: $60°$,
- front wall:
  - location (of the center) $\longrightarrow (x:\ 0m,\ y:\ 0m,\ z:\ 2m)$,
  - dimensions $\longrightarrow 8m \times 4m$,

- MIDDLE WALL:
  - location (of the center) $\longrightarrow (x:\ 2{,}35m,\ y:\ 0m,\ z:\ 2m)$,
  - dimensions $\longrightarrow 3.30m \times 4m$,
- GAP BETWEEN THE TWO WALLS: $70cm$,
- SENSOR BASIC LOCATION:
  - position $\longrightarrow (x:\ 1m,\ y:\ -1m,\ z:\ 1.65m)$,
  - rotation $\longrightarrow (x:\ 90°,\ y:\ 0°,\ z:\ 50°)$[1],
- HIDDEN OBJECT BASIC LOCATION:
  - position $\longrightarrow (x:\ 1m,\ y:\ 1m,\ z:\ 1.65m)$,
  - rotation $\longrightarrow (x:\ 0°,\ y:\ 0°,\ z:\ 0°)$,
  - max size $\longrightarrow (x:\ 0.5m,\ y:\ 0.5m,\ z:\ 0.5m)$.

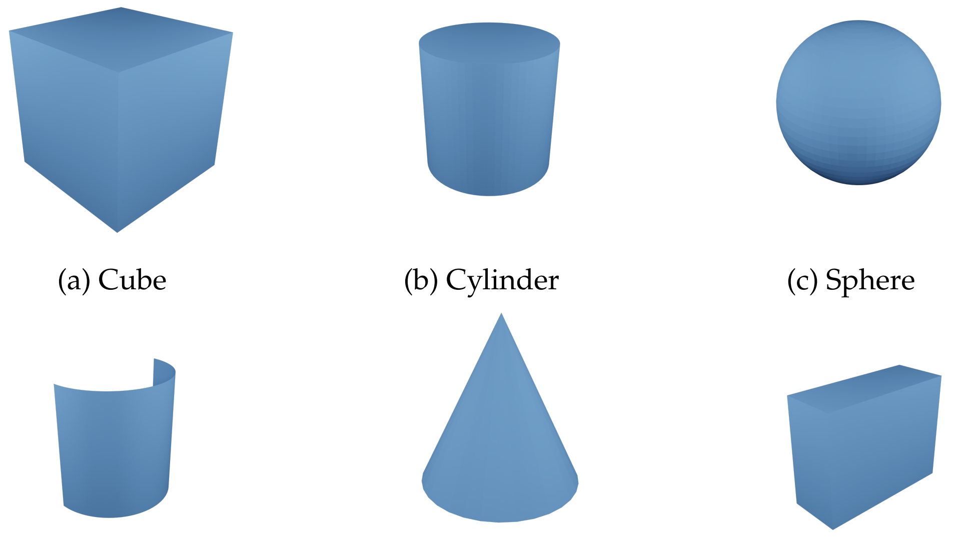
An essential aspect to keep in mind while designing a synthetic dataset is that it has to guarantee a high enough level of variability. In other words, it must represent the most possible number of different situations to capture as much as possible the variability that characterizes the real counterpart. This is fundamental since the data contained in the dataset has to contain enough information to allow a NN model to learn the desired task.

In order to give some variability to our dataset we decided to consider different locations and orientations for both the sensor and the hidden object, and also to allow the front wall to assume different roughness values. In particular, the possible object shape are presented in fig. 4.2. Other than the basic shape $\frac{1}{7}$ of the scene uses as hidden object a random combination of two casually sampled basic objects[2]. As regards the camera position and rotation they are randomly sampled from:

- possible position $\longrightarrow (x \in [1,\ 1.3]m,\ y \in [-1.3,\ -1]m,\ z \in [1.5,\ 1.8]m$,
- possible rotation $\longrightarrow (x \in [85,\ 95]°,\ y \in [-5,\ 5]°,\ z \in [50,\ 90]°$.

[1]The rotation of 50° on the z axes correspod of rotating the sensor 40° towards the middle wall

[2]This type of object is generated by sampling one of the basic shapes and after its position has been determined, sample another shape translate it of a value $t \in [-0.12,\ 0.15]m$ and rotate it of $r \in [-45,\ 45]°$



(a) Cube (b) Cylinder (c) Sphere

(d) Concave plene (e) Cone (f) Parallelepiped

Figure 4.2: Overview of the considered hidden object's shapes

While for the hidden object:

- possible position $\rightarrow (x \in [1,\ 1.3]m,\ y \in [0.5,\ 1.3]m,\ z \in [1.25,\ 1.95]m,$
- possible rotation $\rightarrow (x \in [-90,\ 90]^\circ,\ y \in [-90,\ 90]^\circ,\ z \in [-90,\ 90]^\circ.$

Finally the roughness value of the front wall can take values in $\alpha \in [0.3,\ 1]$ (step 0.05). All the possible position/rotation values are randomly sampled to further increase the variability.

Other than adding a lot of variabilities we also wanted to generate a flexible dataset, to ensure that it would be possible to train the network only on a specific part of the dataset where only one aspect is varying (e.g., only the hidden object position/rotation). In this way, during the designing of the NN model, it would be possible to understand which elements are the more problematic for the architecture to learn. To fulfill this requirement, the dataset has been divided into two main groups each one of them further split into two different sub-categories:

- CONSTANT FRONT WALL: in all the scenes the front wall is the same, characterized by a perfectly white and diffuse material:
  - FIXED CAMERA: the camera position and rotation are the same in every scene,

  - VARIABLE CAMERA: the camera position and rotation differ for each scene.

- VARAIBLE FRONT WALL: the front wall is located always in the same position but in each scene the roughness of the material is different:
  - FIXED CAMERA: the camera position and rotation are the same in every scene,
  - VARIABLE CAMERA: the camera position and rotation differ for each scene.

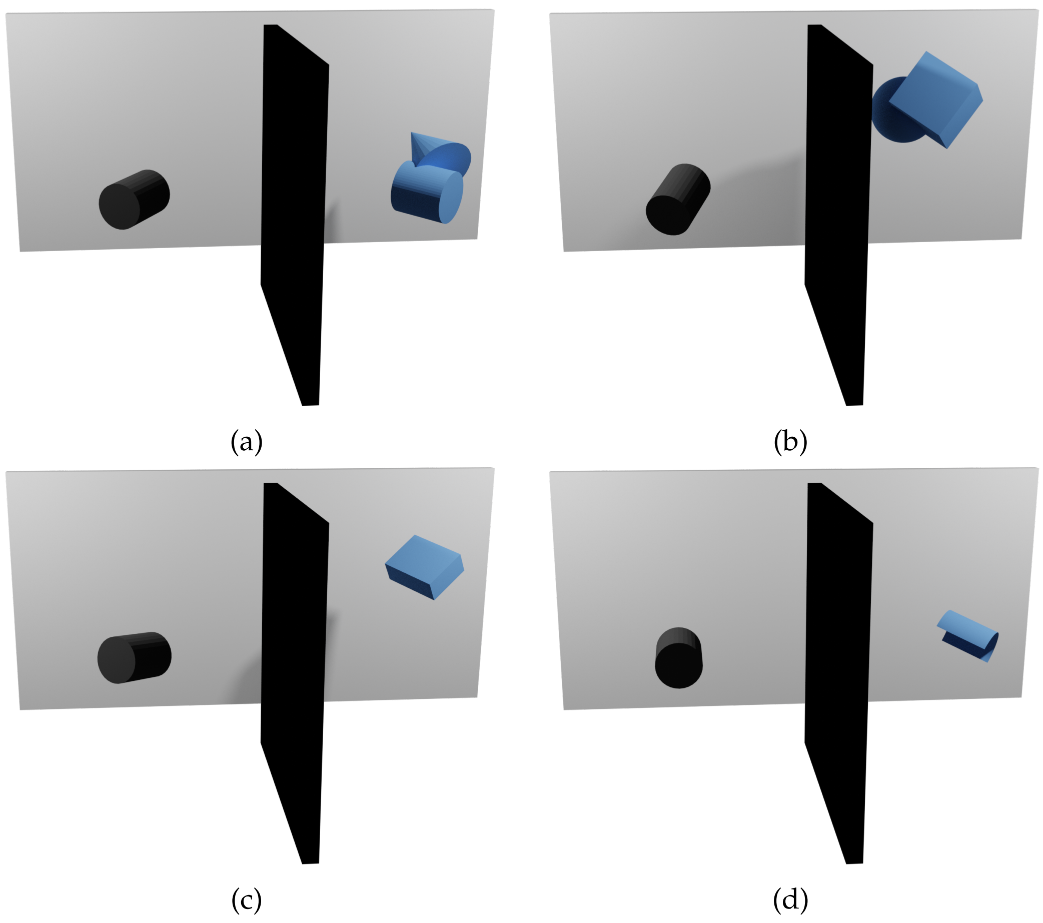

(a) (b) (c) (d)

Figure 4.3: Example of some scenes of the dataset

5 Fermat flow analysis and test

The method proposed in our work to perform Non-Line-of-Sight imaging is based on the use of Time of Flight sensors combined with a Neural Network model. Of course, this is not the only possible approach to accomplish this task but exists many more alternatives solution (i.e., methods based on different sensors and/or technologies).

In particular, we will focus our attention on the NLoS imaging pipeline proposed in [3], the *Fermat flow*. We will present, test, and discuss this method since it is based on a consistently different paradigm to solve our same issue and so could represent a good candidate to perform performance comparison. Other than that, we will also propose an extension of this method that aims at overcoming some of its limitations.

In the following sections, we will present and then deeply test the *Fermat flow*.

5.1 How the Fermat flow works

The work proposed in [3] tries to overcome two main issues that affect NLoS imaging approach based on transient information produced by a direct Time of Flight (dToF). This kind of method inverts the time-resolved radiometric image formation process to perform the 3D reconstruction. Proceeding in this way has two big problems:

- it relies on radiometric information and, given the nature of the actual Single-Photon Avalanche Diode (SPAD) sensor, it is affected by photon noise and ambient light;

- to simplify the inverse problem it works only on objects characterized by Lambertian reflectance[1].

On the other hand, the solution proposed in [3], still exploits the light reflection to reconstruct the hidden object but, to do so, uses only geometric constraints derived from the transient measurements of a NLoS object. The authors have developed a method based on Fermat's principle. More precisely, they have defined the concept of *Fermat path* that corresponds to a specific path followed by a photon between the LoS and NLoS scene and it is linked to discontinuities in the transient measurements. Another important contribution of the considered paper is represented by the *Fermat flow* algorithm that allows for an accurate NLoS shape reconstruction.

It is important to notice that the proposed approach is made in such a way that makes it agnostic to the specific transient imaging technology used but also BRDF-invariant and robust to imperfections in intensity measurements.

5.1.1 Implementation details

In [3] to perform all the measurements and tests has been used a transient imaging system composed of a light source and a detector located at position $\boldsymbol{s},\ \boldsymbol{d} \in \mathbb{R}^3$ respectively. Other than that to properly understand the work done it is necessary to define some other key elements:

- the *visible scene* $\mathcal{V} \subset \mathbb{R}^3$ is the union of surfaces contained within the common line of sight of the source and detector;
- the *Non-Line-of-Sight scene* $\mathcal{X} \subset \mathbb{R}^3$ is the union of all the surfaces located outside of the LoS of the sensor that can indirectly receive light from the light source using single reflection or transmission through the visible scene and can indirectly send light to the detector in a similar manner.

Under the above constraints it is possible to say that the transient $I(t; \boldsymbol{v})$ [2] measured by the detector corresponds to only photons that has followed paths of the form $\boldsymbol{s} \rightarrow \boldsymbol{v} \rightarrow \boldsymbol{x} \rightarrow \boldsymbol{v} \rightarrow \boldsymbol{d}$ [3].

[1]Lambertian reflectance = property that defines an ideal "mate" or diffusely reflecting surfaces. The apparent brightness of a Lambertian surface to an observer is the same regardless of the observer's angle of view. More technically, the surface's luminance is isotropic, and the luminous intensity obeys Lambert cosine law.

[2]t represents the time-of-flight

[3]Photon paths characterized by more than one interaction in the NLoS scene have greatly reduced Signal-to-Noise-Ratio (SNR) and so are negligible

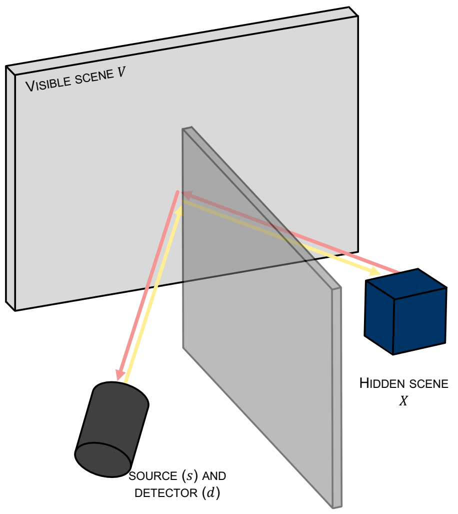

VISIBLE SCENE $\mathcal{V}$
HIDDEN SCENE $\mathcal{X}$
SOURCE (s) AND DETECTOR (d)

Figure 5.1: Representation of a standard light path $\boldsymbol{s} \to \boldsymbol{v} \to \boldsymbol{x} \to \boldsymbol{v} \to \boldsymbol{d}$

Finally it is necessary to assume to have calibrated the distance from the source to the visible point and the way back: $\tau_{\mathcal{V}}(\boldsymbol{v}) \triangleq \| \boldsymbol{s} - \boldsymbol{v} \| + \| \boldsymbol{d} - \boldsymbol{v} \|$. This quantity is needed to compute the path length travelled in $\mathcal{X}$ as $\tau \triangleq ct - \tau_{\mathcal{V}}(\boldsymbol{v})$ (c = speed of light). At this point, we can write the transient as:

$$I(\tau; \boldsymbol{v}) = \int_{\mathcal{X}} f(\boldsymbol{x}; \boldsymbol{v})\delta(\tau - \tau(\boldsymbol{x}; \boldsymbol{v}))\, dA(p, q), \tag{5.1}$$

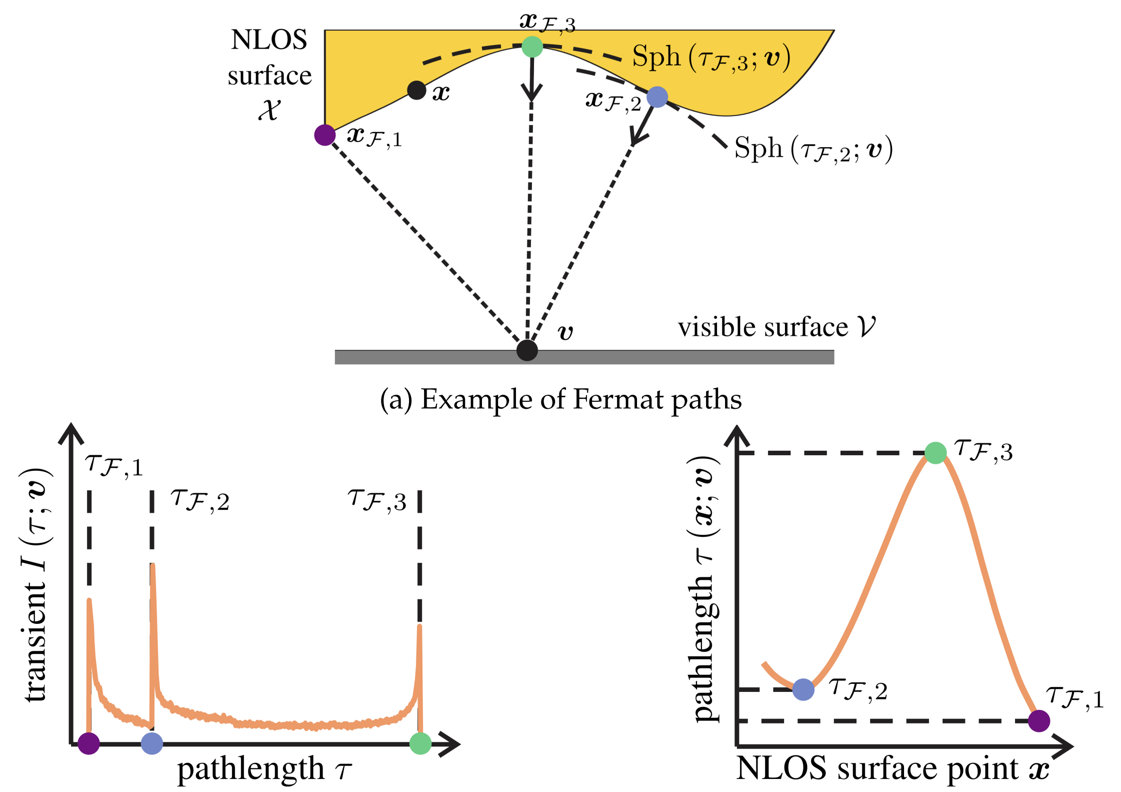
where $\tau(\boldsymbol{x}; \boldsymbol{v}) \triangleq 2 \cdot \| \boldsymbol{x} - \boldsymbol{v} \|$, $(p, q) \in [0, 1]^2$ is a parametrization of the NLoS surface $\mathcal{X}$, $A(p, q)$ is the corresponding area measure and the throughput f summarize shading, reflectance and visibility.

