UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Depth Image-Based Rendering for Multiview Plenoptic Camera

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Vicent Moltó Gallego

Tutor: Andrés Boza García

Tutor Externo: Mehrdad Teratani

Curso académico: 2022-2023

Abstract

Nowadays, the field of virtual reality is becoming increasingly more popular: many applications are surfacing, with better results each passing day. 3D information of the scene that will be rendered in virtual reality is mandatory for it to be realistic. That 3D information can be used to compute the depth of a scene and allow rendering virtual scenes that achieve realistic levels of depth, making the user perceive it as he/she would in the real world.

The 3D information can be captured by means of a plenoptic camera, which is a specialized camera that can capture the depth of a scene, along with the image corresponding to the scene itself. Common approaches to compute the depth of an image (known as depth map) require, in general, long computation time which makes it impossible to use the computed depth in real-time applications, making plenoptic cameras much more suitable for the job.

Rendering virtual views can be achieved with a technique called Depth Image Based Rendering (DIBR). It uses real images captured by some camera and their respective depths to synthesize virtual views located in between the reference images. This technique, combined with plenoptic cameras, would enable view synthesis in real-time.

This master thesis evaluates the performance of plenoptic 2.0 cameras for DIBR. It will also present a reproducible methodology that can be used for any kind of depth-sensing device. To evaluate the performance of the plenoptic camera, a dataset of images will be captured using a RayTrix plenoptic 2.0 camera. Then, depth estimation using tools will be performed. Those tools are the MPEG-I reference software Depth Estimation Reference Software (*DERS*) and the open source 3D reconstruction software *Colmap*. DIBR will be performed using the depth maps generated by these two offline approaches, as well as with the depth map generated in real-time by the plenoptic camera. The synthesized views will be used as a measure for quality assessment of the depth maps generated by each one of the three approaches. There will be two view synthesis experiments: one using only one view as reference and the other using multiple views as reference. Finally, a comparison with another depth-sensing device, he Azure Kinect, will be done.

Results show that the best depth maps are yielded by *DERS*, followed by RayTrix. *Colmap* falls behind because its depth maps are very limited since they are incomplete, but having great potential. Lastly, performance of RayTrix camera is better than the one of Azure Kinect when capturing close detail in the scene, whereas the Kinect can capture a wider area.

Keywords: plenoptic camera, Depth Image Based Rendering, view synthesis, Virtual Reality, depth estimation, depth-sensor, assessment, total focus image, RayTrix, Calibration, DERS, Colmap, RVS, RLC, RPVC.

Resumen

Hoy en día, el campo de la realidad virtual se está volviendo cada vez más popular: están surgiendo muchas aplicaciones, con mejores resultados cada día que pasa. Información 3D de la escena que se renderizará en realidad virtual es necesaria para que ésta sea realista. Esa información 3D se puede usar para calcular la profundidad de una escena y permitir renderizar escenas virtuales que alcanzan niveles realistas de profundidad, haciendo que el usuario las perciba como lo haría en el mundo real.

La información 3D se puede capturar por medio de una cámara plenoptica, que es una cámara especializada que puede capturar la profundidad de una escena, junto con la imagen correspondiente a la misma. Los enfoques comunes para calcular la profundidad de una imagen (conocidos como mapas de profundidad) requieren, en general, un tiempo de computación prolongado, lo que hace imposible utilizar la profundidad calculada en aplicaciones en tiempo real, lo que hace que las cámaras plenopticas sean mucho más adecuadas para el trabajo.

El renderizado de vistas virtuales se puede lograr con una técnica llamada Depth Image Based Rendering (DIBR). Utiliza imágenes reales captadas por alguna cámara y sus respectivas profundidades para sintetizar vistas virtuales situadas entre las imágenes de referencia. Esta técnica, combinada con cámaras plenopticas, permitiría síntesis de vistas en tiempo real.

Este trabajo de final de máster evalúa el rendimiento de las cámaras plenoptic 2.0 para DIBR. También presenta una metodología reproducible que se puede utilizar para cualquier tipo de dispositivo de detección de profundidad. Para evaluar el rendimiento de la cámara plenoptica, se capturará un conjunto de datos de imágenes utilizando una cámara plenoptica 2.0 RayTrix. Luego, se realizará la estimación de la profundidad utilizando herramientas. Esas herramientas son el software de referencia MPEG-I Depth Estimation Reference Software (*DERS*) y el software de reconstrucción 3D de código abierta *Colmap*. DIBR se realizará utilizando los mapas de profundidad generados por estos dos enfoques offline, así como con el mapa de profundidad generado en tiempo real por la cámara plenoptica. Las vistas sintetizadas se utilizarán como medida para evaluar la calidad de los mapas de profundidad generados por cada uno de los tres enfoques. Habrá dos experimentos de síntesis de vistas: uno usando solo una vista como referencia y el otro usando múltiples vistas como referencia. Finalmente, se realizará una comparación con otro dispositivo capaz de capturar la profundidad, el Azure Kinect.

Los resultados muestran que los mejores mapas de profundidad son producidos por *DERS*, seguidos por los de RayTrix. *Colmap* se queda atrás porque sus mapas de profundidad son muy limitados ya que están incompletos, pero tienen un gran potencial. Por último, el rendimiento de la cámara RayTrix es mejor que el de Azure Kinect al capturar detalle en la escena, mientras que Kinect puede capturar un área más amplia.

Palabras clave: cámara plenoptica, Depth Image Base Rendering, síntesis de vistas, realidad virtual, estimación de profundidad, sensor de profundidad, evaluación, imagen de enfoque total, RayTrix, calibración, DERS, Colmap, RVS, RLC, RPVC.

Resum

Avui dia, el camp de la realitat virtual s'està tornant cada cop més popular: sorgeixen moltes aplicacions, amb millors resultats cada dia que passa. Informació 3D de l'escena que es renderitzarà en realitat virtual és necessària perquè sigui realista. Aquesta informació 3D es pot fer servir per calcular la profunditat d'una escena i permetre renderitzar escenes virtuals que arriben a nivells realistes de profunditat, fent que l'usuari les percebi com ho faria al món real.

La informació 3D es pot capturar per mitjà d'una càmera plenòptica, que és una càmera especialitzada que pot capturar la profunditat d'una escena, juntament amb la imatge corresponent. Els enfocaments comuns per calcular la profunditat d'una imatge (coneguts com a mapes de profunditat) requereixen, en general, un temps de computació prolongat, cosa que fa impossible utilitzar la profunditat calculada en aplicacions en temps real, cosa que fa que les càmeres plenòptiques siguin molt més adequades per a la feina.

El renderitzat de vistes virtuals es pot aconseguir amb una tècnica anomenada Depth Image Based Rendering (DIBR). Utilitza imatges reals captades per alguna càmera i les seves profunditats respectives per sintetitzar vistes virtuals situades entre les imatges de referència. Aquesta tècnica, combinada amb càmeres plenòptiques, permetria síntesi de vistes en temps real.

Aquest treball de final de màster avalua el rendiment de les càmeres plenoptic 2.0 per a DIBR. També presenta una metodologia reproduïble que es pot fer servir per a qualsevol tipus de dispositiu de detecció de profunditat. Per avaluar el rendiment de la càmera plenòptica, es capturarà un conjunt de dades d'imatges utilitzant una càmera plenòptica 2.0 RayTrix. Després, es farà l'estimació de la profunditat utilitzant eines. Aquestes eines són el programari de referència MPEG-I Depth Estimation Reference Software (*DERS*) i el programari de reconstrucció 3D de codi obert *Colmap*. DIBR es farà utilitzant els mapes de profunditat generats per aquests dos enfocaments offline, així com amb el mapa de profunditat generat en temps real per la càmera plenoptica. Les vistes sintetitzades s'utilitzaran com a mesura per avaluar la qualitat dels mapes de profunditat generats per cadascun dels tres enfocaments. Hi haurà dos experiments de síntesi de vistes: un usant només una vista com a referència i l'altre usant múltiples vistes com a referència. Finalment, es farà una comparació amb un altre dispositiu capaç de capturar la profunditat, l'Azure Kinect.

Els resultats mostren que els millors mapes de profunditat són produïts per *DERS*, seguits pels de RayTrix. *Colmap* es queda enrere perquè els seus mapes de profunditat són molt limitats ja que estan incomplets, però tenen un gran potencial. Finalment, el rendiment de la càmera RayTrix és millor que el d'Azure Kinect en capturar detall a l'escena, mentre que Kinect pot capturar una àrea més àmplia.

Paraules clau: càmera plenòptica, Depth Image Base Rendering, síntesi de vistes, realitat virtual, estimació de profunditat, sensor de profunditat, avaluació, imatge d'enfocament total, RayTrix, calibració, DERS, Colmap, RVS, RLC, RPVC.

Résumé

De nos jours, le domaine de la réalité virtuelle devient de plus en plus populaire : de nombreuses applications voient le jour, avec de meilleurs résultats de jour en jour. Pour reconstituter la scène virtuellement, les informations 3D de celle-ci sont nécessaires. Ces informations 3D peuvent être utilisées pour calculer la profondeur d'une scène et permettre le rendu de scènes virtuelles atteignant des niveaux de profondeur réalistes, permettant à l'utilisateur de les percevoir comme il le ferait dans le monde réel.

Les informations 3D peuvent être capturées au moyen d'une caméra plénoptique, qui est une caméra spécifique capable de capturer la profondeur d'une scène, ainsi que l'image qui lui correspond. Les approches courantes pour calculer la profondeur d'une image (appelée cartes de profondeur) nécessitent généralement un long temps de calcul, ce qui rend impossible l'utilisation de la carte de profondeur calculée dans les applications en temps réel, incitant les caméras plénoptiques à être beaucoup mieux adaptées à ce genre d'applications.

Le rendu de la vue virtuelle peut être réalisé avec une technique appelée Depth Image Based Rendering (DIBR). Cette technique utilise des images réelles capturées par une caméra et leurs profondeurs respectives pour synthétiser des vues virtuelles situées entre les images de référence. Celle-ci, associée à des caméras plénoptiques, permettrait une synthèse des vues en temps réel.

Ce mémoire évalue les performances des caméras plénoptiques 2.0 pour la synthèse de vues en utilisant DIBR. Il présente également une méthodologie reproductible qui peut être utilisée pour tout type de dispositif de d'acquisition de profondeur. Pour évaluer les performances de la caméra plénoptique, un ensemble de données d'image sera capturé à l'aide d'une caméra plénoptique RayTrix 2.0. Ensuite, l'estimation de la profondeur sera faite à l'aide d'outils. Ces outils sont le logiciel de référence MPEG-I Depth Estimation Reference Software (*DERS*) et le logiciel open source de reconstruction 3D *Colmap*. La synthèse de vue par DIBR sera réalisé en utilisant les cartes de profondeur générées par ces deux approches, ainsi que la carte de profondeur générée, celle-ci en temps réel, par la caméra plénoptique. Les vues synthétisées seront utilisées pour évaluer la qualité des cartes de profondeur générées par chacune des trois approches. Il y aura deux expériences de synthèse de vues : l'une utilisant une seule vue de référence et l'autre utilisant plusieurs vues de référence. Enfin, une comparaison sera faite avec une autre caméra, l'Azure Kinect, elle-aussi capable d'acquérir la profondeur d'une scène.

Les résultats démontrent que les meilleures cartes de profondeur sont produites par *DERS*, suivi de RayTrix. *Colmap* étant la dernière car ses cartes de profondeur sont incomplètes, mais elles ont un grand potentiel. Enfin, la caméra RayTrix est plus performante que l'Azure Kinect pour capturer les détails de la scène, tandis que la Kinect peut capturer une zone plus large.

Mots-clés: caméra plénoptique, Depth Image Base Rendering, synthèse de vues, réalité virtuelle, estimation de profondeur, capteur de profondeur, évaluation, image à plusieurs focus, RayTrix, Calibration, DERS, Colmap, RVS, RLC, RPVC.

Acknowledgements

I would like to thank, first of all, my promoter, professor Mehrdad Teratani for guiding and supporting me during the elaboration of this thesis and my MA1 project, helping me out in any way possible. I would also like to thank professor Gauthier Lafruit for his valuable advice during our discussions. Secondly, I thank my supervisors, Hamed and Armand, that gave me hand whenever I asked this half a year. Finally, I appreciate the support given to me by my friends and, specially, my family, for making the effort of having me for a year a half far away from home, here in Brussels.
This work was supported in part by the HoviTron project (N° 951989), in part by the FER 2021 project (N° 1060H000066-FAISAN), and in part by the Emile DEFAY 2021 project (N° 4R00H000236).

Contents

List of Figures

List of Tables

1 Introduction and content

1.1 Introduction

Nowadays, the field of Virtual Reality (VR) is becoming more and more popular: controlling robots or drones, automated cars, 3D videoconferences or gaming are just examples of it. Contrary to what general masses think, Virtual Reality applications are not only the ones that use a head-mounted display, there are many more. Free point television [1,2], for example, is an emerging technology that allows to user to change the point of view of which the TV program is being seen: move the aerial view of a football match however the user wants, even making it not follow the ball, could be an example of it.

In order to perform VR correctly, the user must feel like he/she is ”transported” to the virtual world. Depth perception is key to achieve that. In the case of using a head mounted display, the so-called 6 Degrees of Freedom (6DoF) must be achieved, which allow not only to see your surroundings depending on your position, but also to get closer or further from the objects that one sees. It allows natural movement, as if one was really there, in another world. In order to achieve that, 3D information of the scene must be taken into account.

That 3D information can be acquired mainly in two different ways. The first would be estimating it after the capture of multiple images. With those images one could generate 3D reconstruction, a point cloud or perform stereo matching. The second option would capturing it directly along with an image using a specialized device, a plenoptic camera. A plenoptic camera (e.g. RayTrix [3]) captures in one shot hundreds of tiny images thanks to a Lenslet objective (an objective composed of hundreds of micro lenses). The advantage of the produced Lenslet images is that they contain the 3D information of the scene. It presents as a main advantage that it is able to capture 3D information directly and immediately, whereas estimating it from a set of images requires time and much computation.

With the 3D information, one may extract depth from it. This depth information greatly increases realism in VR applications. But not only depth perception must be of quality to achieve realism, transition between images must also be. Smooth transitions can be performed if one captures a lot of images from different positions, following a path, but too many images would be required, making it a titanic task. That can be solved with view synthesis: generating by computer synthetic images that are in between the images that have been captured with a camera, reducing the number of pictures to be physically captured by a lot.

This view synthesis can make use of the depth extracted from the 3D information of the scene by means of a technique called Depth Image Based Rendering (DIBR). This technique makes use of pictures captured by a camera and the depth of the scene (known as depth map) to synthesize new views in between. Using a that set of images to estimate depth can produce good results, but that estimation is slow. This makes it extremely difficult, if not impossible, to perform DIBR with regular cameras in real-time applications. This leaves room to use devices specialized in depth acquisition, such as plenoptic cameras.

This thesis has as a main objective assessment of plenoptic cameras for DIBR real-time applications. As a secondary objective, a reproducible pipeline, which can be used in other experiments, is proposed. To achieve those objectives, DIBR is performed using the depth captured by means of a RayTrix plenoptic camera, and then be compared with two depth estimation approaches: the stan-

dard software Depth Estimation Reference Software (*DERS*) and the open source software *Colmap*.

A dataset of images using a plenoptic camera is captured. This dataset contains images shot from slightly different positions: 9 of them are in a 3x3 matrix, and are separated 3cm from each other. 58 more images are positioned in the middle row of the matrix, with a distance of 1mm with respect to each other. Then, depth estimation is performed using the offline approaches *DERS* and *Colmap*. Finally, view synthesis takes place, using a tool known as Reference View Synthesized (*RVS*). The synthesized views are assessed for quality, and are used as a metric of the performance of the depth information generated by each approach. This view synthesis is performed using only one view as reference: using the central view as reference, along with its depth map, the rest are synthesized. In addition, a little experiment in which view synthesis using multiple reference images is performed. In this case, the central view of the matrix is generated using the corners as reference. This approach is closer to real-world applications. Finally, the performance of the RayTrix camera is compared to that of another depth sensing device: Azure Kinect.

1.2 Content of the Thesis

This thesis is divided into three main sections. The first one will be an state-of-the-art explaining key concepts to understand the whole thesis. It will include a discussion to illustrate the difference between regular and plenoptic cameras and key concepts on depth computation and DIBR. The distinct tools and software that will be used in the experiment will also be introduced. The second chapter will be the main topic of the thesis: the experimentation. It will have four main steps: acquisition, depth estimation, view synthesis and quality assessment. Finally, a comparison of performance with the Azure Kinect will be performed.

2 State of the art

In this section, we define the concepts necessary to completely understand the experiment performed during the elaboration of this thesis. We will first start with an introduction to the light field, which will be necessary for the following part, acquisition. In the acquisition section, different types of that will be used in the experiment will be explained. Next, depth computation, mentioning several different approaches to perform it. Finally, some words on Depth Image Base Rendering, explaining the concept. Several tools that will be used during the experiments will also be introduced throughout the different sections.

2.1 Light Field

A light field is a vector function that describes the amount of light that traverses all points in space in all possible different directions, thus giving three-dimensional information. In imaging, it can be represented as *n*x*n* matrix of images [4]. These images, also called views, capture the same scene, but from slightly different perspectives. These small differences allow to capture that light reflected and refracted from the scene in different directions, thus generating a light field. Other option to create a light field is 3D reconstruction. This approach also requires to capture (or synthesize) a certain amount of images from different perspectives.

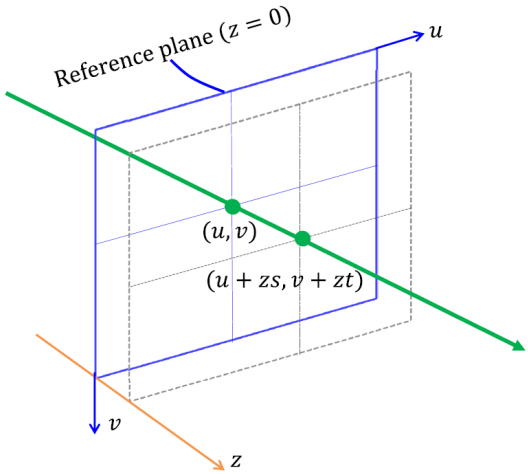


Figure 1: 3D information extracted from a light field

2.2 Acquisition

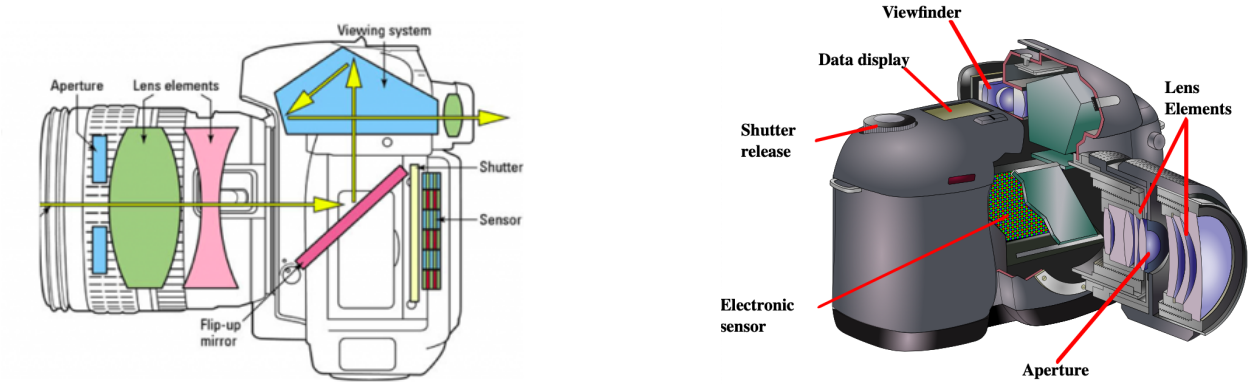
Acquisition might be one of the most crucial steps when rendering real scenes in virtual reality. In order to show the complexity and advantages of a plenoptic camera with respect to what normally is considered as a standard or regular camera, this section will explain how both of them work, showing the crucial differences among them. Some brief words will also given about Azure Kinect [5].

2.2.1 Regular Cameras

When thinking of ”regular” cameras, many types can come to mind: digital cameras, instant cameras or even film cameras. All of them share the same mechanism to capture images, with slight differences, specially between digital-type cameras and non-digital ones. Digital cameras can show the image in a screen immediately after the picture is taken, whereas the non-digital ones can't. But, in general, they share the same mechanism. Note that we are talking about cameras that capture light from the visible spectrum, we are not taking into account specialized cameras like infrared or ultra-violet.

Any type of ”regular” camera can be used to capture scenes that can be rendered for virtual reality applications, but it is recommended to use a high-quality camera. Such cameras can be single-lens reflex (SLR) or digital single-lens reflex (DSLR) camera, being the latter one of the most commonly used in professional photography nowadays.

In this document, we will be focusing mostly on the design of SLR cameras, but the mechanism through which the other types of cameras take images is the same, the main difference will be how the user sees what it is going to be captured by the camera.



(a) SLR camera mechanism [1]

(b) Basic elements of a reflex camera [2]

Figure 2: Regular cameras

Figures 2a and 2b show the basic components and mechanism of a SLR camera. Light enters through the aperture and goes through the lenses. Then it is reflected on the mirror and enters the viewing system, that allows the user the see what is going to be captured. When the shutter is released, the mirror will move, allowing the light to be capture by the sensor, thus creating the picture.

In order to control the amount of light that enters inside the camera, the aperture and the shutter are used. The aperture is the main opening, and can be adjusted to be bigger or smaller by overlapping plates called the aperture ring. It is typically installed with the lens, and to adjust it one normally need to rotate it. Adjusting the aperture allows the camera to focus on shorter or longer distances: with a narrow aperture, the depth of field increases. This means that objects that are far from the camera will be in focus. On the contrary, when the aperture is wide, the objects that will be focused will be the ones that are close to the camera.

Regarding the shutter, it is used to control the amount of time the sensor is exposed to the light that is entering the camera through the aperture and the lenses. It works in the most simple of manners: it opens, the sensor is exposed to light, and then closes. The duration for which the shutter is

[1] Image taken from Howthingswork.org.
[2] Image taken from Wikipedia.

open or released is called shutter speed or exposure time. This exposure time can be used to blur the image, and, when done properly, give the impression of movement in still photograph.

When too little light is let into the camera, the image will be darker or under-exposed. Otherwise, it will be pale, or over-exposed. Adjusting both the shutter speed and the aperture is crucial when taking quality pictures. It is worth mentioning that a longer shutter time can be compensated with a smaller aperture, and vice versa.

With respect to the lenses, many types of lenses exist, but the two main types are prime lenses, which have a fixed focal length, and zoom lenses, whose focal length is variable. The shape of the lens is also important: convex lenses (converging lenses) will focus the light on one point, while concave ones (diverging lenses) will disperse it.

Note that most of modern cameras include a microprocessor, which will be able to calibrate the camera automatically, up to a certain point, to help the user capture quality images.

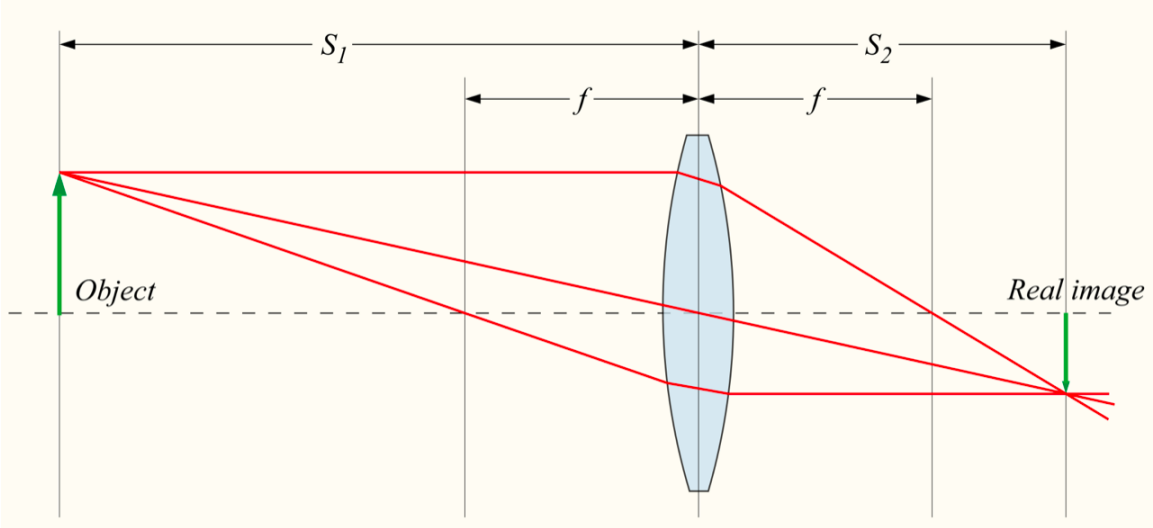


Figure 3: Basic camera lens [3]

In Figure 3, we can see the basic mechanism of a simple converging lens, which is able to create an image of the object whose light is reflected on the lens. In the image, *S1* is the distance between the object and the lens, *S2* the distance between the lens and the image and f the focal distance. Inside a camera, the sensor would be placed at distance *S2* from the lens, exactly where the image would be formed, in order to capture it.

Different configurations of lenses (thickness, shape, etc.) or even multiple lenses at a time can be used to achieve several results.