In the following, we will consider the NLoS scene $\mathcal{X}$ as it is formed by a union of smooth surfaces. Other than that we introduce also the concept of the boundary of $\mathcal{X}$, $\partial\mathcal{X} \subset \mathcal{X}$, that is the set of points on the NLoS surface where a surface normal is not defined. Then given any visible point $\boldsymbol{v}$ we can define three sets:

- the *specular set* $\mathcal{S}(\boldsymbol{v}) \subset \mathcal{X}$ that consists of all points $\boldsymbol{x} \in \mathcal{X}\backslash\partial\mathcal{X}$ such that the vector $\boldsymbol{v} - \boldsymbol{x}$ is orthogonal to the tangent plane $T_{\boldsymbol{x}}\mathcal{X}$ of $\mathcal{X}$ at $\boldsymbol{x}$;
- the *boundary set* $\mathcal{B}(\boldsymbol{v}) \subset \partial\mathcal{X}$ that consists of all points $\boldsymbol{x} \in \nabla\mathcal{X}$ such that the vector $\boldsymbol{v} - \boldsymbol{x}$ is orthogonal to the tangent vector $\hat{\boldsymbol{t}}(x)$ of $\partial\mathcal{X}$ at $\boldsymbol{x}$;
- the *Fermat set* $\mathcal{F}(\boldsymbol{v}) \subset \mathcal{X}$ that consists of the union of the two previous sets, $\mathcal{F}(\boldsymbol{v}) \triangleq \mathcal{S}(\boldsymbol{v}) \cup \mathcal{B}(\boldsymbol{v})$.

Putting together all the above evaluations and the Fermat's principle it is possible to say that:

Proposition 5.1.1. *Assume that the BRDF of the $\mathcal{X}$ surface is non-zero in the specular direction. Then, for all $\boldsymbol{x} \in \mathcal{F}(\boldsymbol{v})$, the transient $I(\tau; \boldsymbol{v})$ will have a discontinuity at pathlength $\tau(\boldsymbol{x}; \boldsymbol{v})$. If $\boldsymbol{x} \in \mathcal{S}(\boldsymbol{v})$, then $I(\tau; \boldsymbol{v})$ will additionally have a vertical asymptote at $\tau(\boldsymbol{x}; \boldsymbol{v})$ (see fig. 5.2).*


NLOS
surface
$\mathcal{X}$
$\boldsymbol{x}_{\mathcal{F},3}$
Sph $(\tau_{\mathcal{F},3}; \boldsymbol{v})$
$\boldsymbol{x}$
$\boldsymbol{x}_{\mathcal{F},2}$
$\boldsymbol{x}_{\mathcal{F},1}$
Sph $(\tau_{\mathcal{F},2}; \boldsymbol{v})$
$\boldsymbol{v}$
visible surface $\mathcal{V}$
transient $I(\tau; \boldsymbol{v})$
$\tau_{\mathcal{F},1}$
$\tau_{\mathcal{F},2}$
$\tau_{\mathcal{F},3}$
pathlength τ
pathlength $\tau(\boldsymbol{x}; \boldsymbol{v})$
$\tau_{\mathcal{F},3}$
$\tau_{\mathcal{F},2}$
$\tau_{\mathcal{F},1}$
NLOS surface point $\boldsymbol{x}$

(a) Example of Fermat paths

(b) representation of the transient $I(\tau; \boldsymbol{v})$

(c) representation of the pathlength function $\tau(\boldsymbol{x}; \boldsymbol{v})$

Figure 5.2: Representation of the theory of Fermat paths. Points $(x_{\mathcal{F},2}, x_{\mathcal{F},3})$ (surface) and $x_{\mathcal{F},1}$ (boundary) are the only points $x \in \mathcal{X}$ in the NLoS that generate path that satisfies Fermat's principle. [3]

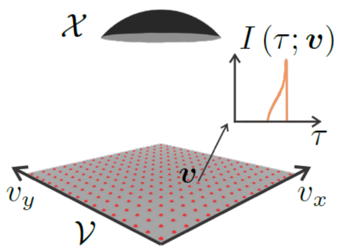
Proposition 5.1.1 implies two interesting consequences:

- pathlengths where the transient $I(\tau; \boldsymbol{v})$ is discontinuous are determined completely by the function $\tau(\boldsymbol{x}; \boldsymbol{v})$ that depends only on the geometry of $\boldsymbol{v}$ and $\mathcal{X}$ ($\Rightarrow$ *BRDF invariance*);
- each of the Fermat pathlengths is a stationary point of the function $\tau(\boldsymbol{x}; \boldsymbol{v})$.

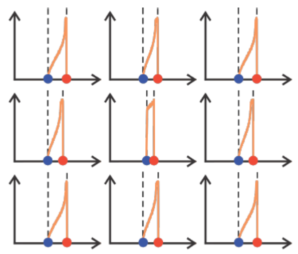
Furthermore, if the BRDF of $\mathcal{X}$ is not perfectly specular, it is possible to identify also the type of stationarity from the shape of the transient at the neighborhood of the discontinuity[4].

All of the above discussions have been performed on a confocal setup where both the source and the detector are pointing at the same point, but it is possible to straightforwardly generalize everything to the non-confocal case.

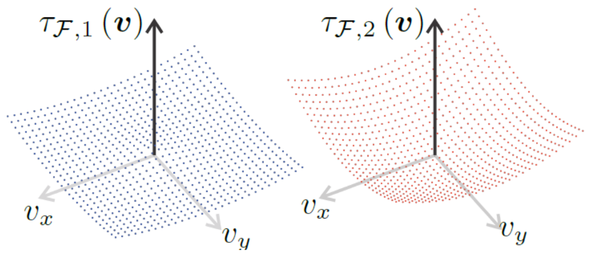
Once we have extracted all the Fermat paths from the NLoS scene it is necessary to properly reconstruct the surface. More precisely it is possible to say that each path length $\tau_{\mathcal{F}}$ constraint the corresponding point $\boldsymbol{x}_{\mathcal{F}} \in \mathcal{F}(\boldsymbol{v})$ to lie on the tangent sphere $Sph(\tau_{\mathcal{F}}/2; \boldsymbol{v})$ [5] and, if $\boldsymbol{x}_{\mathcal{F}} \in \mathcal{S}(\boldsymbol{v})$, also constraints its normal and curvature. So knowing that in order to reconstruct the surfaces it is necessary to develop a procedure able to fully determine $\boldsymbol{x}_{\mathcal{F}}$ and it's normal. At this point, it is possible to produce an oriented point cloud of the NLoS surface $\mathcal{X}$. In [3] has been proposed an algorithm that perform this reconstruction called *Fermat flow* that is summarized in fig. 5.3.



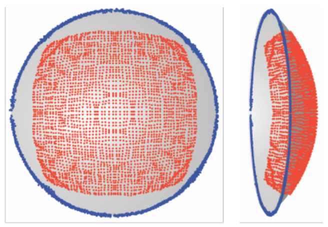
(a) Scanning



(b) Discontinuity detection



(c) Gradient estimation by interpolation



(d) Point cloud reconstruction

Figure 5.3: Reconstruction pipeline [3]

Starting from the Fermat pathlength function $\tau_{\mathcal{F}}(\boldsymbol{v})$:

$$\tau_{\mathcal{F}}(\boldsymbol{v}) = \{\tau : I(\tau; \boldsymbol{v}) \; is \; discontinuous\}, \tag{5.2}$$

[4]For example, if $\tau_{\mathcal{F}}$ is a local maximum, the discontinuity in the transient $I(\tau; \boldsymbol{v})$ occurs at the limit from the left, $\tau \to \tau_{\mathcal{F}}^{-}$, and the transient decreases to the right of $\tau_{\mathcal{F}}$

[5]This represents a sphere of radius $\tau_{\mathcal{F}}/2$ centered in $\boldsymbol{v}$

it is possible to define the point $x_{\mathcal{F}}$ in the NLoS as:

$$x_{\mathcal{F}} = v - \left(\frac{\tau_{\mathcal{F}}(v)}{4}\right) \nabla_v \tau_{\mathcal{F}}(v) \tag{5.3}$$

The operation described in eq. (5.3) can be performed by a simple geometric operation, intersecting the sphere $Sph(\tau_{\mathcal{F}}/2; v)$ with the line $v - \lambda \nabla_v \tau_{\mathcal{F}}(v)/2$. Then if $x_{\mathcal{F}} \in \mathcal{S}(v)$ it is also possible to reconstruct the normal at $x_{\mathcal{F}}$ as:

$$\hat{n}(x_{\mathcal{F}}) = -\frac{\nabla_v \tau_{\mathcal{F}}(v)}{2} \tag{5.4}$$

All of this perfectly works, the only problem is that it is not possible to measure the gradient $\nabla_v \tau_{\mathcal{F}}(v)$ directly, for that reason the only way to proceed is to compute it by interpolation (note that the interpolation procedure must be performed separately for each branch of the Fermat pathlength function $\tau_{\mathcal{F}}(v)$).

The just described procedure produces an oriented point cloud, of density comparable to one of the measurements in $\mathcal{V}$. To conclude our work and obtain the final output it is possible to use an algorithm that takes advantage of normal information to fit a surface representation to the point cloud (e.g., triangular mesh).

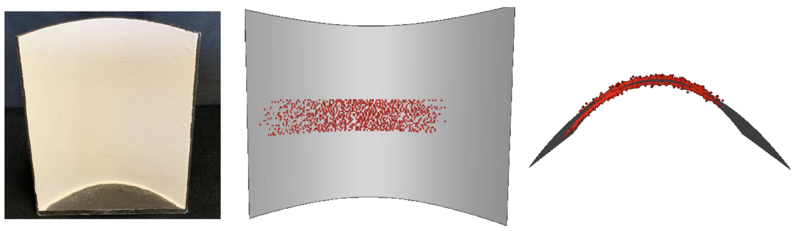
5.1.2 Performance evaluation

In order to test the proposed approach, the authors of [3] have performed different real-world tests in NLoS scenarios using different types of surfaces (concave and convex) characterized by different BRDF. They also used two different types of transient imaging sensors.

As we can see in fig. 5.4 the Fermat flow approach has been able to produce surfaces that closely reproduce the shape of the original objects, in particular, it matches the ground truth within $2mm$.

Figure 5.5 shows some other examples of reconstructed surfaces using the *Fermat flow* algorithm. Also in this case it is possible to notice that the proposed approach produces surfaces that closely match the shape of the given object, including accurate normals. If we use a more precise sensor than the one used for the reconstruction of fig. 5.5 we can recover even finer details from the object in the NLoS as we can see in fig. 1.1.

Of course as with all the other approaches, also the one proposed in [3] has


(a) Photograph and 3D reconstruction of a paraboloid object

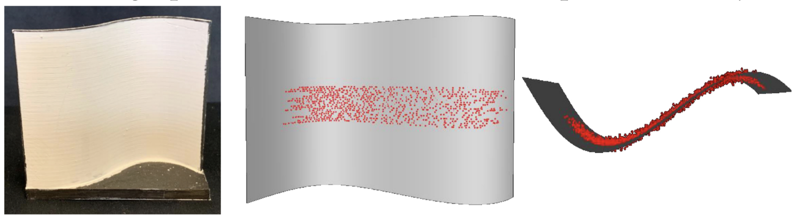
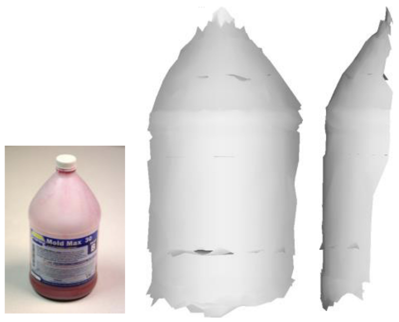
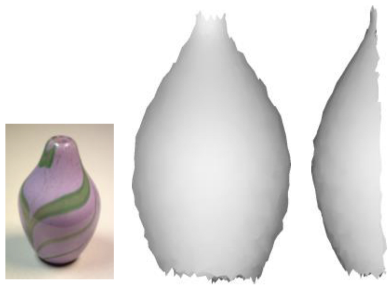
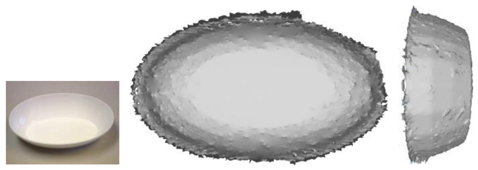

(b) Photograph and 3D reconstruction of a sigmoid object

Figure 5.4: Comparison with the ground truth [3]


(a) Plastic jar


(b) Glass vase


(c) Plastic bowl

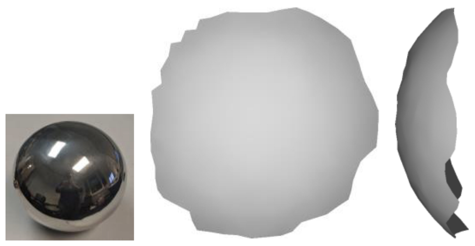

(d) Metal sphere

Figure 5.5: Some example of reconstructions produced by the Fermat flow (two views for each object)[3]

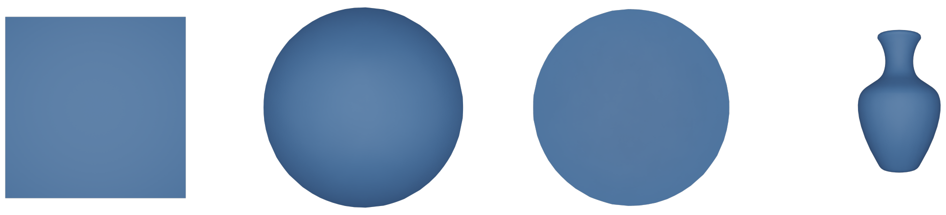
its drawbacks. In particular, it is important to keep in mind that the produced reconstruction can be sensitive to inaccurate discontinuity detection. The final quality of the reconstruction may also suffer if it is not available sufficiently dense measurements for estimating the Fermat pathlength gradient through interpolation. Finally, in order to ensure the BRDF invariance, the model makes no use of information about the measured intensity in the NLoS scene.

5.2 Limitations of the Fermat flow algorithm

Looking at the results proposed by the author of [3] (fig. 5.5) it is clear that the proposed approach works well and reliably in the reconstruction of the NLoS scene. These results are not enough to guarantee that this method works well in conditions not covered by the original paper. For this reason, was essential to deeply test the *Fermat flow* algorithm in our scenario, in order to properly stress the method and found all its limitations.

The testing procedure was performed as follows:

1. test the algorithm on the same scene proposed by the author of [3],
2. test the algorithm on some of our scenes,
3. compares the obtained results in order to extract the limitation of the method.

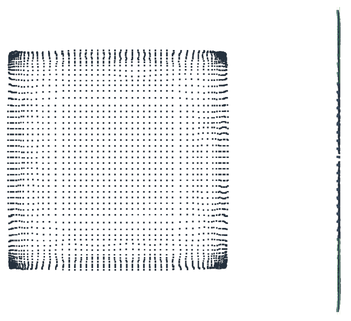


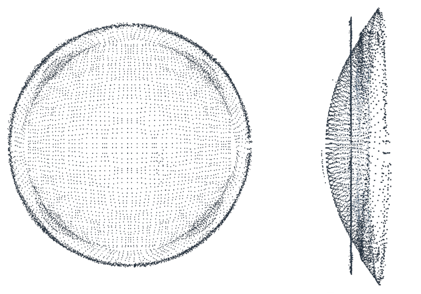
(a) Plane ground truth (b) Concave sphere ground truth (c) Convex sphere ground truth (d) Diffuse vase ground truth

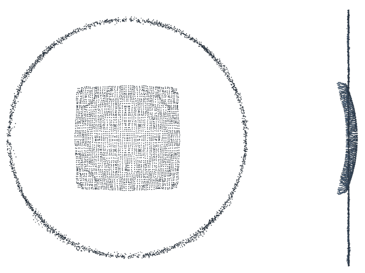
Figure 5.6: Sample reconstructions of the object used in [3]

As it is possible to see comparing fig. 5.6 with fig. 5.7 it is clear that the *Fermat flow* algorithm works really well on the sample object provided by the authors. Indeed the reconstructed point cloud perfectly reflects the general shape of the ground truth object.

In any case, it is important to point out that in more complex scenes where the surface is not perfectly flat the reconstruction is not complete and presents big holes. Other than that, it is important to analyze the reconstruction of fig. 5.7c. In this specific case in fact, in order to reconstruct both the inner part and the outer part of the sphere, it is required that the *Fermat flow* algorithm analyzes more than just the first discontinuity. Following the explanation of how *Fermat paths* work (section 5.1) it is clear that each discontinuity of the transient vector


(a) Plane


(b) Concave sphere

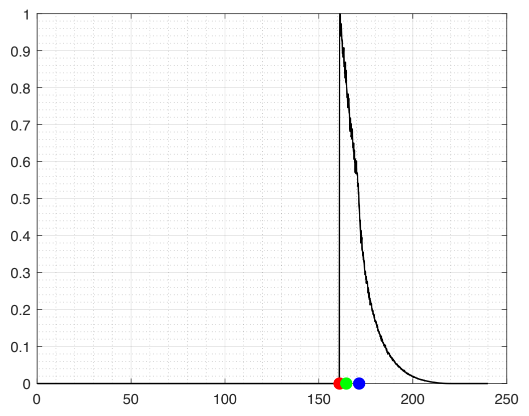

(c) Convex sphere

(d) Diffuse vase

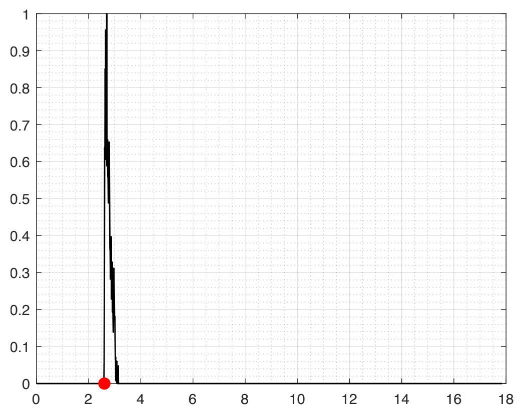
Figure 5.7: Sample reconstructions of the object used in [3]

allows identifying a different set of points in the NLoS scene. Consequently, in order to fully reconstruct a complex object, it is necessary to recover information coming not only from the first discontinuity. This problem does not occur for simple shapes due to the fact that the information recovered from the first discontinuity of the transient essentially characterizes all the hidden points that are the closest to the front wall. In other world the ones for which the path $\boldsymbol{s} \rightarrow \boldsymbol{v} \rightarrow \boldsymbol{x} \rightarrow \boldsymbol{v} \rightarrow \boldsymbol{d}$ is the shortest. Indeed looking at fig. 5.7a where each point has the same distance from the front wall the first discontinuity is enough to recover the full shape of the target object. This aspect represents a major limitation to the real-world usability of the method since reliably recovering the discontinuities following the first one is extremely difficult.

In fig. 5.8 a sample transient vector coming from the convex sphere of [3] and one coming from a scene of our dataset (hidden object = cube parallel to the front wall) are compared. It is clearly visible that from our transient it is extremely difficult to reliably identify more than the first discontinuity. This seems to identify that the samples used by the authors of [3] are perfectly ideal. The result could become even worse if the provided input transient comes from a real sensor, and is so affected by all its related sensor noises.



(a) Sample transient vector of the convex sphere



(b) Sample transient vector of one of our scenes (cube)

Figure 5.8: Transient comparison between one scene from [3] and one from our dataset (red dots mark the first discontinuity, the other dots the following ones)

Another big limitation that we have encountered in the *Fermat flow* is the fact that, as stated in section 5.1.1, to work it requires that the scene to acquire is not illuminated full field but pixel by pixel. This means that if the sensor is, for example, QVGA it is necessary to perform $320 \times 240 = 76800$ different acquisitions. Of course, during the whole duration of the process, neither the sensor nor the hidden object can be moved otherwise the entire acquisition is useless.

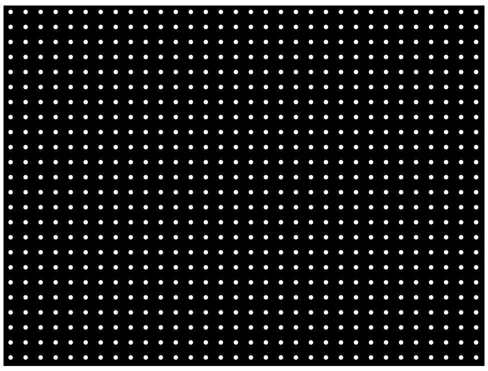


Figure 5.9: Example of an illumination grid of 32×24 pixels (time multiplexed)

This aspect made the retrieval procedure extremely complex and long. It is possible to use a spot illuminator and illuminate not the full FoV of the sensor but just a subsample grid of the total pixels in order to reduce the acquisition time (e.g., fig. 5.9). All the selected dots still required to be illuminated and acquired one by one. This workaround simplifies the process but at the same time also reduce the quality of the obtained result since the fewer pixel are illuminated the fewer NLoS point will be recovered.

After this first consideration, we tested the *Fermat flow* on some of our scenes. From all of our testing, we did not recover any useful reconstruction. Independently from the considered scene, the results were always really similar to the one of fig. 5.10, which should represent the reconstruction of a cube parallel to the front wall (similar setup to the one shown in fig. 4.1).

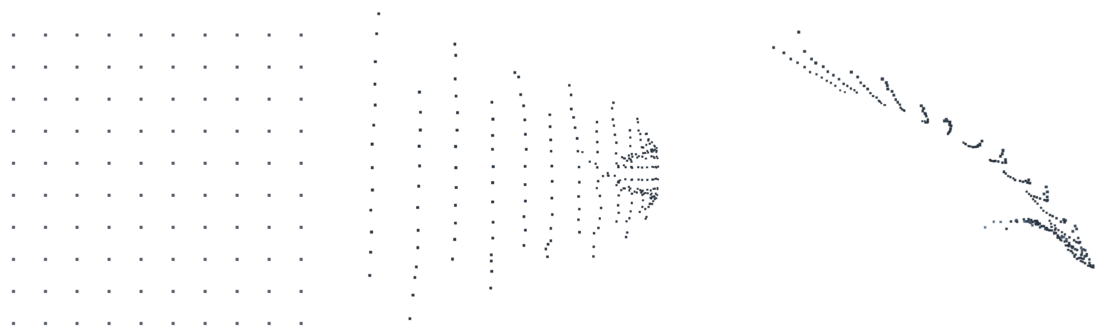

(a) Ground truth (b) Front view (c) Side view

Figure 5.10: *Fermat flow* reconstruction of a scene of our dataset (hidden object = cube) using an illumination grid of $32 \times 24 = 768px$

These kinds of results are definitely not representing the desired object and are extremely worse than the ones obtained on the provided test scenes. For this reason, we tried to understand which elements of our scenes could cause such a difference. In particular, we have identified two main possibilities:

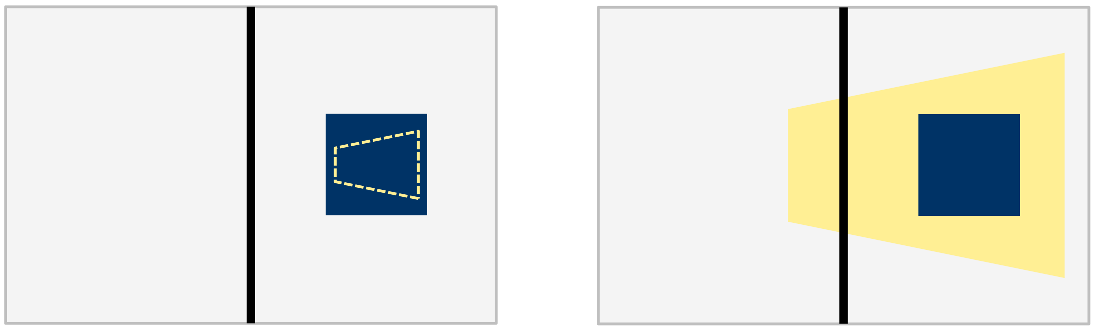
1. in [3] the hidden object and the sensor are located at a maximum distance of $80cm$ from the front wall and $2cm$ from the middle one while in our case the relative distances are on the order of $1m$,
2. the FoV of the sensor used in the original paper illuminates a portion of the front wall that is always smaller or equal to the projection of the hidden object on the same wall (fig. 5.11)[6].

To understand which of these two elements is the problem we performed two different tests.

For the first problem, we tried to generate a scene based on our dataset but with much reduced relative distances between the camera, the hidden object, and the wall. After running the *Fermat flow* algorithm on the test scene just

[6]In the scenes of our dataset, due to the geometry of the environment, the FoV of the sensor is always bigger than the surfaces obtained by projecting the hidden object on the wall

described we obtained the same result of fig. 5.10. We can conclude that this is not the issue.

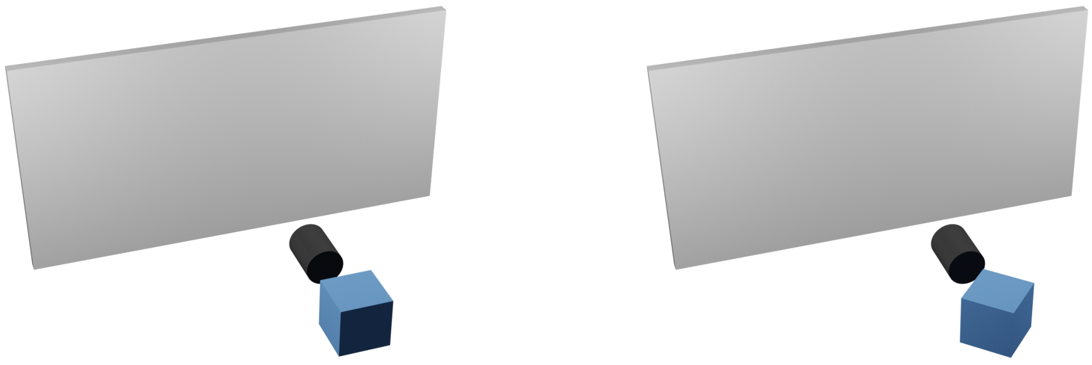


(a) Optimal setup, the illuminated area (yellow) is smaller than the surface obtained by projecting the hidden object (blue cube) on the front wall

(b) Suboptimal setup, the illuminated area (yellow) is bigger than the surface obtained by projecting the hidden object (blue cube) on the front wall

Figure 5.11: Schematized representation of the illuminated area limitation of the *Fermat flow*

Regarding the second test, we designed two different ideal test scenes that force the FoV of the sensor to be smaller than the area obtained by projecting the hidden object on the front wall. As it is possible to see in fig. 5.12 we removed the middle wall, pointed the camera directly at the front wall, and put the hidden object directly behind the camera. Of course, a setup like that is absolutely not realistic but represents a good testing environment.

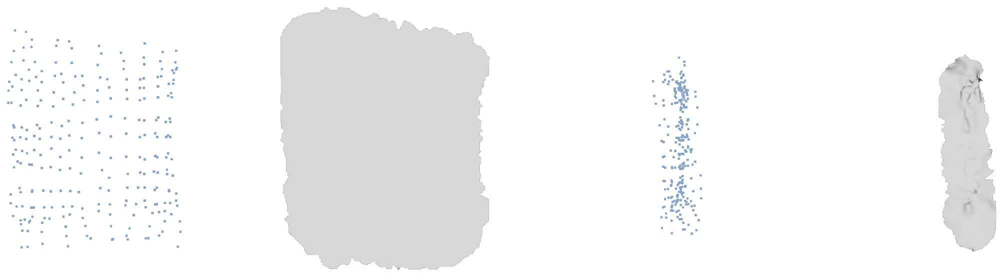


(a) Cube parallel to the front wall

(b) Cube rotated of 45° on the z axes

Figure 5.12: Representation of the two simplified scenes created to test the *Fermat flow*

Looking at fig. 5.13 it is clear that, under ideal conditions, the *Fermat flow* works. And as it is possible to see in figs. 5.13c and 5.13d using only one discontinuity this method is able to recover only the edge of the cube (the closest section to the white wall). This test proves that in order to make *Fermat flow* works it is essential that the illuminated section of the front wall is smaller than the area obtained by projecting the hidden object on the same wall. Other than that it also demonstrates that using only one discontinuity, in many cases is not enough.



(a) Cube parallel to the front wall (point cloud)

(b) Cube parallel to the front wall (mesh)

(c) Cube rotated of 45° over the z axes (pc)

(d) Cube rotated of 45° over the z axes (mesh)

Figure 5.13: Reconstructions produced by the *Fermat flow* of the scenes represented in fig. 5.12

After these considerations, we still have to find a way to properly apply the Fermat theory to our dataset scenes. To accomplish that the only way that we have found is to pre-process the transient data that we give in input to the *Fermat flow* in order to analytically remove the biggest possible number of points that will end up being outside of the allowed area on the white wall. Due to the setup of our scene, all of the transients to be removed are characterized by a really small global component x_g, since if a ray never hit the object there will be no reflected information by the object but only rendering noise. To clean the input data we applied an energy-based thresholding mechanism. More precisely for each transient vector, it is required to compute the maximum value of the global component and compare it with the sum of x_g. If the sum value is smaller than 65% of the maximum value the considered transient will be discarded.

Applying such a cleaning we manage to obtain the reconstruction of fig. 5.14 starting from the one of fig. 5.10. From this new reconstruction, it is possible to see that most of the information present in the previous one was generated by noise and the points actually related to the hidden object represent a small section in the lower-right part of fig. 5.10c. To verify if this type of cleaning properly works, we tested it also on a slightly different scene. We considered

(a) Point cloud of the reconstruction (b) Mesh of the reconstruction

Figure 5.14: *Fermat flow* reconstruction of a hidden cube after the pre-processing

the same scene presented in fig. 4.1 swapping the cube with a sphere.

Looking at fig. 5.15 it is clear that our cleaning process is not good enough to perfectly remove all the outliers[7]. Indeed in fig. 5.15b it is possible to see that the surface of the sphere was correctly retrieved, but unfortunately, it is still surrounded by a lot of noise.

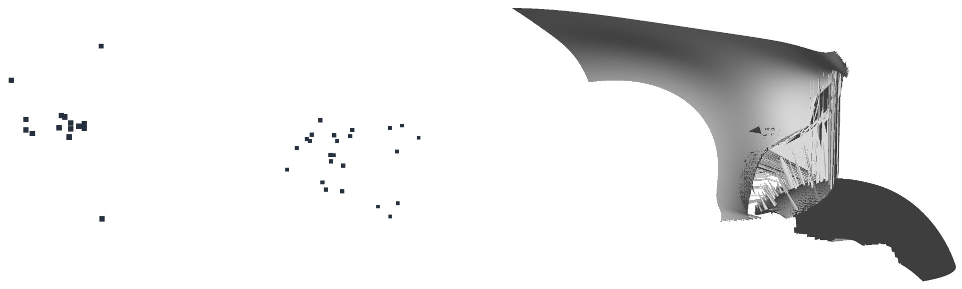

(a) Point cloud of the reconstruction (b) Mesh of the reconstruction

Figure 5.15: *Fermat flow* reconstruction of a sphere after the pre-processing

5.3 Fermat Flow with standard direct Time of Flight and indirect Time of Flight sensor

From the discussions of section 5.2 it is evident that outside of the strict constraints presented, the *Fermat flow* algorithm does not work well enough to be actually used for Non-Line-of-Sight imaging. Apart from that this method

[7]Also increasing the threshold value is not enough to obtain a perfect result, the cleaning is either too strong or too weak

still has its own strengths, such as the fact that its reconstructions depend only on the geometry of the scene and it is completely BRDF-independent.

Since the strengths of the approach proposed in [3] are not enough to overcome its limitations, we propose an extension of this method that will alleviate some of them. More precisely we propose to build a Neural Network model that takes in input the transient information of a scene illuminated full field and finds the position of the first discontinuity as if the pixel is illuminated alone. Once the discontinuity locations are defined, the following steps of the *Fermat flow* algorithm can be applied. In this way the acquisition process will be much faster and, at the same time, it would be possible to illuminate the wall with a lot more points in order to retrieve a hidden object reconstruction with a higher resolution. Possibly such a network could also be extended to retrieve also discontinuities after the first one in order to allow the *Fermat flow* algorithm to reconstruct also more complex hidden surfaces.

This method could only work with direct Time of Flight sensors since it requires the transient vector. But in theory, it could also work with the use indirect Time of Flight devices if we use the model described in section 2.3.1 to convert the iToF output into a dToF one. Of course in this specific case will be almost impossible to recover more than the first discontinuity.

6 Implementation

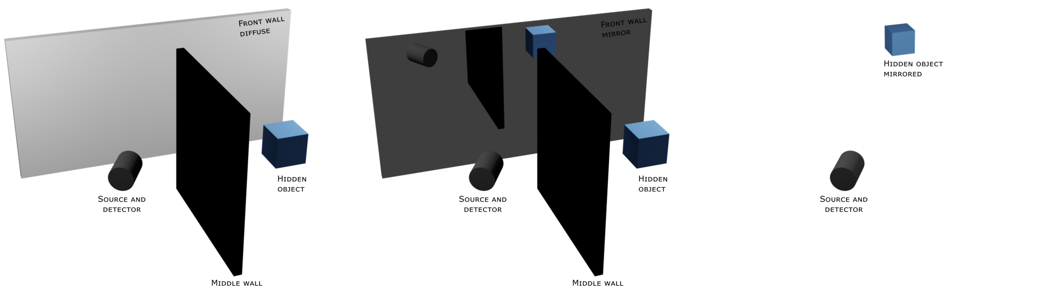
After the discussion about the theoretical background, the generation of the dataset, and the evaluation of *Fermat flow*, is now time to talk about the implementation details of our proposed method. In particular, as anticipated before, the main burden of the Non-Line-of-Sight imaging will be handed by a Neural Network model. For this reason, it is essential to introduce the trick that we have introduced in order to allow the network to learn, together with the actual structure of the architecture. Other than that the following sections will also present the pipeline used to generate the related ground truth.

6.1 "Mirror trick" approach

Due to the complexity of the task, it has been decided to utilize a supervised approach to solve it. This, for sure helps the model to learn faster and more accurately, but on the other hand, requires to have a precisely labeled ground truth used to compute the loss of the network.

That represents one of our bigger constraints since having a good and reliable ground truth in a NLoS environment is not easy. Other than that depending on how we design it also the network structure and output will change consequently. In other words, the problem is to identify what the network should actually give in output. The best scenario would be to have a NN that, given in input raw indirect Time of Flight data, will directly produce as output the depth map of the hidden object as if the point of view is located on the front wall looking in the direction of the object. Of course, this approach is demanding too much from the network. This makes clear that to have such a clean result it is required to introduce some intermediate steps able to reduce the overall complexity of the task and so help the network to learn.