2.2.1.1 Camera Parameters

Camera parameters [6] describe the camera properties (intrinsic parameters) and the location and orientation of the camera (extrinsic parameters). Intrinsic camera parameters depend on the characteristics of the camera, such as resolution, principle point, focal length and skewness. On the other hand, extrinsic camera parameters refer to the physical position of the camera in space: its

[3] Image taken from Wikipedia.

rotation and translation, always with respect to some reference. Both type of parameters are usually represented in the form of a matrix:

$$I = \begin{pmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \qquad E = \begin{pmatrix} R_{3x3} & T_{3x1} \\ 0_{1x3} & 1 \end{pmatrix} \tag{1}$$

I represents the 3x3 intrinsic parameter matrix. α represents the focal length in terms of pixels, γ the skew coefficient between the *x* and *y* axis, and u_0 and v_0 refer to the pixel coordinates of the principle point (center of the captured image by rule of thumb).

E refers to the extrinsic parameters matrix, which is 4x4. In it, R_{3x3} refers to the rotation 3x3 matrix and T_{3x1} to the translation column vector of three components.

Camera parameters are used to perform transformations on images, as well as in image synthesis.

2.2.1.2 Camera Parameter Calibration using Colmap

Colmap [7] is an open-source general-purpose Structure-from-Motion (SfM) [8] and Multi-View Stereo (MVS) [9] pipeline with a graphical and command-line interface. It offers a wide range of features for reconstruction of ordered and unordered image collections.

It can perform 3D reconstruction and extract the camera parameters from it. It needs as input the set of images one needs their camera parameters and/or their depth maps.

To obtain the camera parameters (both intrinsic and extrinsic), it takes several points, known as features, of all the inputted images, normally thousands of points, and tries to identify where they lie in 3D space, generating something similar to a point cloud. Since it knows what points correspond to what image and how they moved to match the same 3D position, it can infer the camera extrinsic parameters (position and rotation) of each camera. Note that the program assumes each image corresponds a different camera. For the intrinsic parameters, it will use different methods which depend on the camera type the user has chosen, which normally are a simplified versions of the camera types (and intrinsic parameters by extension) offered by *OpenCV* [10].

Refer to Appendix B.5 for further information on installation and operation.

2.2.2 Plenoptic Cameras

Plenoptic cameras, also known as light field cameras [11,13], are a special kind of cameras which are able to capture information about the light field from the scene they are capturing. That means they are able to capture the intensity of the light, but from different angles thanks to their intrinsic architecture. This is the main difference between plenoptic and regular cameras, since regular cameras are only capable of capturing the intensity of the light of the scene in just just angle.

One of the main advantages of plenoptic cameras is that one can extract the depth of the image from it. The other great advantage is the possibility to capture several ”would-have-been” views. That is, capturing the same image multiple times, but with small displacement between each of the captured views.

Plenoptic cameras can be divided into two main types: standard plenoptic cameras, also known as plenoptic 1.0 cameras, and (multi-) focused plenoptic cameras, or plenoptic 2.0 cameras.

2.2.2.1 Standard Plenoptic Cameras

Plenoptic cameras introduce a mechanism to capture the light field of a scene: the lenticular array [11], or, in other words, an array of microlenses (Figure 4). This array of microlenses must be focused on the principal plane of the main lens, and must be placed right next to the sensor. One can think of this array of microlenses as an array of small cameras, all aimed at the main lens. This mechanism could be compared to taking a lot of small pictures of areas of a bigger picture at once, and merging them together in the same resulting image.

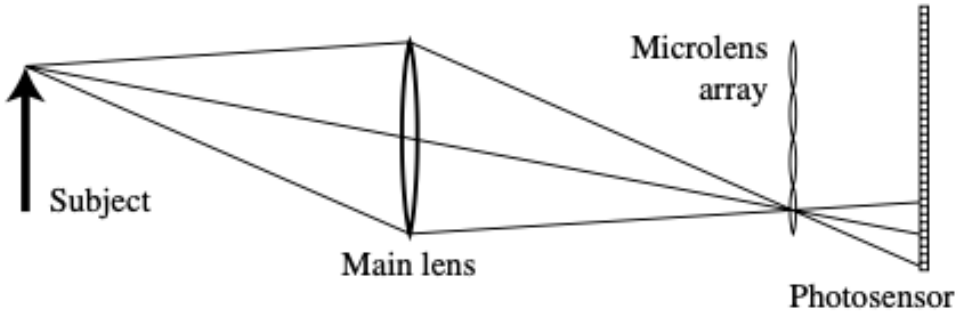


Figure 4: Optical system of a standard plenoptic camera
[12]

This configuration presents one main problem. Since the size of the sensor is the same as in a regular camera but we are using the cells to capture information of the same area several times, the resolution in general will be much smaller than in a regular image. That is, we are dividing one macropixel, which would correspond to a regular pixel in a normal camera, into n subpixels, which correspond to the actual cells of the sensor. In a normal camera, the mapping was *1-to-1*, but now the mapping is *1-to-n*, thus decreasing the resolution if the sensor size is kept the same. That is the same as saying that we are obtaining n views from one macropixel.

In order to capture quality images [12], the *f-number*, which is the aperture diameter divided by the focal length, of both the main lens and the microlens array must be adequate. If the main lens' *f-number* is bigger (i.e. its aperture is smaller relative to its focal length), then the pixels are cropped and the resolution is wasted. Otherwise, they overlap too much, thus contaminating each other's signals.

Once the image is processed, it will give what is called a lenslet image. Since it was captured using a standard plenoptic camera, it would be a standard lenslet image or lenslet 1.0 image. From it, one can extract up to n views from original scene from which the image was taken. That can be achieved by selecting the appropriate pixels to synthesized the different views.

One of the main advantages is the possibility to refocus digitally the already-captured image. This can be achieved just shifting and adding the different sub-aperture images, which correspond to the different views.

2.2.2.2 Focused Plenoptic Cameras

Further development in the field led to (multi-) focused plenoptic cameras:

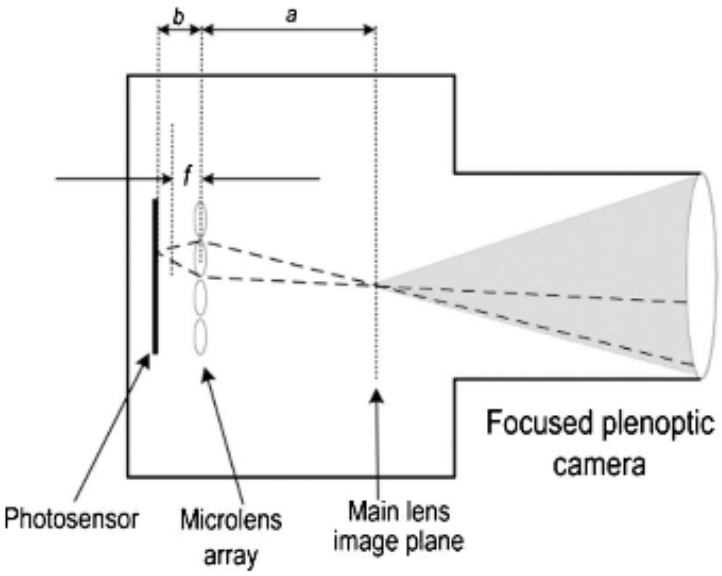


Figure 5: Focused plenoptic camera
[13]

Its main difference with the standard plenoptic camera is that the microlens array is placed either before or behind the focal plane of the main lens, that is, at a certain distance from the sensor, instead of right next to it [13]. In Figure 5, the position of the microlenses satisfies the equation $1/a + 1/b = 1/f$, where a, b, and f are, respectively, the distance from the microlens to the main lens image plane, the distance from the microlens to the sensor, and the focal length of the microlens. This allows to use bigger microlenses than the ones used in the standard plenoptic cameras, which leads to integrate data across the different microlens images instead of integrating the data from within the microlens images, giving a new format of image as a result: focused lenslet image or lenslet 2.0 image.

This new way of generating the data allows to highly increase the resolution. The main reason is that we are using several pixels per microlens, since they are bigger. Taking into account that plenoptic cameras can capture multiple angular directions, when rendering with just one angular direction, we would obtain a total of M samples. This would give a final resolution of M times the original resolution of the standard plenoptic camera.

Another advantage of the focused plenoptic camera with respect to the standard plenoptic camera is the ability to focus certain parts of the captured images during the rendering process: when transforming the lenslet image to obtain regular images that are agreeable to the human eye. This is allowed by the dedicated rendering algorithm used by this type of cameras. The main idea is to select a pitch from each one of the microlens images. That pitch is defined by the number of pixels of the macroimage. Then, a square of pixels is selected from each one of the microimages and put together to form the final image. Figure 6 shows it schematically:

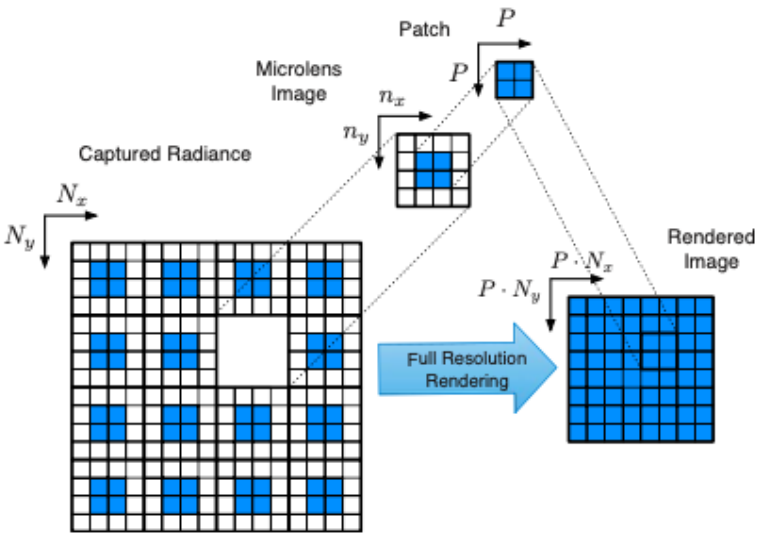


Figure 6: Focused plenoptic camera rendering algorithm
[13]

A different pitch size would correspond to a different depth. Choosing different pitch sizes, one could render the same image but with different focus.

Using the same pitch size when rendering images will create artifacts in areas which are not focused. In order to solve this issue, one can use different pitch sizes for different areas. That way, there will be a main focused area and the non-focused area without artifacts. This can be achieved by estimating the depth of each microlens image and choosing the pitch accordingly to its depth and the depth of its own neighbours. However, it has a drawback: the resulting image will have ”all in-focus”.

To solve that problem, a blending method was introduced, which consists in averaging the same spatial point across multiple microlens images. Combining the depth estimation and the blending methods, one can focus the rendered image at a certain distance with a minimal amount of artifacts.

It is also worth mentioning that, recently, multi-focused plenoptic cameras have been developed. These cameras use the same basic design as a focused plenoptic camera, but having microlenses in its lenticular array with three different focal lengths, instead of having them all the same focus. With those three types, one can get three different images of just one spot with different focuses. Higher spacial and angular resolution can be achieved, leading to the capture of even more information with just one shot.

2.2.2.3 Reference Lenslet content Convertor

RLC stands for Reference Lenslet content Convertor [14,15,16,17]. This software belongs to the standard MPEG-I (Moving Picture Expert Group - Immersive).

This tool converts a lenslet image obtained from a multi-focused plenoptic camera, i.e. RayTrix camera, into an array of 5x5 multi-view images. This array can be considered as an array of normal images that capture the same scene but from a slightly different point of view, as explained in the previous section.

It works as follows: first it needs to estimate a patch size, as explained in the section of focused plenoptic cameras. In this case, since it is a multi-focused plenoptic camera, it need to take into consideration the different focal lengths of the microlenses. This patch is computed using the Laplacian of all the viewpoints (images).

Then, integration of different types of microlenses is in order. To do so, a weighting averaging method is used. This method will take into account the type of the microlens, as well as the patch size used. That will lead to a great-quality multi-view image array, with high resolution and little artifacts. Figure 7 shows this process schematically.

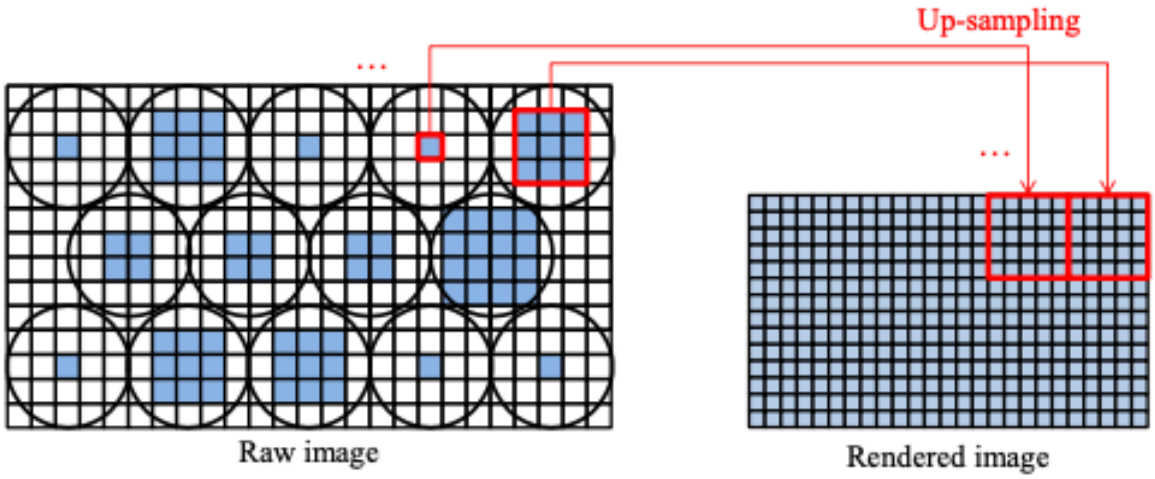


Figure 7: Choosing and integrating the patches in RLC
[15]

It takes as input the lenslet image one wants to convert, the configuration file for the camera with which the lenslet image has been taken and a parameter file. In this parameter file, the paths to the lenslet image and the configuration file are specified, as well as the output path for the multi-view images and the options the user wants to use.

Refer to Appendix B.2 for further information on installation and operation.

2.2.2.4 Reference Plenoptic Virtual camera Calibrator

The Reference Plenoptic Virtual camera Calibrator [18,19,20], or *RPVC*, is a pipeline that belongs to the standard MPEG-I. It includes a set of scripts that are used to calibrate subaperture views of plenoptic 2.0 camera arrays. That is, to calculate the camera parameters of the plenoptic cameras and transform them into camera parameters which can be used by normal cameras to simulate the plenoptic camera by taking several shots. Figure 8 shows the basic pipeline to be followed, which will be explained further in the following lines:

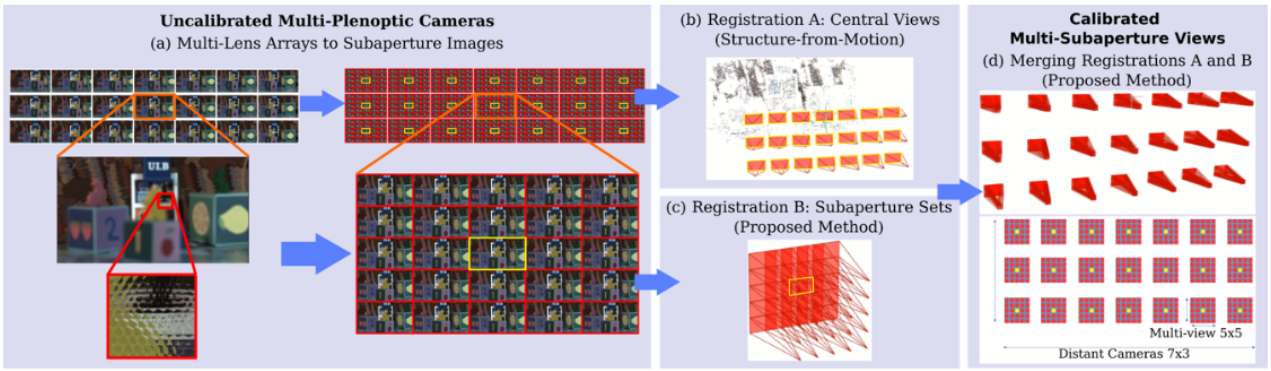


Figure 8: RPVC pipeline
[18]

First of all, one needs to use *RLC* to obtain the 5x5 array of multi-view images from a lenslet image. Then, using the software *Colmap* to obtain the intrinsic and extrinsic camera parameters. The central view (image 13) will then be registered using the camera parameters from the plenoptic camera.

Next, the 24 remaining images need to be registered. Their own individual camera parameters, both extrinsic and intrinsic, have to be calculated using the parameters of the plenoptic camera. This is achieved using *Colmap* to compute their own camera models and the scripts provided with *RPVC* to adjust the format. Then, with *Colmap* their depth maps are computed.

Finally, using the depth maps, camera models and the central views, the final camera parameters for each one of the reaming 24 subaperture views is computed and registered.

They are simply a collection of *Python* scripts, so they only need *Python* [21] to be installed in order to execute, along with *Colmap* and *RLC*. They can be downloaded from this repository.

2.2.3 Azure Kinect

Microsoft Azure Kinect is an active depth-sensing device that uses time of flight in order [22] to sense depth. Time of flight will be briefly explained in Section 2.3.3.

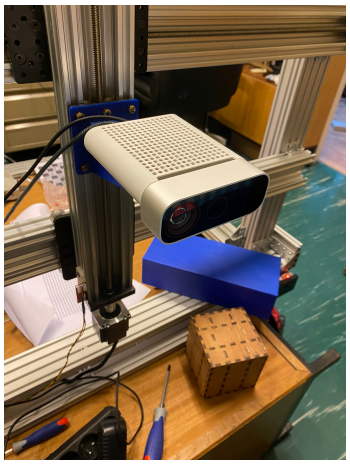


Figure 9: Azure Kinect

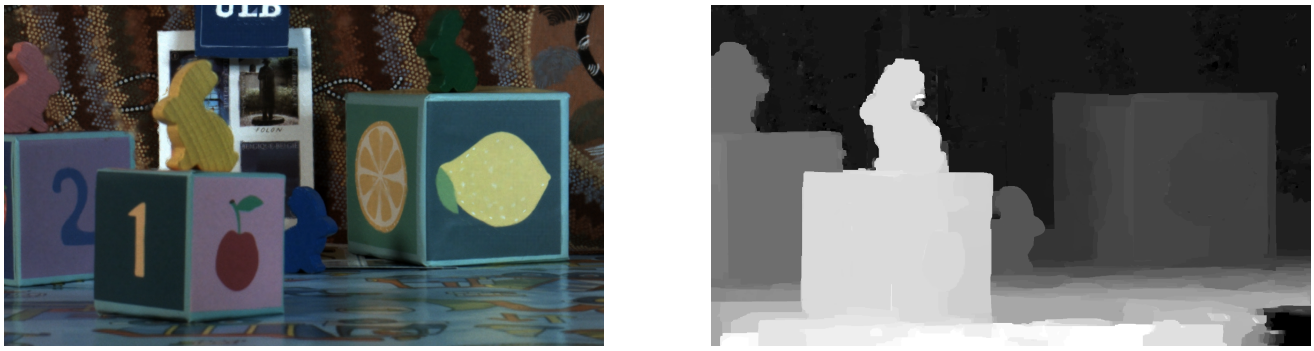
One of the particularities of the Kinect is that it uses two cameras to perform RGBD acquisition: one for colour (thus capturing RGB) and another for the depth. Since those are two different sensors, they are placed in different physical location inside the device, thus possibly generating disocclusion artifacts.

2.3 Depth Computation

Depth computation, as it name suggests, consists in estimating the depth of each one of the elements found in an image. In this section, several approaches will be discussed, being some of them inherent to the camera that is being used to capture the scene whose depth will be computed.

Regardless of the method used, the final product of depth computation is a depth map. It is a 1-channel image in which the pixels take the value of the depth of each correspondent pixel in the reference image. Since it is encoded in only one channel, it will be in grayscale format. Typically close objects (low depth) will be represented in brighter colours (white), whereas far objects (high depth) will be represented in darker colours (black).

Figures 10a and 10b show an image and its respective depth map, generated using *DERS*. Both images are taken from the Rabbit Dataset, captured by the LISA group at Université Libre de Bruxelles (ULB) [23,24,25,26].



(a) View

(b) Depth map

Figure 10: View and depth map

2.3.1 Depth Computation with Regular cameras

When capturing data making use of regular camera, depth must be computed after the acquisition has finished. Indeed, since the depth computation must be performed afterwards the capturing, it won't be possible to use the depth computed in real-time applications. In the case of regular camera, the depth must be estimated, contrary to plenoptic camera, where the depth can be sensed with the light field at capture. Depth estimation techniques generally require the capture of several images of a scene from different positions or viewpoints, and then comparing them.

2.3.1.1 Stereo Matching

Stereo matching [27] is a technique that makes use of the displacement between of the objects that appear in images that capture the same scene from different positions. Since we are talking about images, we can only measure those differences in position in terms of pixels. This difference in pixels

is called disparity.

The disparity is calculated by comparing pixels or windows of pixels along the horizontal line in which they appear. Note that this is only true if the images that are being compared are perfectly aligned. In the case where they are not perfectly aligned, the line that will be followed won't be the horizontal line, but an epipolar line [27]. Regardless of the line that will be followed, the technique is the same: a window surrounding a pixel is taken from one image and moved along that line in the other image(s) until they match. The number of pixels the window has moved is counted, and that is the disparity.

Once the disparity has been calculated, it has to be transformed into real depth. Disparity will be inversely proportional to depth. Indeed, when an object is close to the camera (has low depth), the disparity will be very high, since the object will move quite a lot from one view to the other(s). On the contrary, when an object is far away from the camera (has high depth), the disparity is very low, since the object will remain in a very similar place.

One problems arises because the metrics are different: the depth is continuous and the disparity is discrete. This will make the matching impossible unless the depth is discretized. When discretizing the depth, we talk about depth layers. These layers will be "bigger" when the depth values are high, thus making it more difficult to appreciate depth in objects that are far away. The opposite happen with close objects. Depth can be obtained from disparity using the following equation:

$$Z(d) = \frac{1}{\frac{d}{N-1} * (\frac{1}{Z_{near}} - \frac{1}{Z_{far}}) + \frac{1}{Z_{far}}} \tag{2}$$

where Z is the depth obtained from disparity d, and Z_{near} and $Zfar$ represent the closest and furthest depth layers. N is the total amount of layers.

2.3.1.2 Depth Estimation with Colmap

Colmap can also estimate the depth of the images using a number, generally five, of other images as reference, using a dense reconstruction procedure, also giving a 3D mesh as a final result. It will output both normal and depth maps, both of type photometric and geometric, generating a total of four different maps.

Normal maps refer to the direction of the normal of the objects that are in the scene, having a normal per pixel in this case. On the contrary, depth maps show the depth of each pixel, as explained before. Normal maps will be discarded since they serve no purpose within the present project.

Geometric maps are the ones that pay attention to the shape and geometry of the objects present in an image or scene: contours, texture, shadows, etc. (in other words, features) in order to perform a 3D reconstruction, or to generate a depth map. On the other hand, photometric maps take into account how the objects reflect the light and how much light is received at each pixel to perform the reconstruction.

Refer to Appendix B.5 for further information on installation and operation.

2.3.1.3 Depth Estimation Reference Software

DERS, Depth Estimation Reference Software, as its name implies, is a software that estimates the depth of a given image, creating what is called a depth map. It's a software that belongs to the standard MPEG-I. It has had several revisions throughout the years it has been in development [28,29,30,31]. In the latest versions it is called *RDE*, or Reference Depth-Estimation [32,33].

To compute the depth of the given reference image, several search images are given to the algorithm, as well as the depth range of the image. Then, it uses an algorithm, which consists mainly in three different parts: matching cost, temporal enhancement and graph cut.

For the matching cost part, it will compute, for every search image and depth, the cost of assigning some depth value to a pixel of the reference image leading to the correspondent pixel in the search images, using a modified Sum of Absolute Difference (SAD) algorithm between the reference and the chosen search image, making use of a 3x3 pixel window. The cost has a total of three component, one per channel of a *YUV* image. This process is repeated for all the pixels of the reference image. Then, the chosen cost for each pixel will be the minimal one between the costs with respect to each reference image. Figure 11 shows the idea:

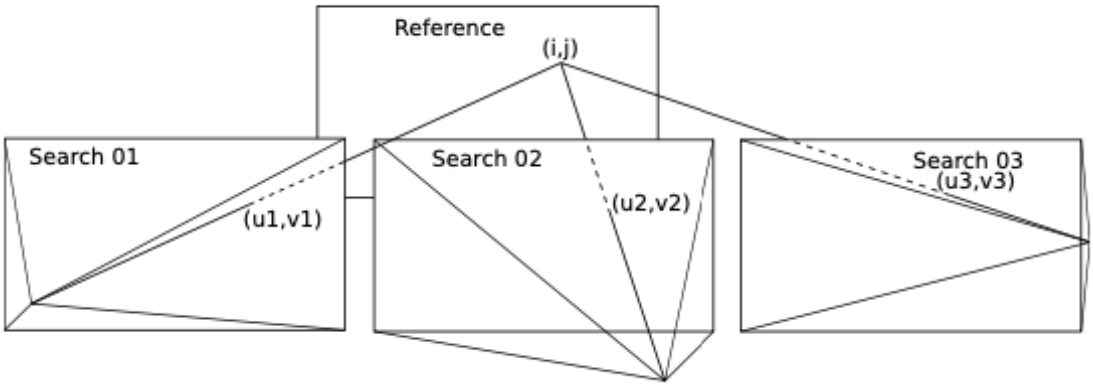


Figure 11: DERS matching cost scheme
[32]

Note that, depending on the baseline (distance between the cameras that take the images), the pixel displacements can vary a lot: ranging from a lot of pixels to less than a pixel, so an accuracy of sub-pixels will be needed.