To alleviate the load of the model we introduce the *"Mirror trick"*. If we consider one of the standard scenes (fig. 6.1a) of the dataset described in section 4.2 and we swap the front wall with a perfect mirror (fig. 6.1b), it is now extremely easier to identify the hidden object. Indeed in this particular scenario, the NLoS condition essentially is no more valid, it is almost like going back to a Line-of-Sight scenario. Furthermore, if the used mirror is ideal, given how a ToF camera works, the depth map produced by the sensor in a setup like the one of fig. 6.1b, is the same as the one produced from the setup of fig. 6.1c. Essentially if the front wall is an ideal mirror it is possible to say that the NLoS object appears to the sensor as if it is flipped over the plane $z = 0$.



(a) Basic scene without any changes

(b) Basic scene with a perfect mirror instead of the front wall

(c) Basic scene after the application of the proposed trick

Figure 6.1: Representation of the *"Mirror trick"*

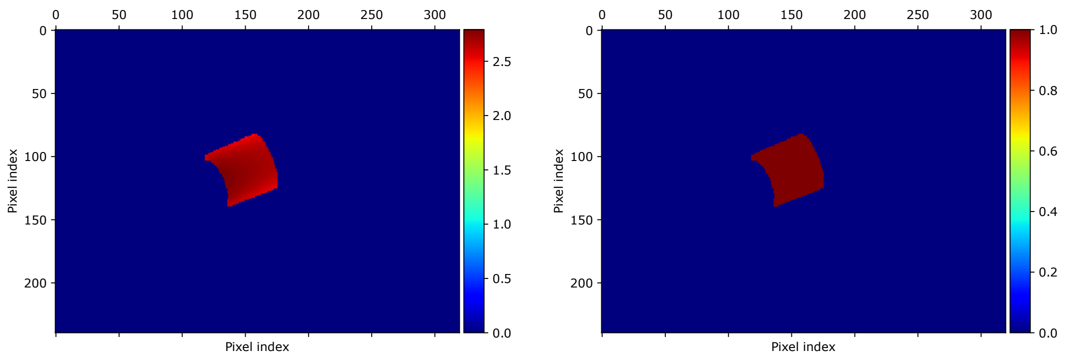
It is, so, possible to both generate the ground truth easily and reduce the complexity of the task that the network needs to solve. At this point, the network will learn a way to convert the given front wall of the scene into a mirror and then extract the depth map of the hidden object.

6.2 How the ground truth is build

Building the ground truth was quite a straightforward process since all the scenes were based on the ones in the dataset and the modification to perform were not many. In particular, to generate the *"mirror ground truth"* it was sufficient to implement the following pipeline:

1. in `Blender`, load the scenes of our dataset then:
   (a) remove the middle wall since it is now unnecessary[1],
   (b) mirrors each hidden object with respect to the front wall,
   (c) remove the front wall,

2. export all the obtained mesh in `.obj` format,
3. generates all the `.xml` files for `Mitsuba 2`,
4. uses the `.obj` and `.xml` files to render all the scenes using `mitsuba2-transient-nlos`.



(a) Depth map (b) Mask

Figure 6.2: Example of a ground truth element

After that, we obtain the transient information of each scene. At this point, it is necessary to compute the depth maps with the related mask. The depth map can be extracted directly from the rendered data using eq. (2.7). From this processing, it is possible to obtain something like fig. 6.2a. Since in the scene setup we use, in the hidden area there is just an object and then nothing else, it is useless to estimate the depth information on all the points that do not belong to the object. For this reason, it could be really useful to have a mask that exactly identifies all the pixels that correspond to the object. In this way, it is possible to use such information to estimate the depth just on the useful points (both during training and inference). To generate the mask it is enough to consider the just generated depth map and set to one all the points where the measured

[1]Thanks to the *"mirror trick"* the NLoS condition decay and so is no more useful to hide the object from the camera

depth is different from zero. This is doable since for how the data is processed all the points where there is no information (all the background points) are set to have a depth equal to $0m$. To better understand how our masks appear it is possible to see an example of a ground truth mask in fig. 6.2b

6.3 Neural network model and implementations

At this point all the elements required to train our model have been developed, it remains just to properly define how the Convolutional Neural Network (CNN) is structured.

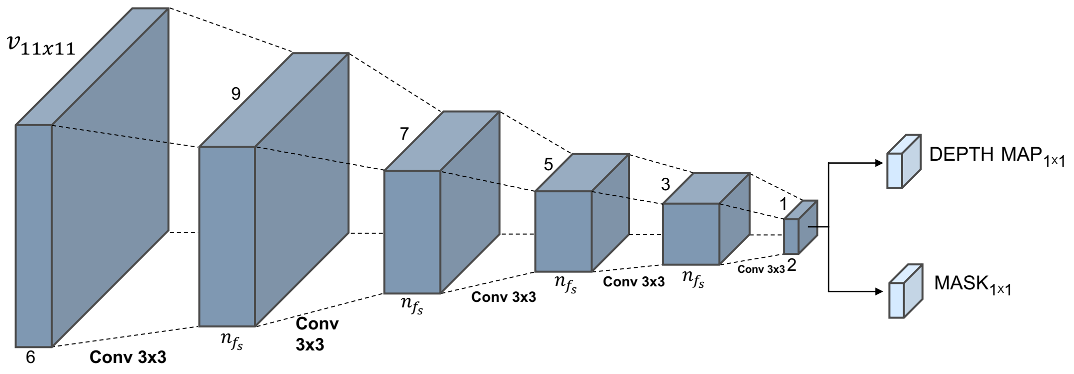


Figure 6.3: Model representation of the used Convolutional Neural Network

As it is possible to see in fig. 6.3 we have designed a CNN characterized by five `2D Convolutional` layers with a filter size of 32. The network is designed to receive in input raw iToF data acquired using three different frequencies: $20MHz$, $50MHz$ and $60MHz$[2]. Since the input data are in the order of the megahertz (10^6) they are not compliant with a NN. For this reason, before feeding the data to the network we have performed normalization to map the data in the range $[0, 1]$. The normalization is performed by dividing elementwise the input by the measured amplitude at $20MHz$. Regarding the output, as anticipated in section 6.2, the network produces two different elements: a depth map and a mask estimation.

[2]This justifies the fact that the input in fig. 6.3 has six layers, one cosine and one sine for each used frequency

6.3.1 Training phase

During the training phase the network work on batches of 8192 patches of 11×11 pixels each. Since the network is designed to learn from patches and not from the full image it is required to extract from the original dataset patches of the image. This task is extremely delicate since the data provided to the network must be balanced. By balanced we mean that the network should see the same number of pixels coming from an object and the background. In this way, it is possible to maximize the possibility of the network properly learning both classes (background and object). This is mainly important for the mask computation that, after all, is essentially a binary classification.

The patches extraction procedure works by sampling from the full image 400 background pixels and 400 object's pixel. From this initial sampling is possible to build 800 different patches for each dataset's scene. We stick to 800 maximum patches for each image in order to ensure that in each case we will be able to sample the same number of background and object elements.

In order to generate the final input data, the original dataset presented in chapter 4 (or only some of its groups) is loaded, split in train (60%), validation (20%) and test (20%) set[3]. Then just from the first two sections, the patches (for the test set is not useful) are extracted. Finally, each sample (both patches or images) is linked to the correspondent element of the "*mirror ground truth*". Only at this point, it is possible to train the model

Given an input of 11×11 processed through five `TensorFlow 2.3.0` [17] `Conv2D` layers the output will be a 1×1 matrix. More precisely the output will be of size $1 \times 1 \times 2$ since there is one layer representing the estimated depth and one the estimated mask. On this output is then computed the loss. We have decided to use as a loss function the MAE. It is important to point out that for the mask the loss is always computed while for the depth map prediction it is computed only if the ground truth mask in that pixel is set to 1. In this way, the loss of the depth map is computed only where it is significant.

The training is performed over 100000 epochs using the `Adam optimizer` [18] with a learning rate of $1 \cdot 10^{-3}$.

[3]The split between the different subset is performed randomly

6.3.2 Inference phase

The inference phase is much simpler than the training one since it is not required to split the data into 11×11 patches but it is sufficient to directly provide the 320×240 full iToF measurement. The network will directly provide the full depth map and mask. Of course, since the data are based on the "*mirror trick*" to have the proper result it is required to flip the z coordinates of the estimated depth map.

It is important to notice that all the results that will be discussed in chapter 7 have been produced doing inference on the test set.

7

Results & comparisons

After the definition of the dataset, the designing of the ground truth, and the description of the network model, it is now time to evaluate the results obtained by using the NN model proposed in chapter 6.

In this chapter, it will be presented some sample results obtained using the novel approach to Non-Line-of-Sight imaging proposed in the previous sections.

In order to give the most comprehensive review of the work, the result analysis will be split into four different sections. This is justified by the need of evaluating the network performance in all the different scenarios contained in the dataset, and at the same time understand which elements are more challenging for the network. This setup for the test is facilitated by the fact that from the beginning the dataset was generated in a modular way. Indeed in order to perform the intended tests, it is enough to perform inference on just one, or a couple, of the different modules of the dataset[1]. In particular, it has been decided to perform the tests on the following four modules:

1. TEST CASE A: fixed sensor location and rotation looking at a perfectly white and diffuse wall,
2. TEST CASE B: variable sensor position and location looking at a perfectly white and diffuse wall,
3. TEST CASE C: fixed sensor location and rotation looking at a wall with different roughness values,

[1]See the last part of section 4.2 to have more details on the different groups

4. TEST CASE D: fixed and variable sensor location and rotation looking at a wall with different roughness values.

Of course in all four test scenarios the hidden object is always free to move inside the hidden scene.

Due to the type of results that our model generates, in order to properly understand the performance of the network, it was essential to find a way of visualizing the results that were at the same time clear and exhaustive. It was fundamental to use some kind of visualization that allows both quantitatively and qualitatively evaluation and comparison of the produced output with respect to the ground truth data. Keeping all of that in mind we decided to use for each tested sample two representations: the first one is characterized by a set of plots that quantitatively compare the obtained results with the baseline while the second one is represented by a set of images of the reconstructed point clouds that helps to qualitatively evaluate the results. Other than that in the plots there is also the computation of the MAE between the prediction and the ground truth that could be used as a metric to evaluate the overall performances.

The first kind of representation (figs. 7.1, 7.3, 7.5 and 7.7) is structured as a matrix of plots evaluating all the outputs of the network. Starting from the left, the first column represents the ground truth, the second one the prediction, and the last one is the difference between the previous two. While starting from the top, the first row is related to the predicted depth map, masked using the ground truth mask, the second one presents the predicted depth map masked by the predicted mask and the last one represents the predicted mask alone.

On the other hand, the second type of results' representation (figs. 7.2, 7.4, 7.6 and 7.8) is presented as a set of four images. The one in the top left corner represents the ground truth point cloud, the one on its right shows the point cloud obtained by using the predicted depth map and the ground truth mask, while the one on the bottom left corresponds to the point cloud actually predicted by the model (without any ground truth). Finally, the image on the bottom right shows the difference between the two point clouds of the first row, in blu the ground truth and in yellow the prediction.

For all the presented test cases it will be used both types of representation in order to properly discuss the obtained results and perform a complete evaluation of the performance of the model.

In the following sections, there will be an in-depth analysis of the performance of the proposed NN model in each of the considered test scenarios. All the scenes are based on the one in fig. 4.3. Other than that in appendix A are contained some other scene examples to prove that what will be discussed in sections 7.1 to 7.4 represents not only a lucky example but a full description of the architecture performances.

7.1 Test case A: fixed sensor and diffuse wall

Here, we have tested the network on a limited portion of the dataset that contains only samples characterized by the sensor fixed in the base location (no translation nor rotation), and a front white wall perfectly diffuse.

In order to discuss the results, it will be used a sample scene extracted from the related test set. More precisely, in this case, the setup is characterized by:

- sensor: fixed
- object:
  - shape $\rightarrow$ concave plane,
  - position $\rightarrow (x: \ 0.9, \ y: \ 0.5, \ z: \ 1.25)m$,
  - rotation $\rightarrow (x: \ -75°, \ y: \ -6°, \ z: \ 5°)$.

This scenario represents the simplest one of all the considered ones since it has fewer variable elements. Indeed essentially only the hidden object represent a variable. For this reason, it is expected to register the best performance of the model.

As it is possible to see from the bottom row of fig. 7.1 the network is able to identify the general location of the hidden object in the space, but it is definitely not able to recover its specific structure. Indeed the obtained reconstruction is somehow of a non-defined shape that roughly surrounds the object area. This behavior is consistent throughout all the reconstructions performed on the test set and so it represents a clear pattern in the mask estimation task. It is still important to notice that even if the reconstruction is not perfect the measured MAE between the ground truth mask and the predicted one is quite low, with a value of 0.04. Since the MAE is used also as the loss function during the training and validation of the network, the fact that is low also on the test set ensures that the network was actually able to learn the task considering the used metrics. Either way, the obtained reconstruction is far from the desired quality this probably indicates that the used loss function is not enough to properly

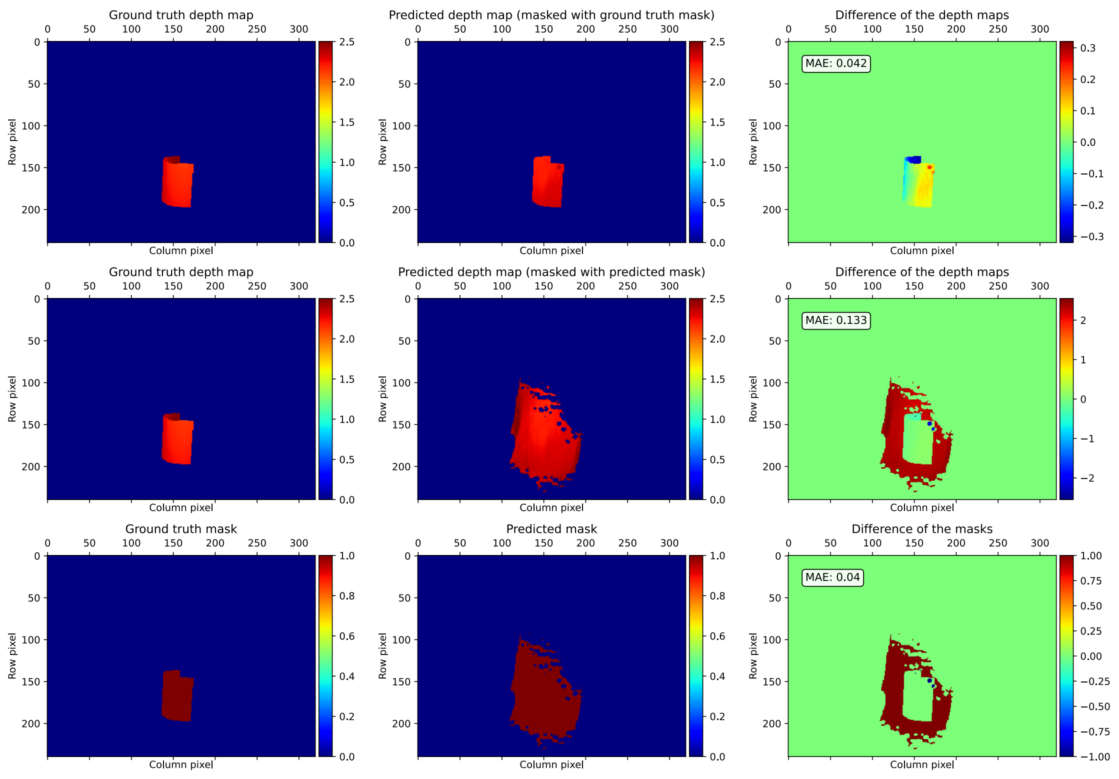
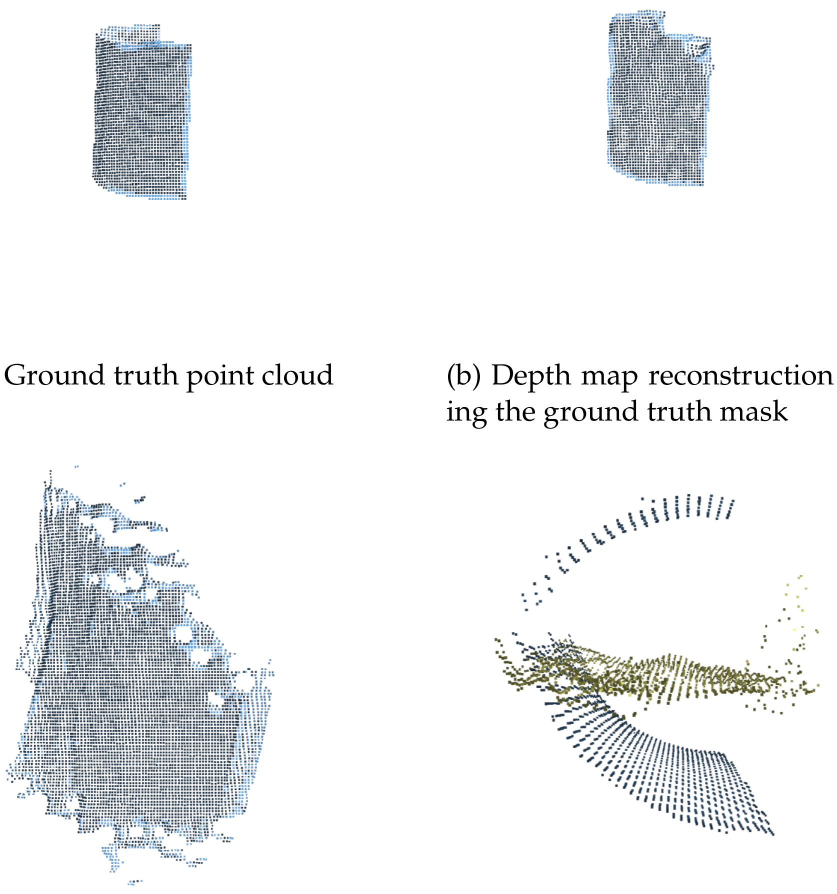


Figure 7.1: Results evaluation (TEST CASE A)

punish the network. This limits the model's capability of understanding the aim of the job to just recover the general location of the hidden object instead of the full shape.