Next step is the temporal coherence. This part will only be useful with video, since it has multiple frames and, by extension, images. The objective of this part of the algorithm is to make sure that, in two consecutive frames, the same pixel has the same depth. This is done to avoid giving the impression that the pixels' distance changes over time, giving bad results when synthesizing video using the DIBR technique. To achieve this, a motion map is used. This motion map will be set to true when a given window of pixels moves with respect to that same window in the previous frame. This motion of pixels is detected by the difference in their luma components. Then, once it is known which pixels move, the rest, which are static, are forced to have the same depth.

The third and final part of the algorithm, the graph cut, is used to improve the quality of the depth map given by the two previous parts. This first depth map will have different problems, since for each pixel it will have a chosen the lowest cost to assign a depth, so there may be inconsistencies

between values of adjacent pixels. In order to solve this problem, a graph cut optimization algorithm, with a Markov Random Field graph, is used to find the optimal depth map. This algorithm will also make use of reliability and smooth maps, that will help when choosing the optimal depth of untextured areas, since those normally pose problems because the cost will be the same for different depth values.

It takes as input the image the user wants to calculate its depth (reference image) and several more search images (at least two) which will help to tool to generate the depth map. It will need a configuration file in which several parameters will need to be specified, such as the method used, as well as the paths to the input images and to the output files. Refer to Appendix B.3 for further information on operation and installation.

2.3.2 Depth Computation with Plenoptic Cameras

In the case of plenoptic cameras, depth estimation will be performed using the light field that the cameras captures. Of course, the same approaches for depth estimation used for regular cameras can be applied for plenoptic cameras, but the benefits of the plenoptic cameras would not be used, so it is a much better approach to make used of the captured light field.

In the case of the plenoptic 2.0 camera manufacturer RayTrix, a software called *RxLive* [34] is provided. This software allows to perform depth computation at the same time an image is captured, thus depth maps can be obtained in real-time, allowing the possibility of using the sensed depth in real-time applications. Note that there is no exact information on how the depth is computed by RayTrix.

2.3.3 Depth Computation with Azure Kinect

Azure Kinect uses a completely different approach: time of flight. The principle behind it is simple: it projects rays to the scene and captures the time those rays take to go back to the camera. With that time, the distance can be inferred. Time of flight is an active depth-sensing method, since it is actively sending rays to the scene to measure depth.

2.4 Depth Image Based Rendering

Depth Image Based Rendering [27], or DIBR for short, is a technique that has been in development for many years and it allows to generate new synthetic views (or images) of a scene that have not been captured directly by a camera. To put it simple, images captured by virtual cameras (non-existent cameras) are generated by computer using images that have been captured by real cameras placed in certain known positions. In order to do so, other views (or images) of the scene captured by some camera must be provided, as well as depth maps of those same views. With that, one can generate the uncaptured views which are in between the ones that have been captured with the camera. This approach is an alternative to 3D reconstruction of the scene, and it is very useful for different applications, such as 3D video.

The most common approach for DIBR is using arrays of cameras to capture the scene and then synthesize the views in between with the help of depth maps, as mentioned before. The quality of the depth maps has a high impact on the quality of the synthesized views, so it is crucial to use an

adequate and reliable approach when performing DIBR to achieve an acceptable quality.

One of the main problems of view synthesis using the DIBR technique is the occlusion/disocclusion problem. In one view, one object may be hiding what is behind it, but on another view that same hidden part might be visible. Since we need at least two views to infer depth, one must be very careful when capturing them, since the occluded area in one view is disoccluded in the other, thus not making it possible to calculate the depth of that area precisely because it only appears in one view. It could also happen that the view to be synthesized contains a disoccluded area, but the reference views and their depth maps have that same area occluded. This issue can be solved with the capture of more than two views and from different, but still similar, perspectives.

2.4.1 DIBR for Regular Cameras

Regular cameras can only capture one RGB image per shot, so to perform DIBR with regular cameras one must take several shots in order to capture multiview content. With only one camera, the process will be slower: each time a shot is taken the camera must be moved to the new position. The other approach would be to make use of multiple cameras in different positions that take shots at the same time. This second approach is generally the most used one, since it is not possible to capture multiview video with just one camera (unless what is captured in the video is repeatable, which is not normally the case in real applications).

Once the multiview content is captured with the cameras, the depth maps must be estimated. There are several approaches and techniques to estimate the depth of an RGB image, such as stereo matching, but all of them are offline techniques that require a certain amount of time, some of them being very slow in order to produce high quality maps. One example can be the *DERS* software. It produces high quality depth maps from RGB images, but it is quite slow, specially as higher quality is required. This makes it extremely difficult, if not impossible, to perform DIBR with regular cameras in real-time applications. This leaves room for the use of specialized cameras, such as RGB-Depth (RGBD) cameras that use time of flight to generate depth maps in real-time (Azure Kinect), or plenoptic cameras (RayTrix).

2.4.2 DIBR for Plenoptic Cameras

As explained before, plenoptic cameras are devices that allow to capture the light field of a scene by using a microlens array, which in practice translates to capturing an array of n images or views, that have a small displacement between them. Plenoptic cameras allow to export lenslet images, as well as total focus images, which are the ”all-in-focus” version of the lenslet image that has been captured. That, along with the depth estimation they perform, makes it possible to perform DIBR in real-time applications.

2.4.2.1 DIBR using Lenslet Images and RLC

Lenslet images are the main type of image captured by plenoptic cameras. They are the direct result of using a lenticular array in a camera when capturing an image. In general, they are structured in a hexagonal-like pattern, as shown in Figure 12. The shown picture is part of the Rabbit Dataset captured by the LISA group at Université Libre de Bruxelles (ULB).

Figure 12: Lenslet image

Using software such as *RLC* one may obtain an array of *n*x*n* normal images from this lenslet image. With this array of images, one may estimate their depth maps using several offline approaches, in the same way they are estimated when using regular cameras. The main benefit of using lenslet images is that one does not need an array of regular cameras to take multiple images, with just one camera and one shot the user can capture multiview content.

Another possible approach is to use the depth map estimated in real-time by the plenoptic camera, which corresponds to the central view of the array of images. For example, if the lenslet image is transformed into an array of 5x5, the depth map would correspond to image number 13, if they are numbered starting with 1 from left to right, and then from up to down.

Using a set-up with multiple plenoptic cameras, one is able to capture a lot of different views at once using lenslet images. Its main drawback is the price of the cameras since commercial plenoptic 2.0 cameras are very expensive. Transforming lenslet content into multiview images requires time, but since the depth map is already provided by the camera in real-time, the total amount of time is much less than estimating the depth maps offline, which makes it possible to use DIBR in real-time applications.

2.4.2.2 DIBR using Total Focus Images by RxLive

Plenoptic cameras allow to obtain what is called a total focus image after processing the lenslet image, which is, in short, the central view of the array of images obtained from a lenslet image. Figure 10a could be considered the total focus version of Figure 12, although it has been obtained using *RLC*. These total focus images can be obtained in real-time in the case of *RxLive* and RayTrix. Figure 13 is a total focus image, directly exported from *RxLive*.

Figure 13: Total focus image

When performing DIBR with total focus images, one can again use offline techniques to estimate the depth map of the total focus image, given that multiple views of the same scene have been captured from different positions, but this approach is not practical since it is basically the same as performing DIBR with regular cameras using a much more expensive device. The real strenght of the plenoptic 2.0 cameras using total focus images is using the depth map generated in real-time by the camera, such that no time is lost generating normal images from a lenslet picture, nor generating an offline depth map. This approach is the fastest one, thus completely enabling DIBR for real-time applications.

Once again, it has the same drawback as in the lenslet image approach: having multiple plenoptic 2.0 cameras is very costly in terms of money. There exist some approaches to simulate plenoptic cameras by means of normal cameras in order to save money, but at the cost of speed and time. An example of simulation of plenoptic cameras can be found at [35].

It is important to note that, using dedicated software and the plenoptic 2.0 camera's API, one can merge both the lenslet and total focus approaches and gain the benefits of both: the speed of obtaining the in real-time the total focus image and its respective depth map, as well as obtaining the set of sub-aperture views from the lenslet image.

2.4.3 Reference View Synthesizer

The Reference View Synthesizer [36,37,38,39], or *RVS*, is the tool used to generate new views using other reference views and some depth maps. This software belongs to the standard MPEG-I.

The way it works is as follows. First, it needs the camera parameters, i.e. the extrinsic parameters (position, rotation, etc.) and intrinsic parameters (focal length, lens distortion, etc.) for reference image/s and the image/s one wants to synthesize. It will also need the depth maps of the reference images. With that, the tool is able to form, from the reference images, triangles with pixels that have the same depth. Then, those triangles are rotated and translated according to the camera parameters of the reference view and the new view. That may cause some distortion in the new image due to a disocclusion. Next, the image will be upscaled during the rasterization of the warped triangles we just obtained. With that, we will have one new upscaled image per reference view.

In order to improve the quality of the resulting image, the several new upscaled images need to be blended together. To do so, their pixels will be divided into high-frequency and low-frequency. The low-frequencies are then averaged together, whereas the high-frequencies will take as value the pixel of the highest weight. This will result into an image of higher quality.

The final step is to ”fill-in the gaps”. Gaps can happen because the reference views may not include certain parts of the image that the synthesized view would, so that part is unknown. To solve this issue, these gaps will be inpainted, that is, giving those pixels a value according to the values of the surrounding pixels. This will only be performed if the area to be inpainted is small enough. Also, regarding the inpainting algorithm it uses, it is worth noting that it takes into account the depth of the pixels in order to avoid inpainting with values of pixels with a very different depth, which could lead to very bad results if their colours are not similar.

Figure 14 shows two images. The one on the left is the synthesized image without removing the warped triangles. On the right, the disocclusion caused by these warped triangles is removed, but there are some black areas that require inpainting. Examples of these small areas can be found between the legs of the man that is closest to the camera, or the area below the object on the left with a red and white triangles texture.

Figure 14: Warped triangles image vs Clean image
[37]

The tool takes as input the reference views, along with their respective depth-maps. It will also need to use the camera parameters, both for the reference views and the views the user wants to synthesize. All these information will put put in a configuration file, where other options, such as the blending method, will be set by the user according to its needs. Refer to Appendix B.4 for further information on installation and operation.

2.4.4 Previous Works on DIBR with Plenoptic Cameras

In the case of immersive applications, there has not been many research on plenoptic 2.0 cameras. Recently, study for assessing plenoptic 2.0 cameras for real-time DIBR [40] applications was published. In that work, the authors do a study of viability for using plenoptic 2.0 cameras for DIBR in

real-time immersive applications.

The procedure they do for evaluating the quality of plenoptic 2.0 camera depth maps generated in real-time (at least at 30 fps) is very similar to the one that will be presented in this thesis in Section 5, even using the same software. Figure 15 shows the pipeline they use:

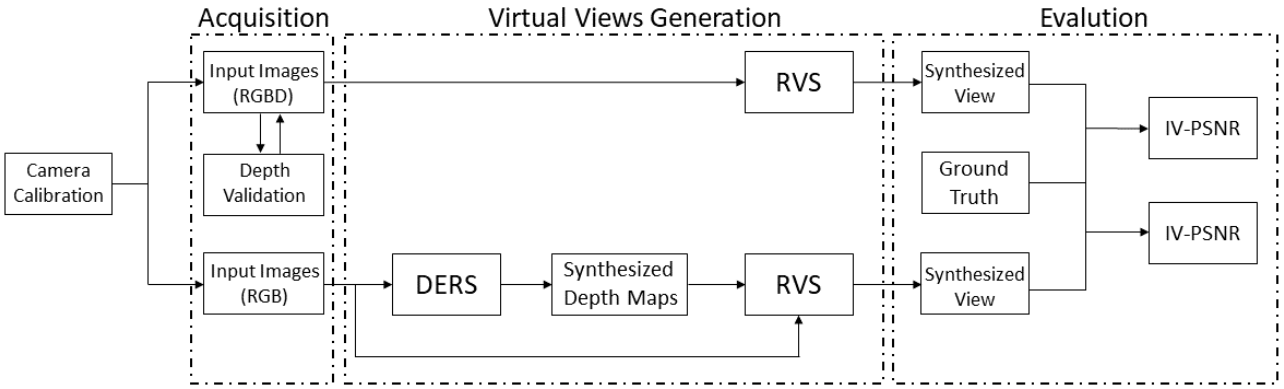


Figure 15: Evaluation pipeline for synthesized views using Raytrix Plenoptic 2.0 camera with its depth (top row) and estimate depth map by DERS (bottom row)
[40]

As shown in the figure, they first calibrate the cameras, then capture a dataset with a RayTrix plenoptic 2.0 camera generating the depth maps and validating them. Then, they synthesize virtual views using the *RVS* tool and the depth maps obtained with the RayTrix camera. In order to assess the quality, they compare it with the synthesized images using depth maps generated with the offline tool *DERS*, which generates high quality depth maps, and then they compare it using IV-PSNR. It is important to note that the depth maps generated by the RayTrix camera need some processing in order adjust to the format accepted by the *RVS* tool.