Moving to analyze the middle row of fig. 7.1, where it is considered the full prediction of the network (mask and depth map together), it is possible to notice that the biggest source of errors in the overall reconstruction is represented by the estimated mask. This is particularly clear if such results are compared with the one on the top row. Considering both the MAE value and the color bars linked to the plots it is evident that inside the ground truth hidden object's area the depth estimation works reaching values close to the real one. On the other hand, the estimation of the depth corresponding to the pixels located outside of the ground truth mask is not so good. This behavior on the mislabeled pixels is expected since, as described in chapter 6, the loss function of the depth map prediction is updated only on target points, so only on the one belonging to the object. Supporting this consideration there is the MAE computed between the ground truth reconstruction and the predicted one masked by the ground truth mask. In this case, it assumes a value of $0.042m$ confirming that the depth estimation (ignoring the mask) is quite good. This level of performance is constant throughout the whole test set, where the value of the MAE, using the ground truth mask is always low despite the different hidden object types and locations.



(a) Ground truth point cloud

(b) Depth map reconstruction using the ground truth mask

(c) Predicted depth map using the predicted mask

(d) Comparison between fig. 7.2a (blu) and fig. 7.2b (yellow)

Figure 7.2: Reppresentation of the reconstructed point cloud and comparison with the ground truth (TEST CASE A)

From the plots of fig. 7.1 it is possible to exhaustively evaluate the mask estimation but not the depth one since it is quite cumbersome to visualize the actual reconstruction and understand if the shape of the object over the x axes is properly recovered or not. In order to better analyze this aspect, it is required some sort of 3D representation of the obtained point cloud. This is addressed by fig. 7.2. Unfortunately, as it is possible to see in fig. 7.2d, the predicted depth map (yellow) is not able to recover the shape of the object but it is just able to extract the average distance between the wall and the target. This behavior is similar to the one of the mask estimation. So, it seems that in general the proposed architecture is able to correctly and accurately recover only general and average information about the hidden object. On the other hand, it is completely unable to recover the fine details represented by the specific shape of the target.

Other than that comparing between each other figs. 7.2a to 7.2c it is possible to confirm what said before while analyzing the plots of fig. 7.1, the complete prediction completely lack information about the specific object apart from the general location and distance from the wall. For this reason, the reconstructed point cloud assumes a non-defined shape that has no resemblance to the original hidden object. Comparing figs. 7.2b and 7.2c it is evident that the main weaknesses of the proposed NN is represented by the estimation of the mask.

To wrap up the analysis on the TEST SET A it is possible to say that considering a scenario in which the only variable is represented by the hidden object the considered model can reliably identify the overall location of it inside of the hidden area (other all the three axes), but it is not able to recover finer information about the specific object shape.

7.2 Test case b: variable sensor and diffuse wall

This time has been introduced a second variable concerning TEST CASE A, indeed in this setup also the sensor can assume different locations and orientations. The wall has still the same property as before. This test has been performed to see if the network is able to generalize over a more difficult setup in which the point of view changes scene by scene. Doing that also the relative area of the Field of View occupied by the hidden object changes much more than in the previous test. So this setup could be a good candidate to verify if the network can locate the object also if it falls into a completely different spot with respect to the standard one.

For the analysis, it has been considered a scene of the test set where:

- SENSOR:
  - position $\rightarrow (x: 1.2,\ y: -1.3,\ z: 1.7)m$,
  - rotation $\rightarrow (x: 88°,\ y: 2°,\ z: 81°)$,

- OBJECT:
    - shape $\longrightarrow$ concave plane,
    - position $\longrightarrow (x:\ 1.2,\ y:\ 0.7,\ z:\ 1.75)m$,
    - rotation $\longrightarrow (x:\ 72°,\ y:\ -69°,\ z:\ -34°)$.

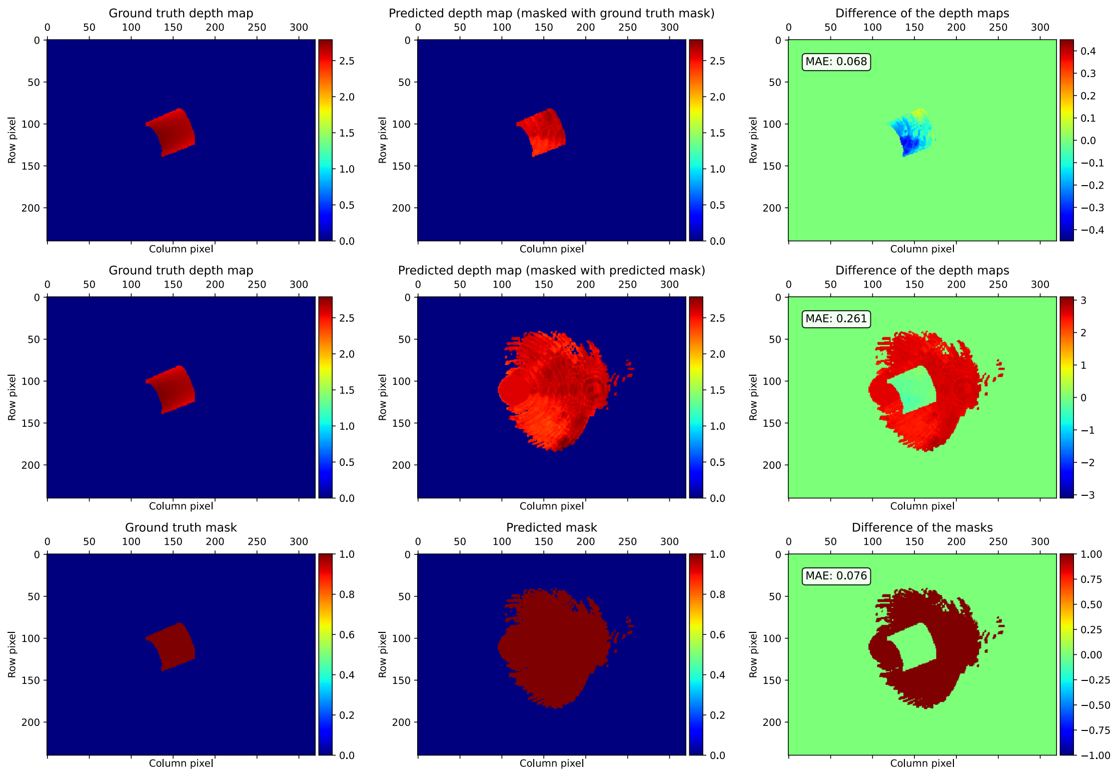
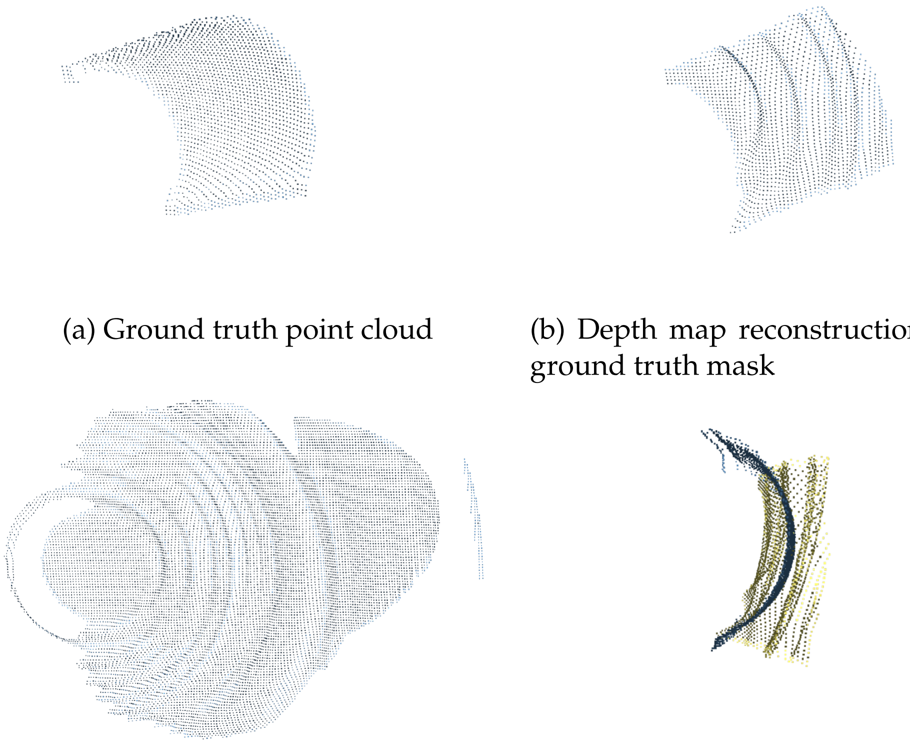


Figure 7.3: Results evaluation (TEST CASE B)

In this scenario, looking at the bottom row of fig. 7.3, it is possible to notice that the results are definitely rougher. Comparing this result with the one obtained in section 7.1 it is clear how the predicted mask identifies a much bigger and unprecise area. The network seems to still be able to correctly identify the location of the hidden object, but this time, with a much bigger uncertainty. To confirm that, also the computed MAE is bigger with a value of 0.076[2]. On the other hand considering the first row of fig. 7.3 it is possible to notice that the performance of the network on the depth estimation task is quite similar to the one obtained in the simpler case of TEST CASE A. While if we consider the middle row of the same figure it is possible to see that the performance in this scenario

[2]For the Mean Absolute Error (MAE), the bigger it is the worst it is

is even more affected by the suboptimal estimation of the mask. In fact, since the area of misleading points is bigger also the area with wrongly estimated depth is now bigger


(a) Ground truth point cloud

(b) Depth map reconstruction using the ground truth mask

(c) Predicted depth map using the predicted mask

(d) Comparison between fig. 7.4a (blu) and fig. 7.4b (yellow)

Figure 7.4: Reppresentation of the reconstructed point cloud and comparison with the ground truth (TEST CASE B)

Considering fig. 7.4, in particular, fig. 7.4d is clearly visible that regardless of the worst performance over the mask estimation the network was still perfectly able to retrieve the average distance between the hidden object and the front wall. While from fig. 7.4c it is even more obvious that the poor performance over the mask estimation greatly decreases the overall performance of the network.

To sum up, the results obtained in the TEST SET B it is possible to say that the proposed NN model was not fully able to generalize over a freely located sensor. It is still able to find the object but with greatly reduced accuracy.

7.3 Test case c: fixed sensor and rough wall

This time around it has been considered a situation similar to the one of test case a but this time the new variable is represented by the material of the front wall that is still white, but no more perfectly diffuse. This time it is characterized by a material that can have a random roughness value (in the range [0.3, 1]). This test setup has been introduced to verify if the network is able to generalize over different types of surfaces.

For the analysis, it has been considered a scene of the test set where:

- sensor: fixed,
- object:
  - shape → concave plane,
  - position → $(x:\ 0.9,\ y:\ 0.6,\ z:\ 1.35)m$,
  - rotation → $(x:\ -13°,\ y:\ 33°,\ z:\ 65°)$,
- wall roughness: 0.45.

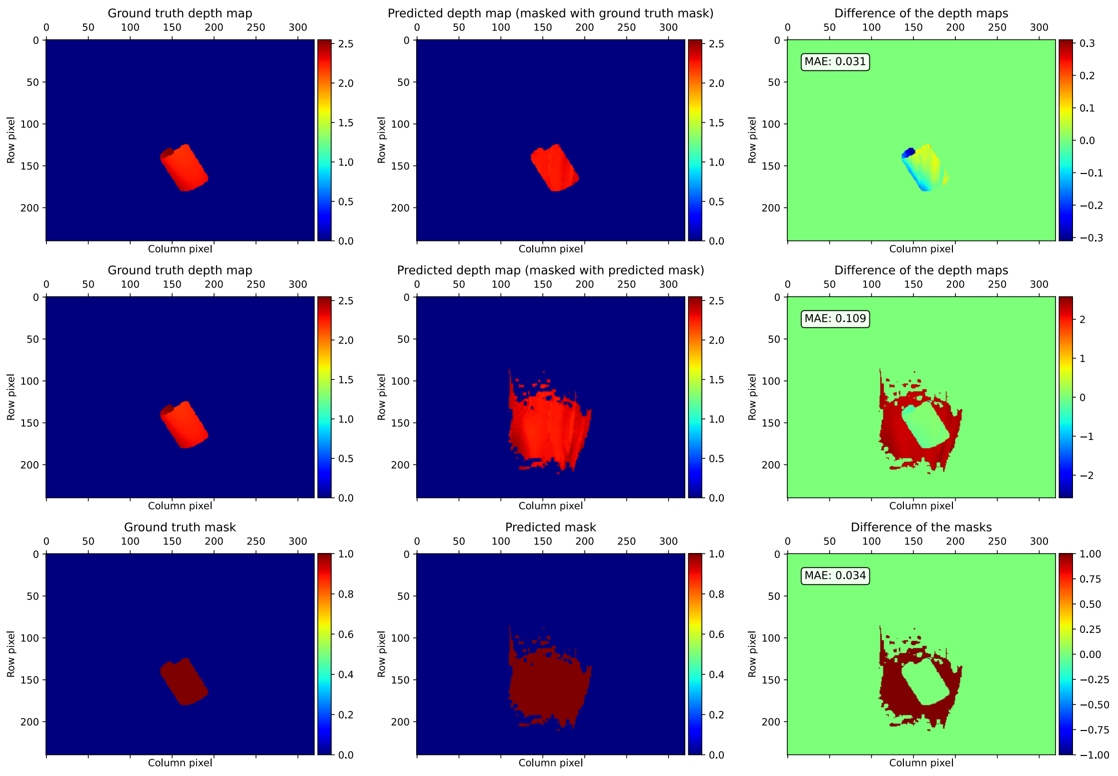
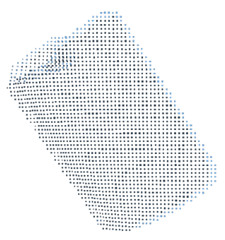
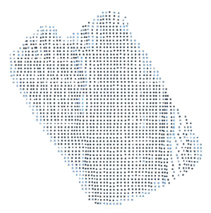
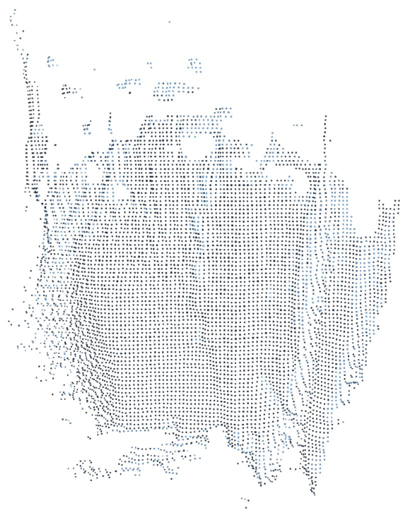


Figure 7.5: Results evaluation (test case c)

This test setup is much similar to the one discussed in section 7.1 since the only difference is represented by the varying roughness of the wall. For this reason, if the model is able to generalize it is reasonable to expect results similar to the one of TEST CASE A.


(a) Ground truth point cloud


(b) Depth map reconstruction using the ground truth mask


(c) Predicted depth map using the predicted mask

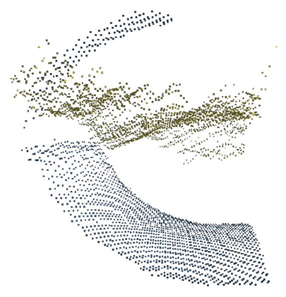

(d) Comparison between fig. 7.6a (blu) and fig. 7.6b (yellow)

Figure 7.6: Reppresentation of the reconstructed point cloud and comparison with the ground truth (TEST CASE C)

Thankfully considering the bottom row of fig. 7.5 it is possible to see that also in this case the network can recover the area where the hidden object is located. This area is quite close to the overall area of the target guaranteeing a MAE of 0.034. From these latest considerations, it is possible to say that concerning the mask estimation the network is able to generalize to a random front wall material without too many problems reaching performance really close to the one of section 7.1. The only downside generated by the different materials seems to be the fact that the edges of the predicted mask area are a bit more rough and imprecise.

Considering fig. 7.6 it is possible to confirm the same behavior noticed over the comparison plots since the reconstructed point cloud has a quality similar to the one obtained under the simpler condition of TEST SET A. Also in this case the estimated depth only captures the average depth of the object but not its shape. Looking at fig. 7.6c it is possible to confirm that still the biggest limitation of the network is represented by the mask estimation.

To conclude the results obtained under the specific conditions of TEST SET C confirm that the proposed NN model is able to generalize quite well over a variable front wall material.