In their work, Razavi et. al. synthesize virtual views using multiple reference views. That allows to generate the virtual views without occlusion/disocclusion artifacts, and that is how it is done in real applications. In order to do so, they need to perfectly calibrate the camera, both for intrinsic parameters and extrinsic parameters. *RxLive* and *OpenCV* are used for calibration. Even after doing the calibration, their results have some artifacts due to small miscalibration errors, although those artifacts are not very noticeable, thus the quality of their results if quite high, both subjectively, checking the results visually, and objectively, using IV-PSNR. They finally conclude that plenoptic 2.0 cameras are suited for real-time DIBR applications, although further research is needed in the case of video, paying special attention to time coherence artifacts.

In the case of this thesis, since the main objective is to evaluate the quality of depth maps generated by different approaches, DIBR will be performed using only one view as reference, in order to avoid calibration issues that are outside of the scope of this work. Note that using just one view as reference to perform DIBR is enough to assess the quality of a depth map. Having said that, in the Appendices it can be found how to perform camera calibration using different approaches, for the sake of completeness and future developments, as well as in Section 3.5 shows the results of using multiple reference views for synthesis of one virtual view using the different approaches, illustrating the problems that might arise if the calibration is not perfectly done, paying special attention to the plenoptic 2.0 camera approach, whose calibration is harder since the depth map must be adapted and be in conjunction with the camera parameters obtained during camera calibration.

3 Pipeline for Depth Evaluation of Plenoptic 2.0 Cameras and Results

This research targets assessment of plenoptic 2.0 cameras for DIBR, as well as the creation of a solid and reproducible pipeline in order to do so. In fact, this pipeline can be used for assessing the performance of any depth-sensing camera that could potentially be used in DIBR. The present chapter will address all the work that has been done in order to achieve the mentioned objectives. It will be explained, step by step, how the proposed pipeline can be carried out. The pipeline will be divided into four different phases that must be completed sequentially, one after the other, being Acquisition, Depth estimation, View synthesis and Evaluation. In each phase, there are several activities to be performed. Figure 16 shows the proposed pipeline in a schematic way:

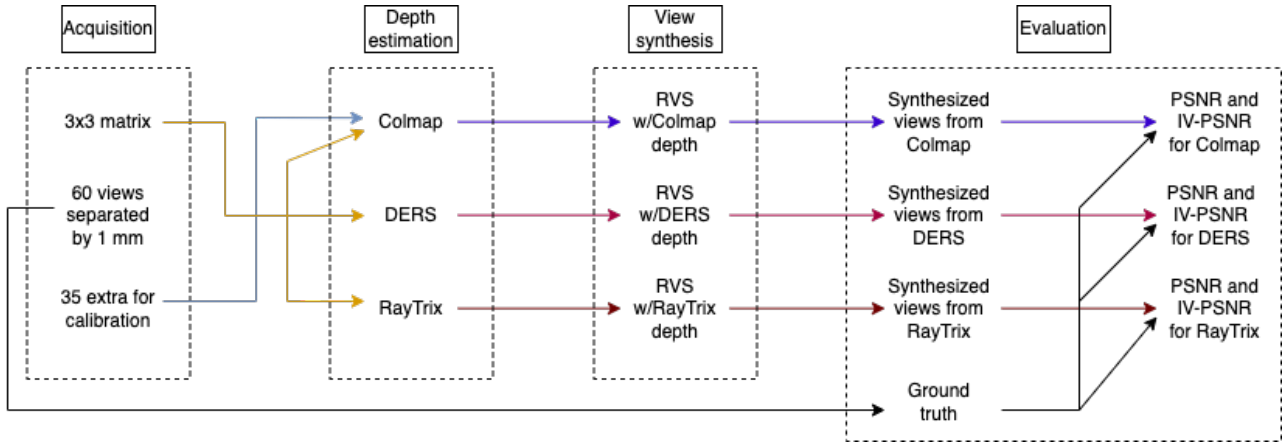


Figure 16: Pipeline of work

The first step is the acquisition. A RayTrix plenoptic 2.0 camera will be used for that, capturing a total of 102 images: a 3x3 matrix (9 images) of views distanced by 3 centimeters, 58 images distanced by 1 millimeters, that will take place in between the images of the second row of the matrix, and finally 35 extra pictures that will be used for calibration and generating a 3D mesh with *Colmap*. Note that the images that will be captured are total focus, instead of lenslet.

Once the full dataset has been captured, then we proceed with the depth estimation. For that, we will use three different approaches: *DERS*, *Colmap* and RayTrix. In the case of *Colmap*, the 3x3 matrix and the 35 extra views will be used to perform a 3D rendering of the scene, that will allow to extract depth maps of the required images. For *DERS*, only the 3x3 matrix will be used as input, obtaining as output the depth maps of those same images. RayTrix, on the other hand, will generate its own depth maps at the time of the capture. They will be processed accordingly to adjust to the format required by *RVS*.

Having obtained both the views and their depth maps, we can continue with view synthesis. Using the central view of the 3x3 matrix as reference view, 30 images to its left and 30 images to its right will be synthesized. The positions of those virtual views are different by 1 millimeter to the right or to the left, depending on the direction. Of course, the furthest virtual views (distanced by 30 millimeters) in each direction will coincide with two of the views of the 3x3 matrix, since those are separated by 3 centimeters. For the little experiment with multiple views as reference, the central view of the matrix will be generated, using the corners of the matrix as reference.

Finally, in the evaluation step, both PSNR and IV-PSNR will be performed, using the images captured by the RayTrix as ground truth.

3.1 Acquisition of Datasets with RayTrix

In this section, it will be explained how the datasets of images have been captured, showing the acquisition setup and the specifics of the RayTrix camera used.

3.1.1 RayTrix Camera

The RayTrix camera used is the RayTrix R8 [41], with a lens of 25 millimeters focal length. It is a multi-focused plenoptic 2.0 camera. It has the capability of changing the lens to other with different focal lenses, using c-mount. Its maximum frame rate is 30 frames per second (fps) and it has a lateral resolution of 2 Megapixels. It has a micro-lens array (MLA) aperture F/Number of 2.8. Its pixel size is 2.24 microns, and it has an electronic shutter of the rolling type, with global start. It is connected to the PC using the high speed USB 3.0, its image sensor is made by Toshiba and the 4D plenoptic sensor baseline is quite small (XS, according to its technical details).

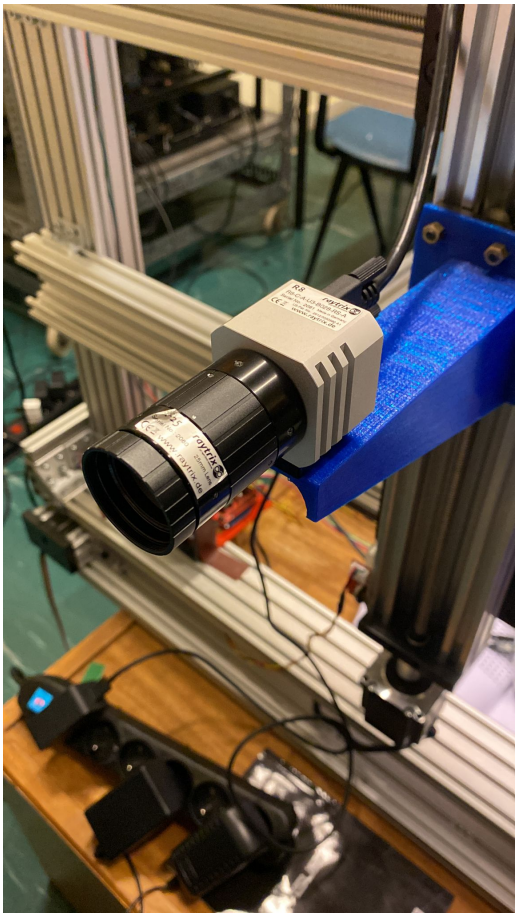


Figure 17: RayTrix R8 plenoptic 2.0 camera

In order to use it, one needs the software *RxLive*. This software is able to capture image and video. It can generate real-time lots of different data: lenslet image, total focus image, depth maps, point clouds, 3D images, 3D meshes, etc. One may modify the parameters to suit many different needs, and it can be done online during capture or offline, after capture. Consult the Appendix B.1

or the RayTrix website for further information.

Before performing the acquisition, the RayTrix camera must be calibrated so that it perceives depth correctly in order to generate correctly depth maps, 3D meshes, etc. Further information can be found in the Appendix C.3.

3.1.2 Acquisition Setup

In order to capture all the required images for the experiment, an acquisition setup is required. The RayTrix camera has been attached to a robot to enable precise 3D movement. That enables knowing the position of the camera at all times, so that the matrix of 3x3 images can be captured maintaining the distance of 3 centimeters between all views at all time. That same robot has been used to captured the images that distance from each other 1 millimeter, since its accuracy is high enough. The robot can be controlled from the PC with which the *RxLive* software is operated, so the pictures can be taken each time the robot is moved.

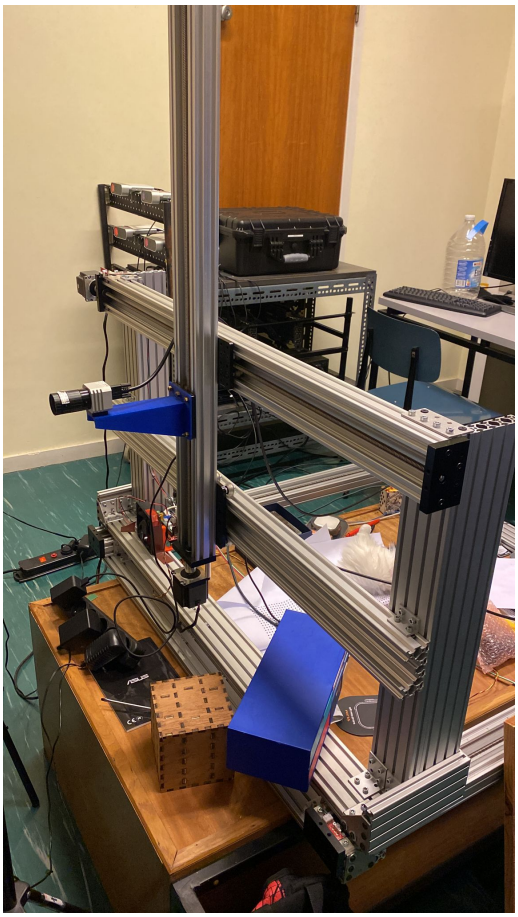


Figure 18: Acquisition robot with the RayTrix camera attached

In the case of the 35 extra views, those have been taken ”by hand”. That is, the camera has been detached from the robot and faced manually towards the scene. In this case, it is not important the position of the camera, since the purpose of this extra pictures is that *Colmap* can use them to generate its 3D reconstruction, since with the 3x3 matrix is not enough because the images are too similar between them.

The scene has been set in table in front of the robot. The camera is about 1.1 meters from the wall, and the object from the scene that is closest to the camera is about 0.8 meters away. A total of four lamps have been used in order to illuminate correctly the scene.

Figure 19: Scene ready to be captured

Figure 20: Complete acquisition setup

3.1.3 Dataset

As mentioned before, the dataset is comprised of a total of 102 images. All these images are total focus instead of lenslet, since it is faster to acquire total focus than capturing lenslet content and then transforming into a matrix of regular images using *RLC*. Also, for the purpose of the experiment, there is no need to obtain multiview content per shot, with just the total focus picture is enough.