7.4 TEST CASE D: FIXED & VARIABLE SENSOR AND ROUGH WALL

This final scenario represents the hardest one among the considered ones. In this case, the sensor can be either fixed to the basic position or assume a random location and orientation, and at the same time, the front wall is not fixed to be perfectly diffuse but can assume a random roughness value as in TEST CASE C.

For the analysis, it has been considered a scene of the test set where:

- SENSOR:
  - position $\rightarrow (x: \ 1.0, \ y: \ -1.0, \ z: \ 1.65)m$,
  - rotation $\rightarrow (x: \ 90^\circ, \ y: \ 0^\circ, \ z: \ 50^\circ)$,
- OBJECT:
  - shape $\rightarrow$ concave plane,
  - position $\rightarrow (x: \ 0.9, \ y: \ 1.0, \ z: \ 1.75)m$,
  - rotation $\rightarrow (x: \ -89^\circ, \ y: \ -22^\circ, \ z: \ 0^\circ)$,
- WALL ROUGHNESS: 0.55.

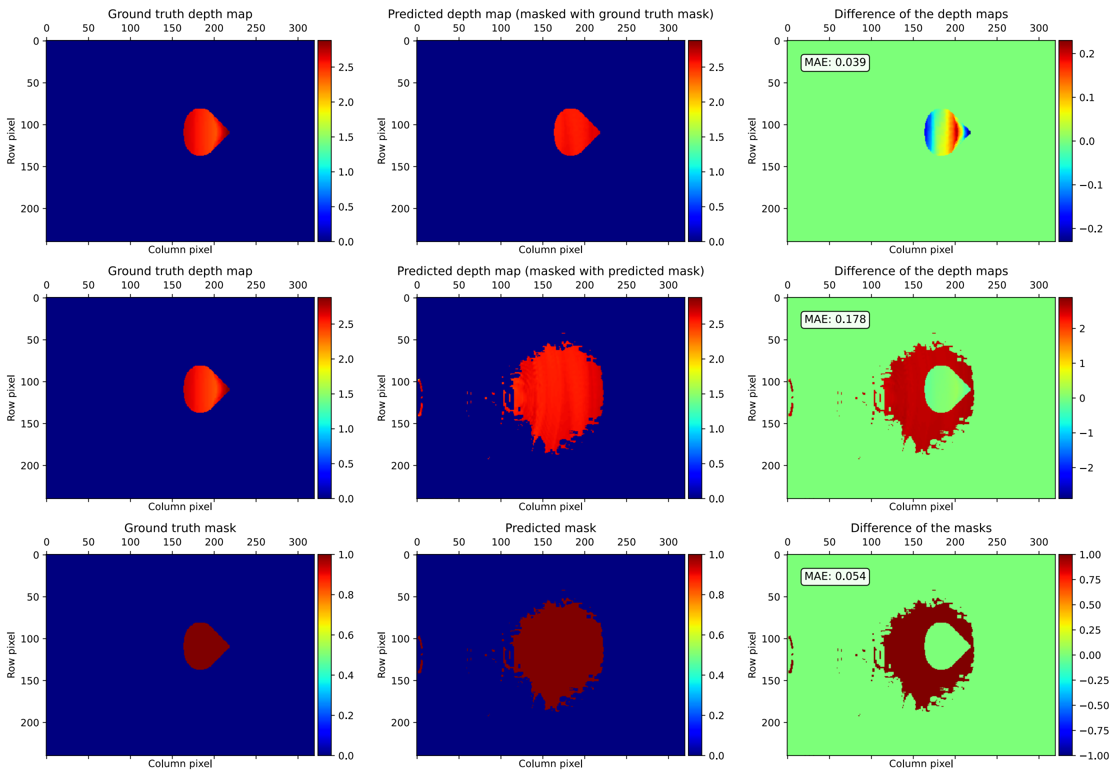
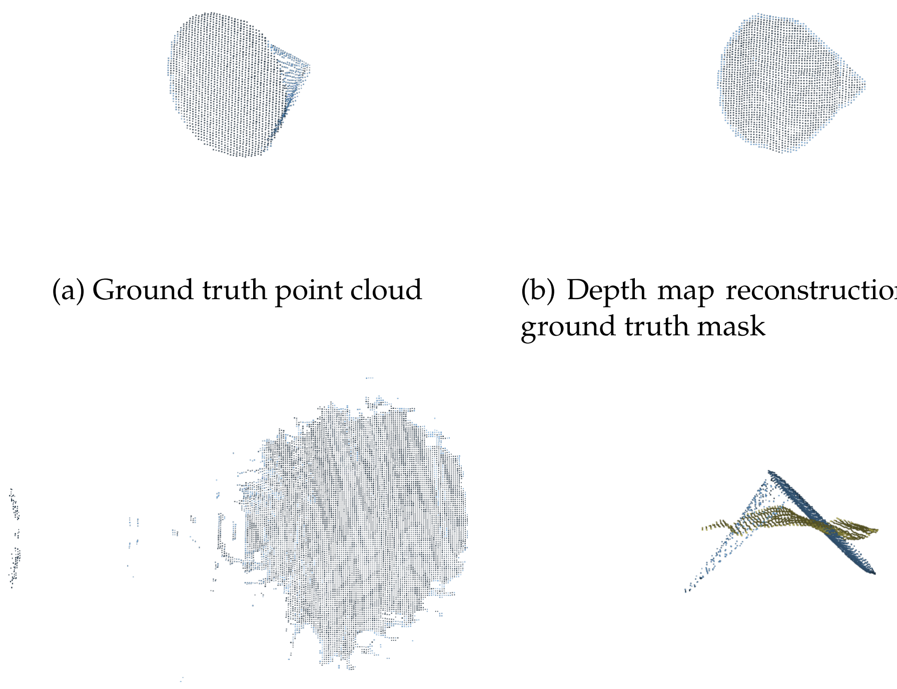


Figure 7.7: Results evaluation (TEST CASE D)

Since this scenario represents the most difficult one of the considered ones, it is expected that the performance will be affected in a negative way. Unexpectedly, looking at the bottom row of fig. 7.7, it is possible to see that regardless of the increased variability of the setup the network was able to generalize quite well regarding the mask prediction. The predicted mask is comparable to the one presented in section 7.1 obtaining a MAE just a little bit worst, with a value of 0.054. The generalization ability of the proposed architecture seems to extend also over the depth estimation task. Also under this scenario, the network was able to reach a MAE between the ground truth prediction and the depth map prediction masked by the ground truth mask of 0.039m (top row of fig. 7.7), comparable with the previous results.

From what has just been said, it seems that under the conditions of TEST SET D the network can generalize over both tasks (mask and depth map prediction). To verify that it is possible to look at fig. 7.8 where it is possible to see that apart from a bit of noise on the left side of fig. 7.8b the overall quality of the reconstruction is similar to the one presented in section 7.1. Other than that the performance regarding the depth map computation is coherent with all the



(a) Ground truth point cloud

(b) Depth map reconstruction using the ground truth mask

(c) Predicted depth map using the predicted mask

(d) Comparison between fig. 7.8a (blu) and fig. 7.8b (yellow)

Figure 7.8: Reppresentation of the reconstructed point cloud and comparison with the ground truth (TEST CASE D)

other cases. In fig. 7.8d it is clearly visible that also here the network is able to recover the average depth of the considered hidden object.

To conclude, also in TEST CASE D, regardless of the added complexity the proposed architecture was able to recover the overall location and depth of the hidden object.

8 Conclusions and Future Works

In order to give a final evaluation of the project, it is important to keep in mind that it represents a feasibility study and so the beginning of a much longer process that will bring this technology to its final state.

That said, in this work has been performed a lot of preliminary work that will represent the foundations for all future works. In particular, all the testing performed over the various version of `Mitsuba 2` (chapter 3) represents an extremely valuable element that guarantees the reliability of this open-source renderer for other research studies. Other than that another essential contribution of this project is the generation of a novel dataset for Non-Line-of-Sight imaging in a "*look around the corner*" scenario (chapter 4), The introduction of this dataset represents quite a big contribution since from our knowledge there is no other alternative in the literature that can be used for such task. Also, the evaluating, testing, and extension of the *Fermat flow* algorithm could represent another big contribution to the development of NLoS imaging. In chapter 5 were pointed out all the limitations of this approach together with some proposals for improvements that aim to overcome the biggest drawbacks of this method in order to take full advantage of its biggest strength, the fact that it is completely BRDF-invariant and relays only on the geometric properties of the scene.

All the things just discussed represent for sure a useful contribution but the core of the project is, still, the Neural Network implementation proposed in chapter 6. It has been introduced a novel approach for NLoS imaging based on the "*mirror trick*", together with a related NN model. Our implementation proves that using an indirect Time of Flight is possible to recover objects located outside of the Line-of-Sight of the sensor. The results presented in chapter 7 highlight that the task is for sure possible but to obtain accurate results it is necessary to further investigate this topic by designing a more robust NN architecture. Doing that most probably it will be possible to reliably and precisely recover useful information from the hidden scene using a simple and cheap sensor as the iToF.

From the above considerations, it is clear that this topic requires much more studies in order to reach a usable state. At the same time from the work proposed in this feasibility study, it is clear that with some tweaks it is, for sure, possible to at least recover a simple shape located behind a corner.

To bring forward this project the first step will be for sure to use a more complex network architecture able to extract finer information from the raw iToF data. A good way to do so could be to use a much bigger patch during the training to allow the network to learn much more spatially correlated features. Other than that it is also necessary to use a more complex network architecture for both the mask and depth estimation. In our opinion, a good option could be to use a U-Net [19]. Other important improvements that could greatly improve the ability of the model of recovering details are to change the used loss function, in fact, the Mean Absolute Error was able to force the network to learn just average information about the hidden object ignoring all its finer details.

Other than improving the model that has been introduced in chapter 6 could be really interesting to evaluate the proposed Neural Network extension of the *Fermat flow* proposed in section 5.3. Given the interesting property of the approach proposed in [3], such as the BRDF invariance, if the network model will be able to overcome all the limitations discussed in section 5.2 the obtained hybrid solution could represent a really valuable solution to the NLoS imaging task for "*look around the corner*" scenario. If properly implemented this method could then also be compared to the enhanced "*mirror trick*" solution.

Overall it is possible to say that this feasibility study gained results able to prove that the aimed task is doable, it just requires some more investigation. Performing some additional work in this fieldì, starting from the contributions introduced here it will be finally possible to use this technology for all the promising applications described in section 1.2.

A

Additional test results

A.1 Test case A: fixed sensor and diffuse wall

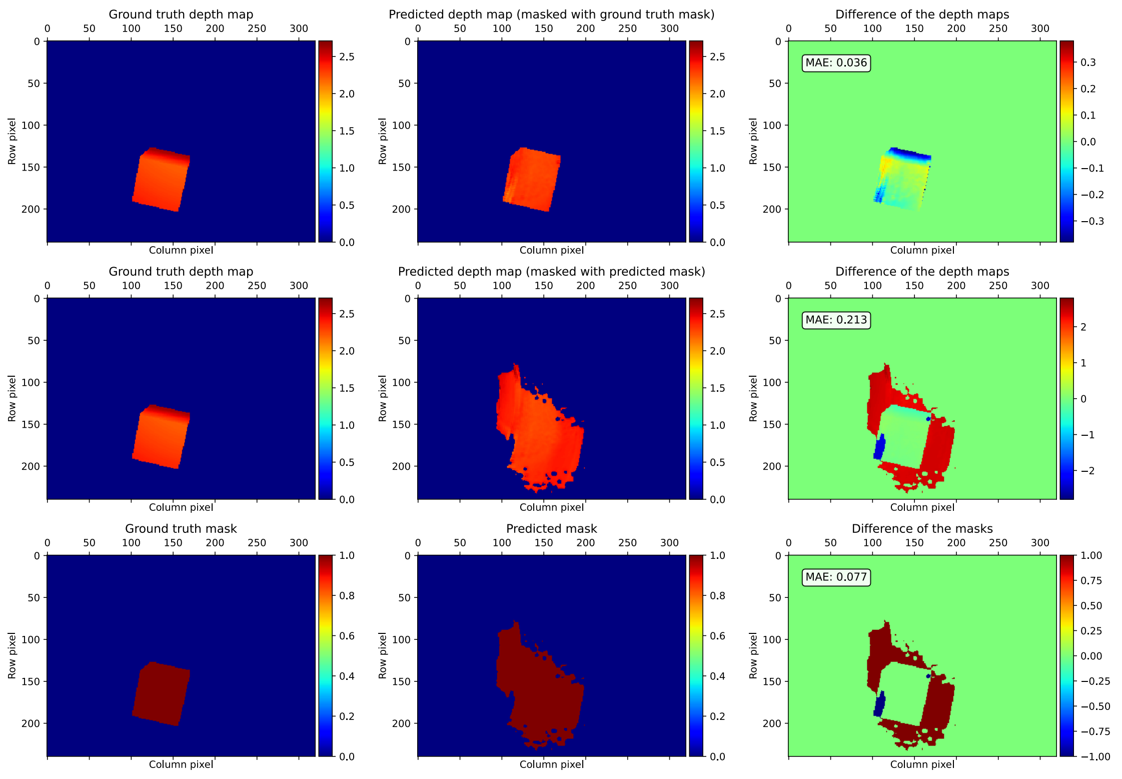


Figure A.1: sensor location: (x : 1.0, y : −1.0, z : 1.65), sensor rotation: (x : 90°, y : 0°, z : 50°) | | object shape: cube, object location: (x : 1.1, y : 0.5, z : 1.25), object rotation: (x : −52°, y : 74°, z : −8°)

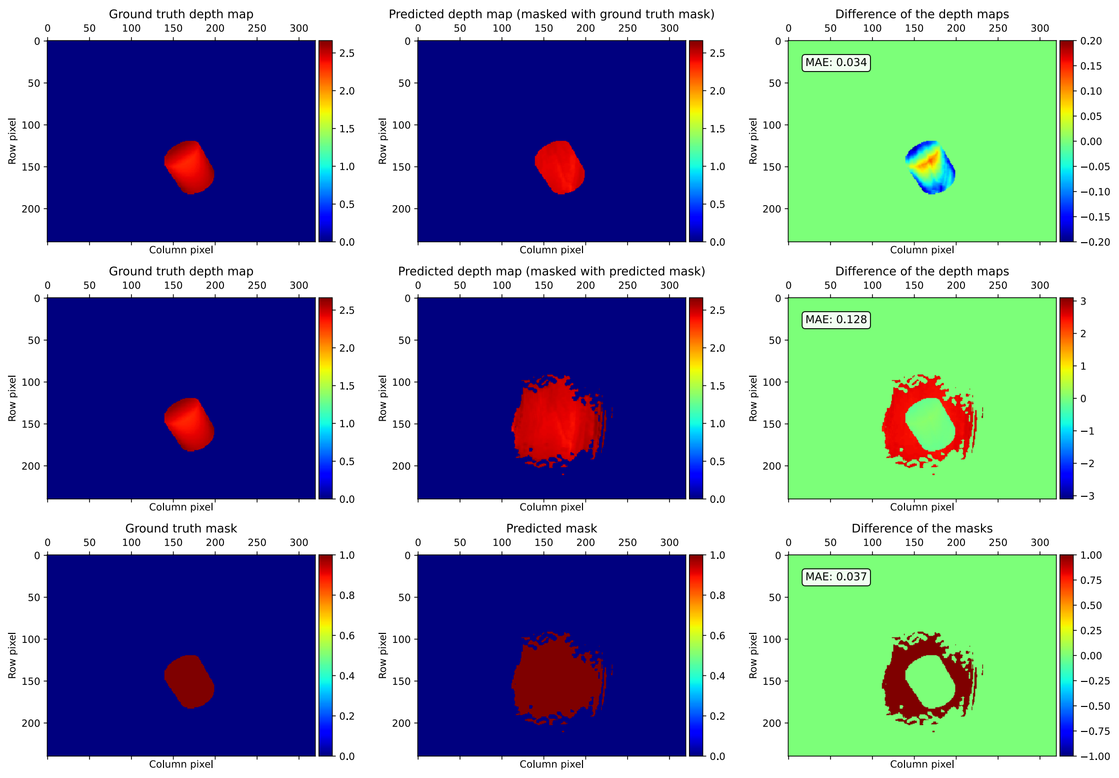


Figure A.2: sensor location: $(x: 1.0,\ y: -1.0,\ z: 1.65)$, sensor rotation: $(x: 90°,\ y: 0°,\ z: 50°)$ | | object shape: cylinder, object location: $(x: 1.0,\ y: 0.8,\ z: 1.35)$, object rotation: $(x: 40°,\ y: -4°,\ z: 0°)$