The dataset can be divided into three parts: a 3x3 matrix of images (9 in total) separated by 3 centimeters each. These images are numbered starting with 1 from left to right, then top to bottom, so the central view is number 5. Then, 58 images with 1 millimeter separation and 35 extra shots taken by hand. The 58 images separated by 1 millimeter can be divided into two groups of 29 each: 29 are in between the central view of the matrix (view 5) and the one to its left (view 4), and the other 29 have the same distribution but on the right direction, so they are placed in between views 5 and 6. Figure 21 shows how those images are distributed graphically:

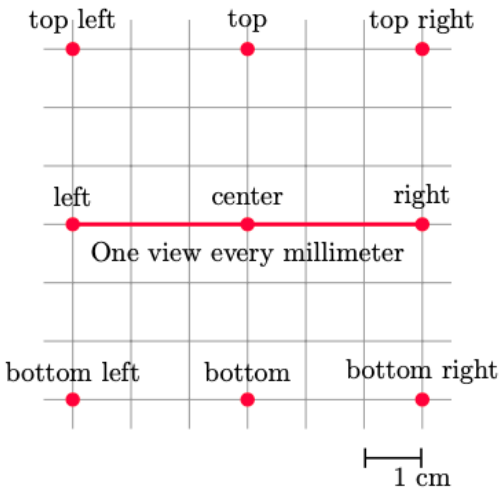


Figure 21: Main distribution of the dataset

The red dots correspond to the 3x3 matrix images: where top left would be view 1, center view 5 and bottom view 8, all of them separated by 3 centimeters in both X and Y directions. On the other hand, the red line represents the 58 1 millimeter-separated images.

The 3x3 matrix will be used for generating the depth maps using *DERS*, the row of 61 views displaced by 1 millimeters (58 + 3 (central, left and right)), for view synthesis, using the central view as reference and the rest as ground truth when doing the evaluation of performance. Also, the corners (views 1, 3, 7 and 9) will be used to synthesize view 5 (center) in the multiview experiment. The central view of the matrix corresponds to Figure 22. As observed in the figure, the field of view (FoV) of the RayTrix camera is very narrow, taking into account that the distance from the camera to the wall is 1.1 meters and not all the objects appear completely in the picture.

Figure 22: Central view of the matrix

Finally, the 35 remaining images, only serve the purpose of helping *Colmap* reconstruct a 3D model of the scene. With that reconstruction, one can export camera parameters that can be used for view synthesis, as well as for generating depth maps with *DERS*.

It is also worth mentioning that the dataset that has been explained is the main one, but several other have also been captured in order to experiment, and some of the images from other datasets will be shown along the document.

All the images are of size 1920x1080 pixels. They have been exported in *JPEG* format with 24-bit encoding from *RxLive*. This encoding has been chosen over 64-bit with *PNG* because *Colmap* does not work properly with 64-bit images.

3.2 Depth Estimation

Once the whole dataset has been acquired, we can proceed with the next step, that is depth estimation. As mentioned before, three different approaches to obtain depth maps will be used: using the MPEG-I software *DERS*, the open source 3D reconstruction software *Colmap* and RayTrix online depth estimation, using *RxLive* to process and export the data.

3.2.1 Depth Estimation using Colmap

As mentioned in the previous sections, *Colmap* is an open source software that offers a general-purpose Structure-from-Motion and Multi-View Stereo pipeline. It can do many things, but in the case of the experiment, it will be use for two purposes: estimate the camera parameters, both intrinsic and extrinsic, and perform 3D reconstruction, enabling extraction of depth maps for the images we need.

Colmap will make use of all the 102 images of the dataset. Once those have been inputted, the reconstruction can be started. A type of camera must be chosen in order to perform the reconstruction. "Simple pinhole" camera model is chosen, since, a priori, the images taken by the RayTrix have no distortion and the camera parameters are unknown. Before the reconstruction, the features of the dataset must be first extracted and then matched. Once that is done, we can perform the 3D reconstruction:

Figure 23: Colmap's 3D reconstruction

With the reconstruction complete, the camera parameters can be exported as text. Then, using the script *colmap_to_json.py* from the *RPVC* pipeline, we can transform the parameters to *JSON* format, which will be usable by *DERS* and *RVS*. Note that it will output the camera parameters for all the cameras: they correspond to each one of the 102 images used for the reconstruction. The parameters for the 35 extra images will be discarded, since they won't be needed. The intrinsic parameters will be the same for all images since the same camera has been used to capture them all, and the extrinsic parameters will reflect the position and rotation of each one of them. In the case of the extrinsic parameters, *Colmap* extracts them "up to scale". That is, the units of the parameters are not known. In the case of this project, the position metrics were very similar to decimeters (assuming some small error), so decimeters was the unit used in all the configuration files.

After the 3D reconstruction is complete, we can start with the dense reconstruction. This will output a 3D model that can be visualized with an external tool, but also the depth maps for all the 102 pictures. It will output two types of depth maps: photometric and geometric, as explained the in the *Colmap* section. Figures 25 and 26 show the photometric and geometric depth maps respectively, along with the reference view in Figure 24. Hotter colours mean that the objects are far from the camera, whereas colder colour signify the objects are close to it, both in photometric and geometric depth maps.

As it can be noted, both depth maps have missing data. It is more noticeable in the geometric depth map, since the missing data is coloured in black, whereas in the photometric one it looks like noise. This happens whenever *Colmap* finds a texturless area, such as the faces of the cubes which

are uniform in colour. In order to try to solve it, the window size patch of the algorithm used to generate the depth maps can be increased up to size 20 (by default it's of size 5).

Figure 24: Reference view (Colmap)

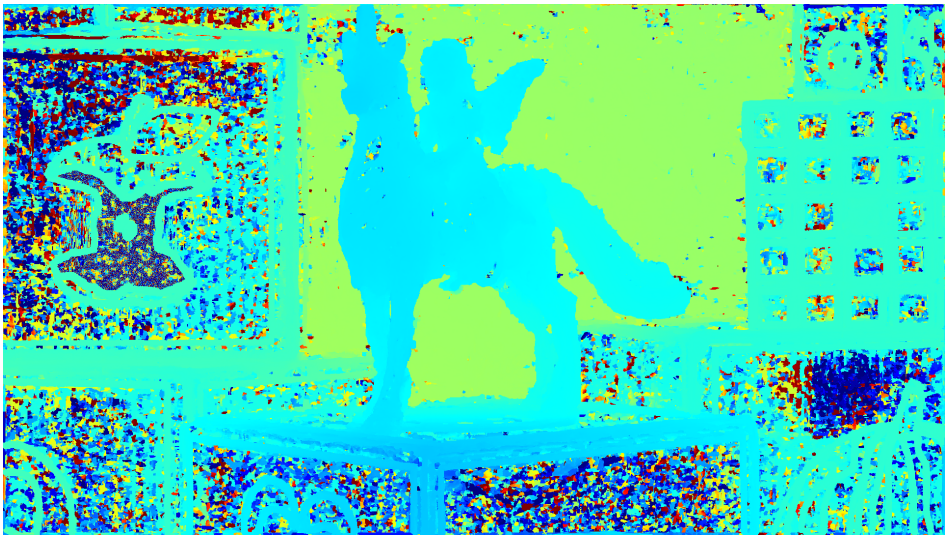


Figure 25: Colmap's photometric depth map

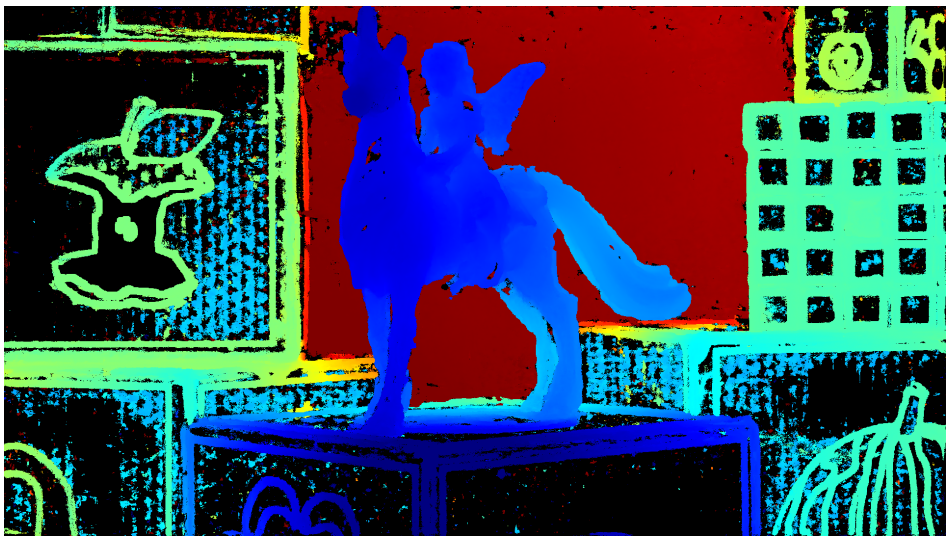


Figure 26: Colmap's geometric depth map

After increasing the size to 20, the depth maps obtained are shown in Figures 27 and 28.

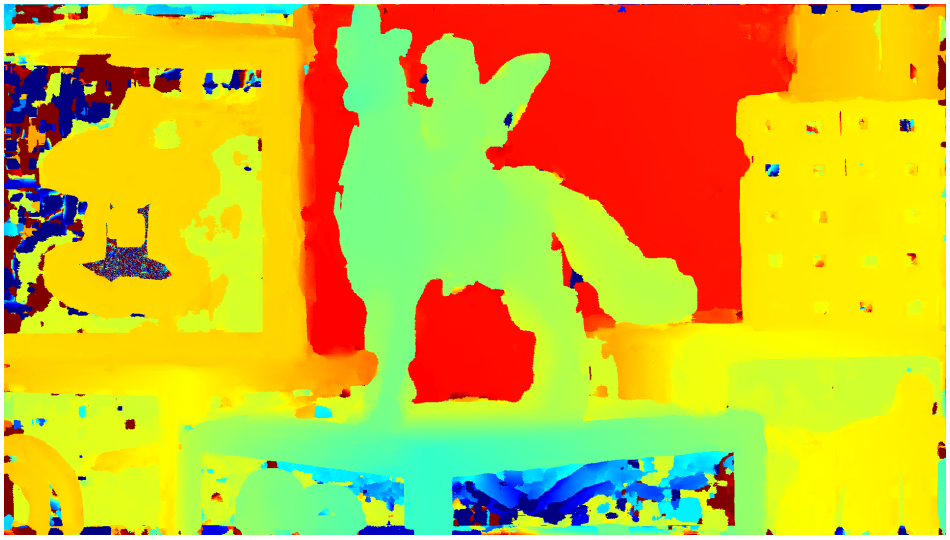


Figure 27: Colmap's photometric depth map (patch size 20)

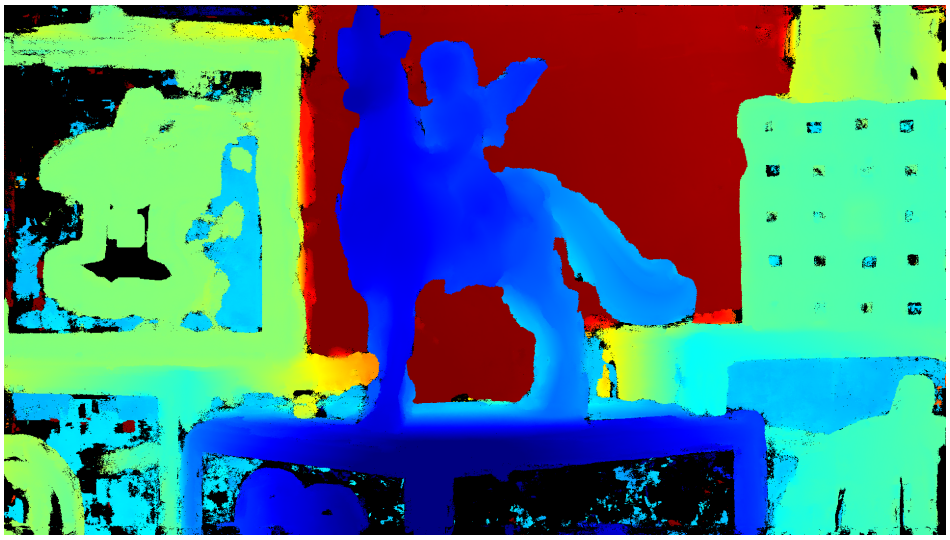
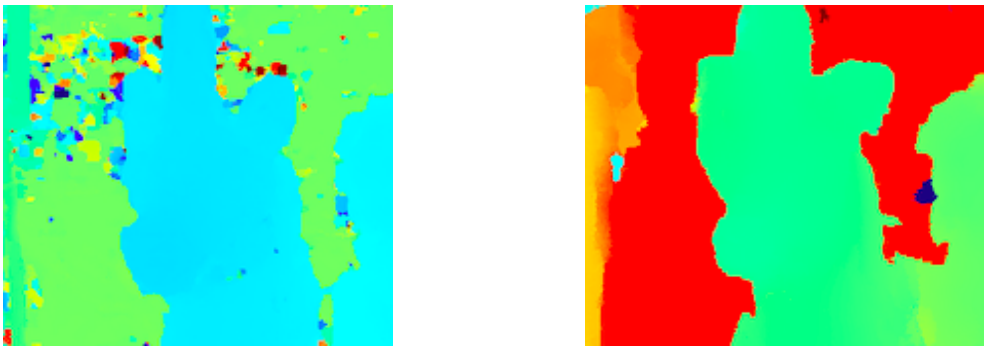