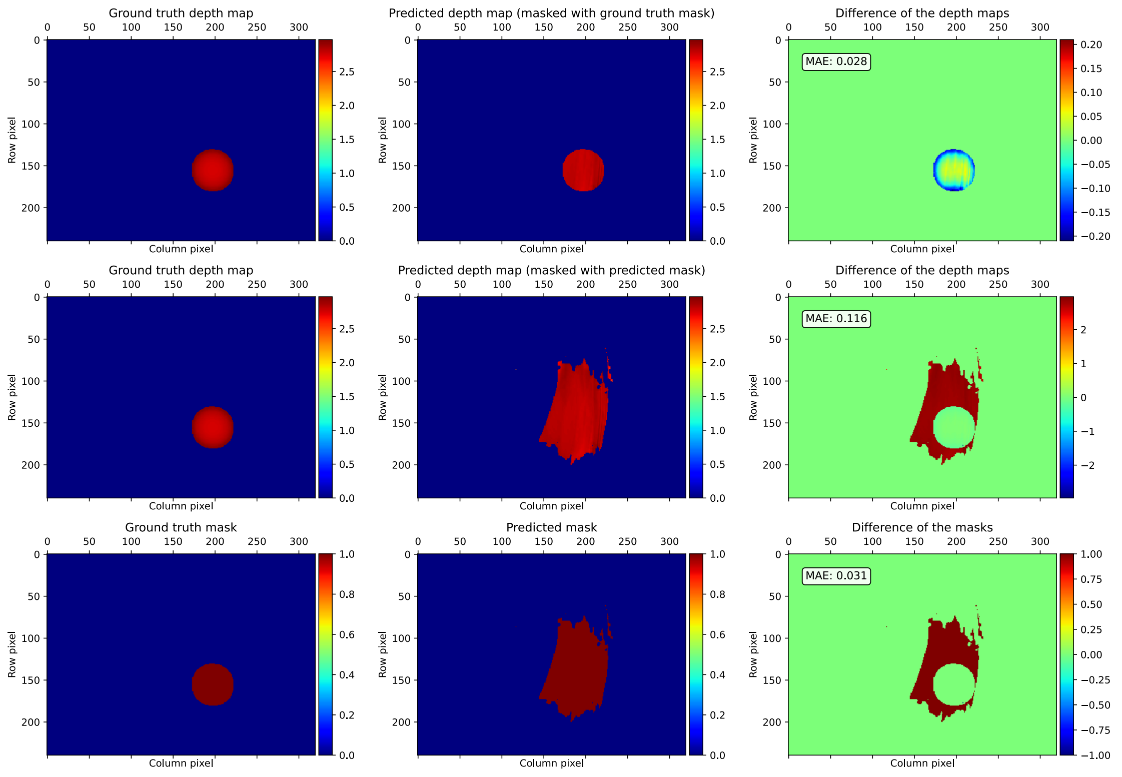


Figure A.3: sensor location: $(x: 1.0, y: -1.0, z: 1.65)$, sensor rotation: $(x: 90°, y: 0°, z: 50°)$ | | object shape: sphere, object location: $(x: 1.1, y: 1.3, z: 1.25)$, object rotation: $(x: 0°, y: 0°, z: 0°)$

A.2 Test case b: variable sensor and diffuse wall

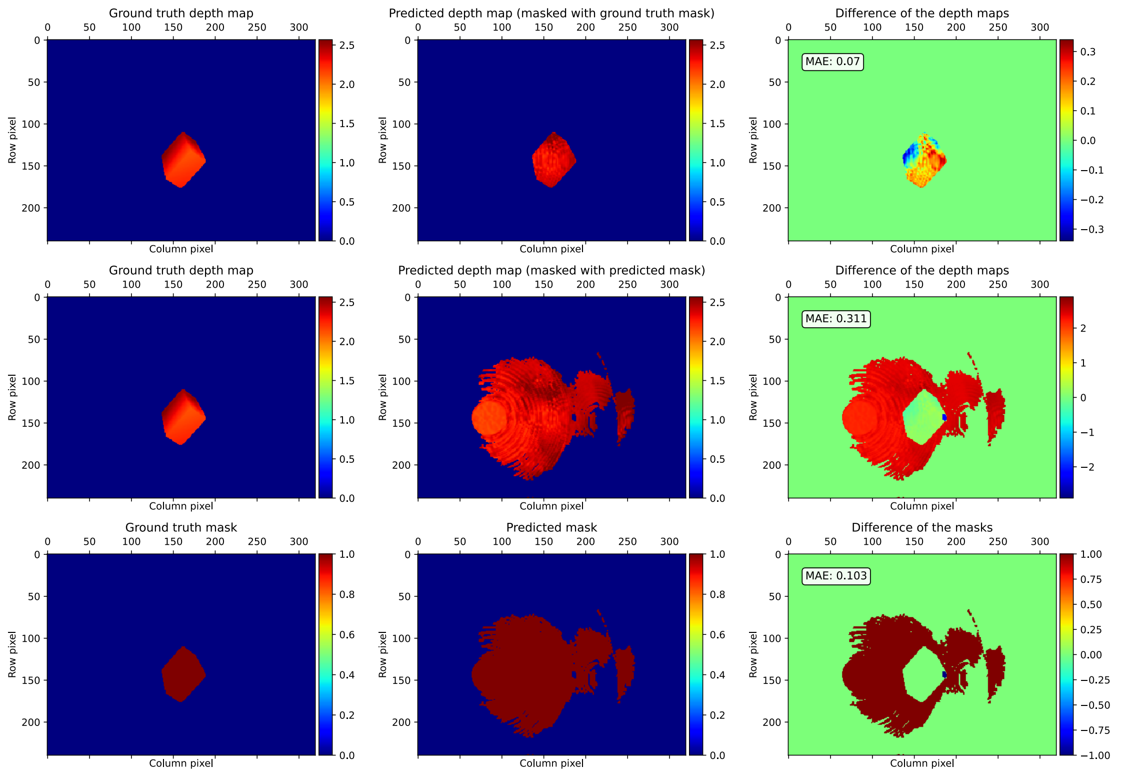


Figure A.4: sensor location: $(x : 1.0, y : -1.3, z : 1.5)$, sensor rotation: $(x : 94°, y : 4°, z : 75°)$ || object shape: parallelepiped, object location: $(x : 0.9, y : 0.6, z : 1.45)$, object rotation: $(x : 27°, y : -34°, z : -76°)$

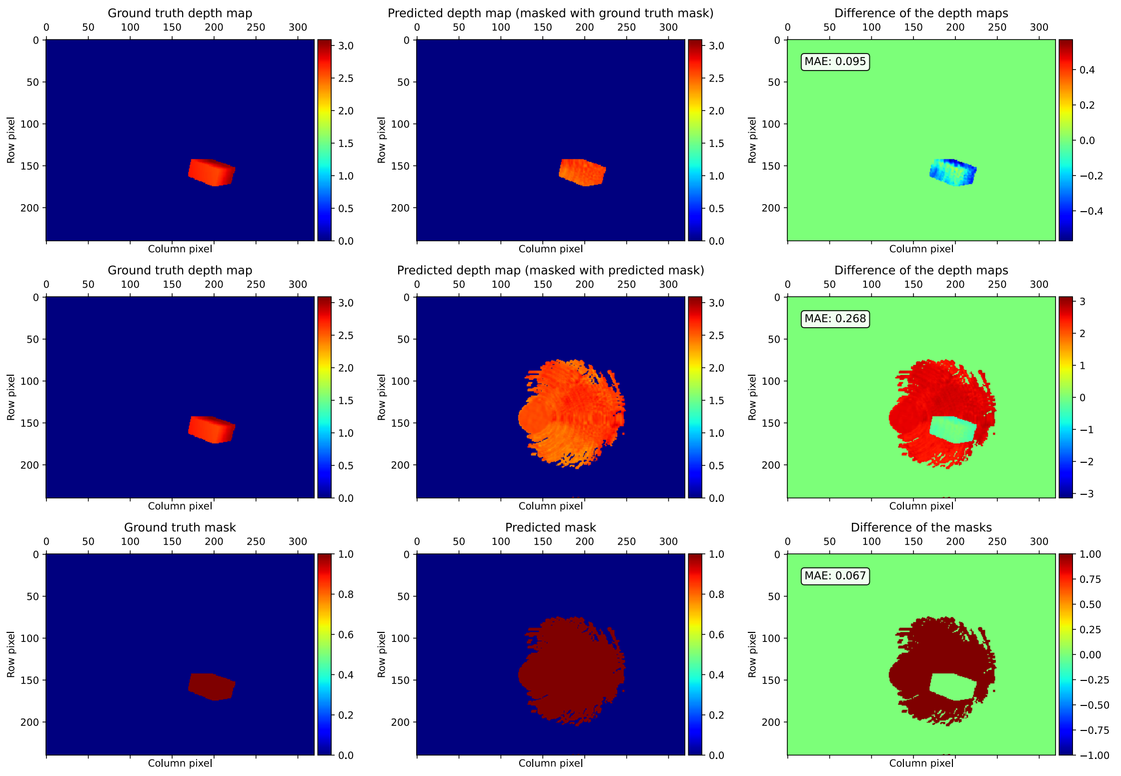


Figure A.5: sensor location: (x : 1.1, y : −1.3, z : 1.7), sensor rotation: (x : 95°, y : 1°, z : 86°) | | object shape: parallelepiped, object location: (x : 1.1, y : 0.5, z : 1.25), object rotation: (x : −52°, y : 74°, z : −8°)

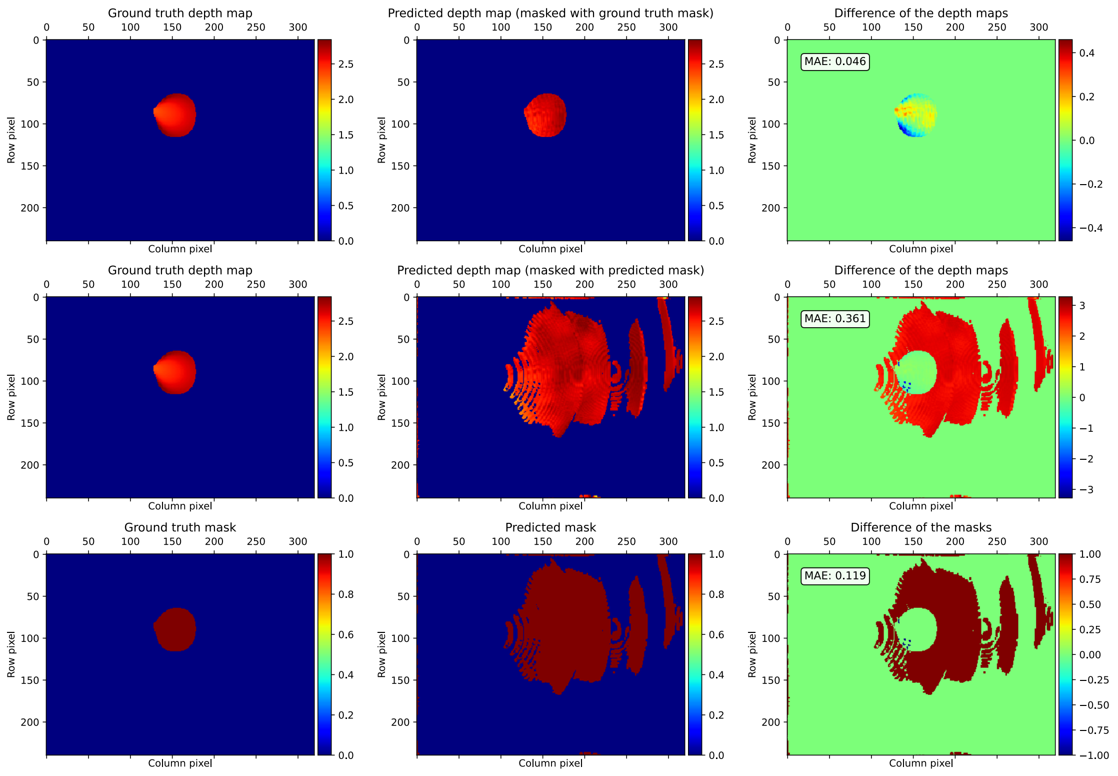


Figure A.6: sensor location: (x : 1.2, y : −1.1, z : 1.5), sensor rotation: (x : 85°, y : −1°, z : 75°) | | object shape: cone, object location: (x : 1.2, y : 0.7, z : 1.95), object rotation: (x : 64°, y : −90°, z : 0°)

A.3 Test case C: fixed sensor and variable wall

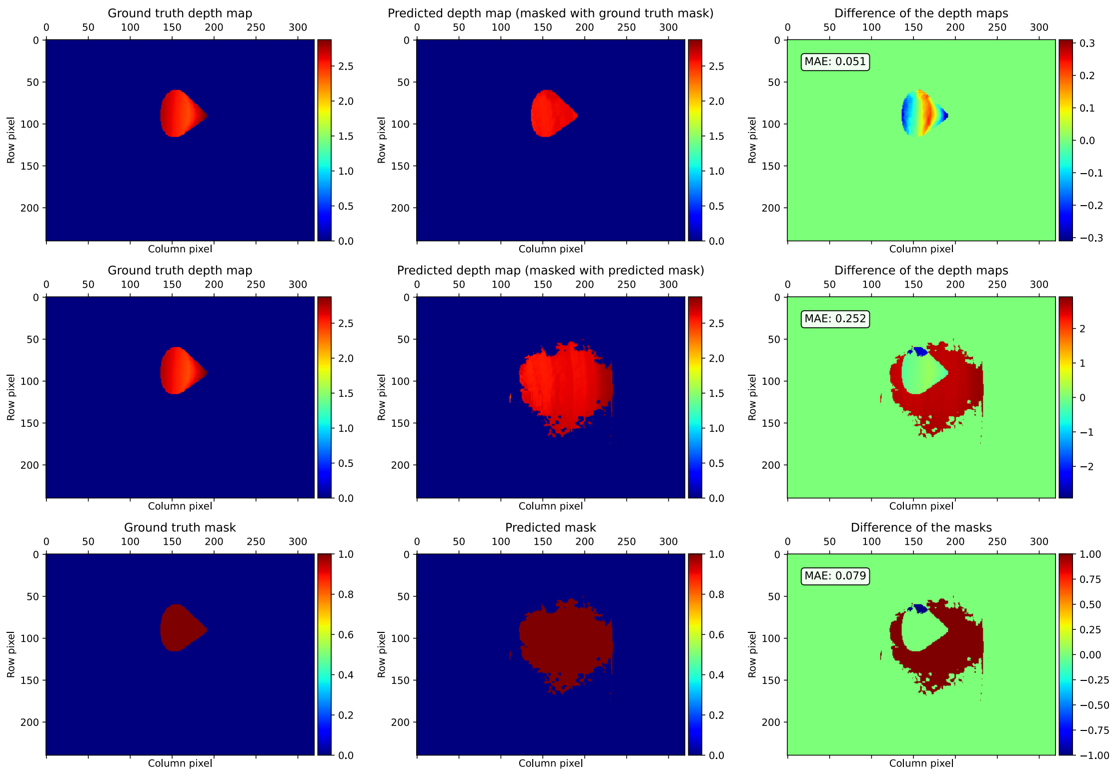


Figure A.7: sensor location: $(x: 1.0, y: -1.0, z: 1.65)$, sensor rotation: $(x: 90°, y: 0°, z: 50°)$ | | object shape: cone, object location: $(x: 1.1, y: 0.8, z: 1.95)$, object rotation: $(x: -89°, y: 22°, z: 0°)$ | | wall roughness: 0.55

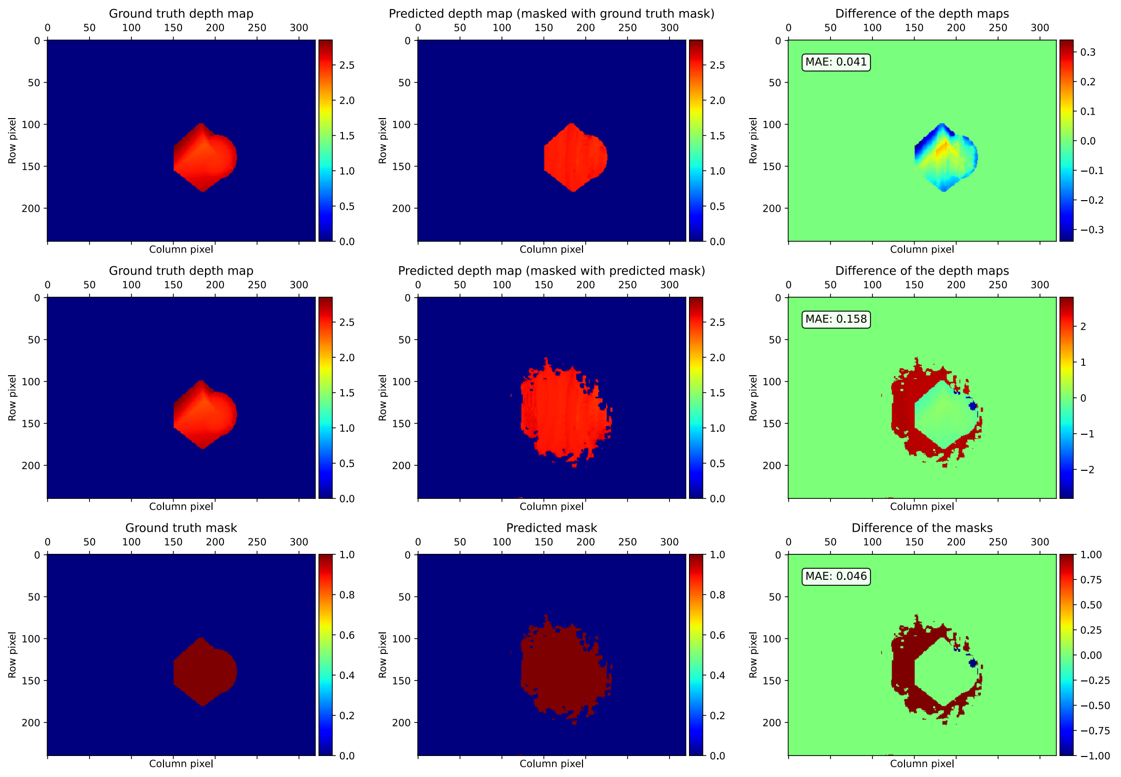


Figure A.8: sensor location: (x : 1.0, y : −1.0, z : 1.65), sensor rotation: (x : 90°, y : 0°, z : 50°) | | object 1 shape: cube, object 1 location: (x : 1.0, y : 1.0, z : 1.45), object 1 rotation: (x : 46°, y : −29°, z : 38°) | | object 2 shape: sphere, object 2 location: (x : 0.9, y : 1.1, z : 1.45), object 2 rotation: (x : 34°, y : −69°, z : 63°) | | wall roughness: 0.6

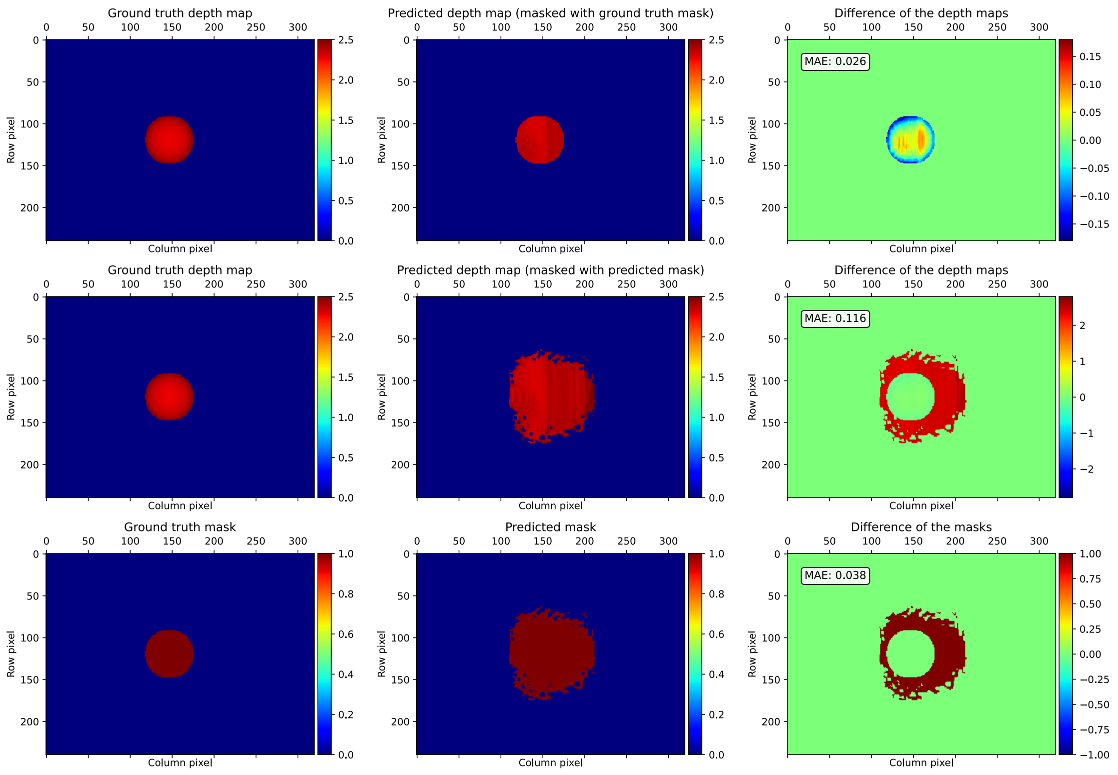


Figure A.9: sensor location: $(x: 1.0, y: -1.0, z: 1.65)$, sensor rotation: $(x: 90°, y: 0°, z: 50°)$ | | object shape: sphere, object location: $(x: 1.1, y: 0.6, z: 1.65)$, object rotation: $(x: 0°, y: 0°, z: 0°)$ | | wall roughness: 0.1

A.4 Test case D: fixed and variable sensor and variable wall

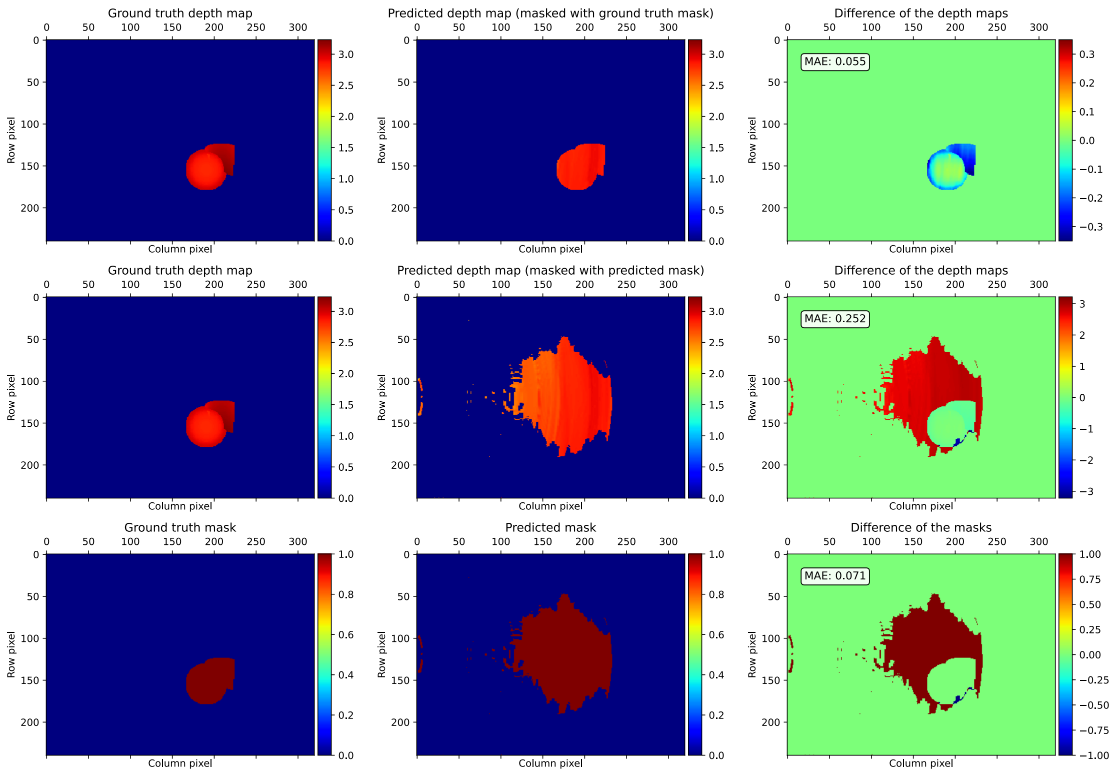


Figure A.10: sensor location: $(x: 1.0, y: -1.0, z: 1.65)$, sensor rotation: $(x: 90°, y: 0°, z: 50°)$ | | object 1 shape: sphere, object 1 location: $(x: 1.2, y: 1.3, z: 1.25)$, object 1 rotation: $(x: 0°, y: 0°, z: 0°)$ | | object 2 shape: concave plane, object 2 location: $(x: 1.1, y: 1.4, z: 1.35)$, object 2 rotation: $(x: 0°, y: 0°, z: 0°)$ | | wall roughness: 0.1

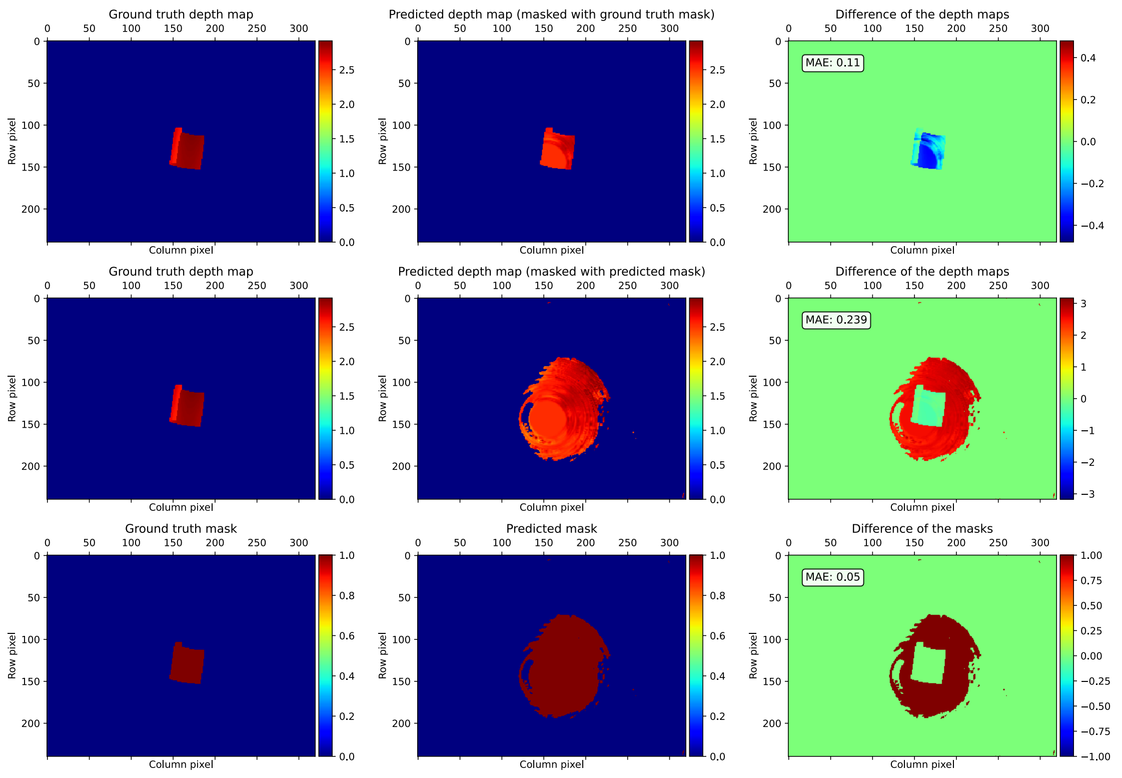


Figure A.11: sensor location: (x : 1.1, y : −1.0, z : 1.5), sensor rotation: (x : 95°, y : −3°, z : 89°) | | object shape: concave plane, object location: (x : 1.2, y : 0.9, z : 1.55), object rotation: (x : −41°, y : 111°, z : 47°) | | wall roughness: 0.65

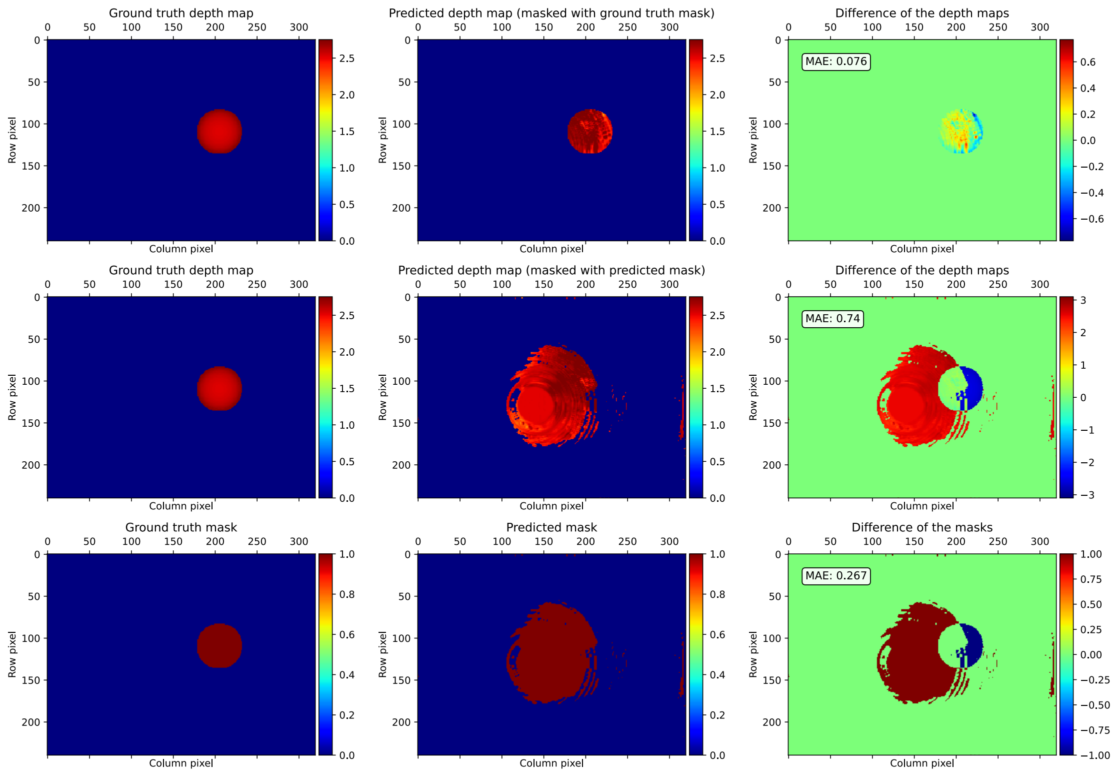


Figure A.12: sensor location: $(x: 1.3,\ y: -1.0,\ z: 1.5)$, sensor rotation: $(x: 92°,\ y: -1°,\ z: 86°)$ | | object shape: sphere, object location: $(x: 0.9,\ y: 1.2,\ z: 1.75)$, object rotation: $(x: 0°,\ y: 0°,\ z: 0°)$ | | wall roughness: 0.75

References

[1] Giovanna Sansoni, Marco Trebeschi, and Franco Docchio. "State-of-The-Art and Applications of 3D Imaging Sensors in Industry, Cultural Heritage, Medicine, and Criminal Investigation". In: *Sensors* 9.1 (2009), pp. 568–601. issn: 1424-8220. doi: `10.3390/s90100568`. url: `https://www.mdpi.com/1424-8220/9/1/568`.

[2] Daniele Faccio, Andreas Velten, and Gordon Wetzstein. "Non-line-of-sight imaging". In: *Nature Reviews Physics* 2.6 (2020), pp. 318–327.

[3] Shumian Xin, Sotiris Nousias, Kiriakos N Kutulakos, Aswin C Sankaranarayanan, Srinivasa G Narasimhan, and Ioannis Gkioulekas. "A theory of Fermat paths for non-line-of-sight shape reconstruction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6800–6809.

[4] Gautam Verma and Dolly Sharma. "Seeing Through the Walls with Wireless Technology: A Review". In: *International Journal of Sensors Wireless Communications and Control* 12.4 (2022), pp. 255–271.

[5] Adriano Simonetto, Gianluca Agresti, Pietro Zanuttigh, and Henrik Schäfer. "Lightweight Deep Learning Architecture for MPI Correction and Transient Reconstruction". In: *IEEE Transactions on Computational Imaging* 8 (2022), pp. 721–732. doi: `10.1109/TCI.2022.3197928`.

[6] Damien Lefloch, Rahul Nair, Frank Lenzen, Henrik Schäfer, Lee Streeter, Michael J Cree, Reinhard Koch, and Andreas Kolb. "Technical foundation and calibration methods for time-of-flight cameras". In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications*. Springer, 2013, pp. 3–24.

[7] Santiago Royo and Maria Ballesta-Garcia. "An overview of lidar imaging systems for autonomous vehicles". In: *Applied sciences* 9.19 (2019), p. 4093.

[8] Mohit Gupta, Shree K. Nayar, Matthias B. Hullin, and Jaime Martin. "Phasor Imaging: A Generalization of Correlation-Based Time-of-Flight Imaging". In: *ACM Trans. Graph.* 34.5 (Nov. 2015). ISSN: 0730-0301. DOI: 10.1145/2735702. URL: https://doi.org/10.1145/2735702.

[9] Enrico Buratto, Adriano Simonetto, Gianluca Agresti, Henrik Schäfer, and Pietro Zanuttigh. "Deep learning for transient image reconstruction from ToF data". In: *Sensors* 21.6 (2021), p. 1962.

[10] Felipe Gutierrez-Barragan, Huaijin Chen, Mohit Gupta, Andreas Velten, and Jinwei Gu. "itof2dtof: A robust and flexible representation for data-driven time-of-flight imaging". In: *IEEE Transactions on Computational Imaging* 7 (2021), pp. 1205–1214.

[11] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". In: *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (Dec. 2019). DOI: 10.1145/3355089.3356498.

[12] Jorge Garcia Pueyo. *Mitsuba 2 Renderer - transient*. Version b9332ae07a bd803dd3a60188d5d6f8f5f4ae6f81. 2021. URL: https://github.com/jgarciapueyo/mitsuba2-transient.

[13] Diego Royo, Jorge García, Adolfo Muñoz, and Adrian Jarabo. "Non-line-of-sight transient rendering". In: *Computers & Graphics* (2022). ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2022.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0097849322001200.

[14] Dorian Ros, Baptiste Nicolet, Delio Vicini, Michael Vasilkovsky, and Avatar Merlin Nimier-David. *Mitsuba Blender Add-on*. Version c6fbc8c3 dbf8be1ecf3e755b84e9a0cf16cedace. 2020. URL: https://github.com/mitsuba-renderer/mitsuba-blender.

[15] Florian Kainz, Rod Bogart, Piotr Stanczyk, and Peter Hillman. "Technical introduction to OpenEXR". In: *Industrial light and magic* 21 (2009).

[16] Jorge Garcia Pueyo. *mitsuba2-transient-scenes*. Version ab67c4caa9228 aa9d4115aeb90c3d3bab0aaf7b5. 2021. URL: https://github.com/jgarciapueyo/mitsuba2-transient-scenes.

[17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow, Large-scale machine learning on heterogeneous systems*. Nov. 2015. DOI: `10.5281/zenodo.4724125`.

[18] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.