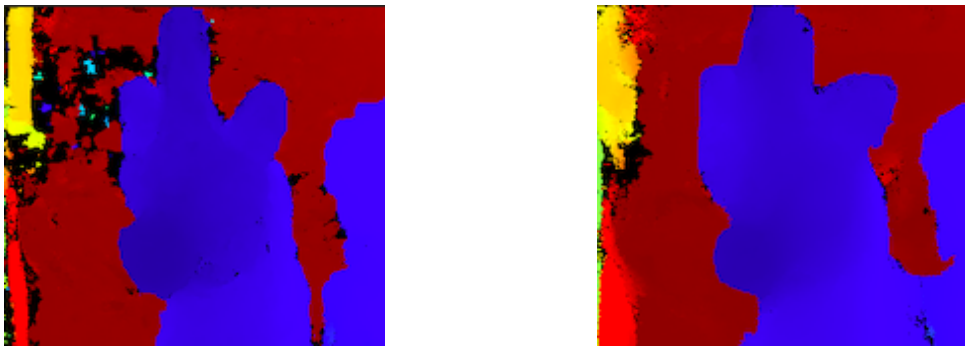
Figure 28: Colmap's geometric depth map (patch size 20)

Once again, even after increasing the patch size to the maximum size supported, the depth maps still have missing data. Another notable thing is the loss of quality and sharpness: the unicorn in the patch size 20 depth maps appears bigger than it actually is. This is specially notable in the head, more precisely in the horn and the ears. That happens in both photometric and geometric depth maps:



(a) Patch size 5 (b) Patch size 20

Figure 29: Unicorn's head, photometric

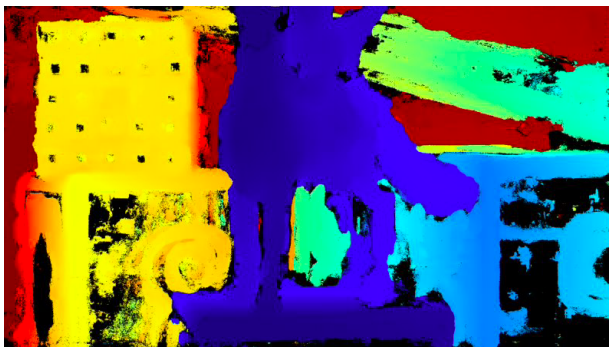


(a) Patch size 5 (b) Patch size 20

Figure 30: Unicorn's head, geometric

Since the areas using patch size 20 are relatively small and are inside regions (borders in general are well defined, regardless of the patch size), we decided to inpaint the depth maps using *OpenCV* [42]. It offers two different algorithms: Telea (method from Alexander Telea) and NS (Navier-Stokes based method). After several experiments, the Telea method works better with the depth map generated using window size patch 20. Note that this inpainting method will only work properly because the edges of the objects are well delimited and it won't mix depth values that are not within the region.

In general, *Colmap*'s depth maps have good quality: the objects and edges are well defined, and it works very well when the objects are very textured. On the other hand, with textureless objects, it does not work as well: there is a lot of missing data, and some regions may not be well defined, mixing with other regions. Figures 31a and 31b show an example of this: the pencil is mixed with the background in the depth map, leading to the same result in the synthesized view.



(a) Depth map

(b) Synthesized view

Figure 31: Pencil image

Note that the dense reconstruction to obtain the depth maps is quite demanding for the computer and takes quite a lot of time, specially in the case of using the maximum window size patch, which takes much more time to complete than using a window size patch of 5.

The generated depth maps are in binary format, so to transform them the script called *dense_to_exr.py* from the *RPVC* pipeline has been used to change its format to *EXR*, which is accepted by *RVS*. Note that not every depth map will be used: only the corners of the 3x3 matrix (views 1, 3, 7 and 9) and the central view (view 5) are useful.

3.2.2 Depth Estimation using DERS

To generate depth maps using *DERS*, the camera parameters exported previously from *Colmap* will be used. In this case, only four five maps will be generated: views 1, 3, 5, 7 and 9 from the 3x3 matrix. *DERS* will use what are called search views in order to synthesize the depth map of the reference view. In order to do so, all 9 images from the matrix will be used as search views in all five depth maps synthesis. In general, it would suffice to use two search views and the reference view, but to increase quality we decided to use all 9 pictures.

In order to generate them, the images have been transformed to *YUV420* format using *FFmpeg* [43], which is an open software to perform multimedia conversion between formats. The output files are in format *YUV400_16le*, which uses 16-bit low endian enconding. It is a greyscale format. Brighter colours mean the objects are close to the camera, whereas darker ones mean the opposite. In the case of the color black, it means there is missing data, the same as in *Colmap*'s depth maps.

The depth maps generated using *DERS* and *Colmap*'s camera parameters will be on the same scale and units as the parameters, so no need for special treatment.

Figure 32: Reference view 5 (DERS)

Figure 33: DERS depth map (view 5)

DERS also takes quite some time to finish generating one depth map, and it is also quite demanding on the computer. But, as can be seen in Figure 33, the result is very good, smooth and precise. It is also worth noting that it needs texture to perform well, the same as *Colmap*, as there can be seen slight differences in depth in areas where there should be no difference, such as in the rubik cube, or some drawings on the cubes, like the apples or the bananas.

3.2.3 Depth Estimation using RxLive

The third and final approach for generating depth maps is using RayTrix's native depth maps, exported from the *RxLive* software. In this experiment, *RxLive 5.0* has been used. In this case, the file exported is the coloured depth map, in greyscale, to match the *DERS* output format.

In this case, special treatment is needed, since RayTrix uses its own scale, as mentioned in Section 3.3. Once the depth map values have been normalized according to the RayTrix scale using the formula presented in [40] and that is depicted in Equation 3, they can be used in multiview synthesis, since they will now match the scale of the camera parameters, which is the real physical scale. In the case of view synthesis using only one reference, there is no need to normalize them. The reason behind it is that no synchronization of the position of multiple reference images (to synthesize one view) is needed, thus there won't be any misalignment between the objects.

$$D_m(x,y) = Z_{min} + \frac{D_{RayTrix}(x,y) * (Z_{max} - Z_{min})}{255} \tag{3}$$

Where D_m represents the normalized depth of each pixel, $D_{RayTrix}$ the depth assigned by the RayTrix to each pixel, and Z_{min} and Z_{max} the minimum and maximum values that the RayTrix assigned to its depth map.

Figure 34: Reference view 5 (RayTrix)

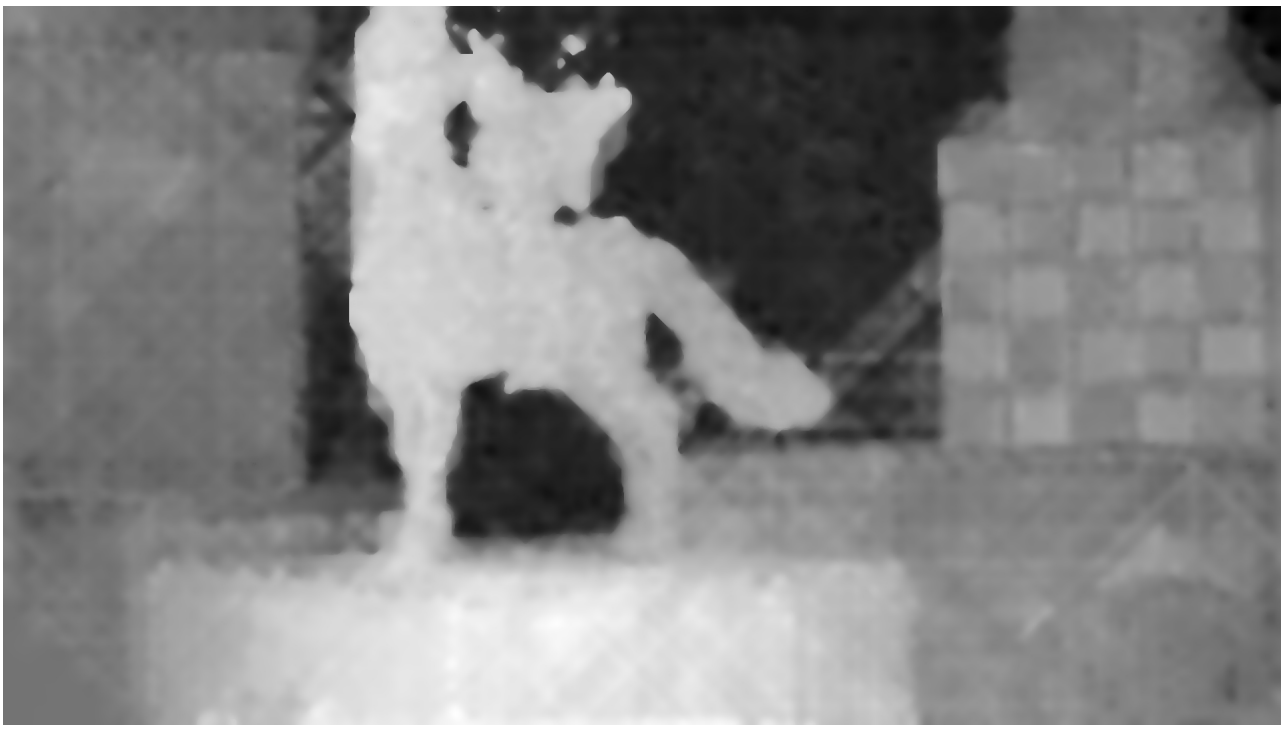


Figure 35: RxLive depth map (view 5)

These depth maps are captured at the same time as the total focus images. They only need some post-processing to improve the quality using *RxLive*.

As seen on Figure 35, the quality of the depth map is very good, although is has some small artifacts due to the refraction of the light and how it is captured by the micro-lens array. Those small artifacts can be eliminated, but at the cost of losing quality on other areas. Once again, brighter colours mean proximity and darker colours mean bigger distance.

3.3 View Synthesis

With the depth maps generated, view synthesis can be performed. Only one view will be used as reference, being this one the central image of the 3x3 matrix. With that, 60 virtual views will be synthesized, 30 to the left of the reference and 30 to the right, using *RVS* in all the cases. This view synthesis will be performed three times: one using *Colmap*'s depth maps, another using *DERS* depth maps and, finally, a third one using RayTrix native depth maps.

As mentioned previously, the synthesized images will be 1 millimeter away from each other, so the first one will be separated 1 millimeter from the reference view to the left, the next one separated 2 millimeters to the left, until a distance of 30 millimeters (or 3 centimeters) is achieved, in both left and right directions. The performance is expected to be similar in both directions. The reference image, along with the images displaced 5mm and 30mm to the left are depicted in Figures 36, 37 and 38.

In each section, it will also be discussed the quality of the synthesized views, that will be the measure of the quality of the depth maps generated by each approach, since the quality of the depth is crucial when performing DIBR.

Figure 36: Reference view

Figure 37: View moved 5mm to the left

Figure 38: View moved 30mm to the left

3.4 Objective Quality Assessment

In this section, it will be discussed objectively using metrics well known in the industry. Note that in most cases, the subjective method of quality assessment through visual checking takes preference over objective methods such as PSNR. The reason for this is that, objective methods only take into account differences in pixel value, but visually one can clearly see if there is something wrong with the image or not. There might be the case where the objective measure is high, but visually one clearly notices very important defects that have a big impact in realism.

3.4.1 Quality Measures

The quality measures chosen will be two: PSNR and IV-PSNR. Both are very similar metrics, being the first a more general approach and the second the specialized approach for immersive video.

3.4.1.1 Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio, or PSNR for short, is a metric that has been used for quality assessment of images. It is a ratio between the maximum value a pixel can take and the noise that can effect the quality. It is measured in decibels (dB), and the higher the value is, the higher quality the image has. It can be calculated using the following equation:

$$PSNR = 10\log_{10}(\frac{MAX^2}{MSE}) \tag{4}$$

Where *MAX* is the maximum possible value a pixel can take (this will depend on the encoding), and *MSE* is the *Mean Squared Error*.

In the case of colour images, since each bit has three channels, the *MSE* is the sum over all squared differences for each colour. Another alternative could be to transform the image to another colour space, and then use *MSE* normally.

Regarding the values of the metric, any image below 20dB has a bad quality, between 20dB and 30dB any image would get a pass, between 30dB and 40dB the image quality is good and above 40dB it's extremely good.

3.4.1.2 Immersive Video Peak Signal-to-Noise Ratio

Immersive video PSNR, or IV-PSNR, is a new metric, which is the version of PSNR adapted to the needs of immversive video. It was presented in [44]. Its main difference with normal PSNR is that, instead doing it pixel by pixel, it uses a window to perform pixel shift in order to find the pixel, within that window, more suitable to perform the comparison with the ground truth image. It also incorporates a global component difference, in order to address any changes that may occur globally in one image (or frame of a video) and that are correct. The formulation to calculate IV-PSNR is the following:

$$IV - PSNR = 10\log_{10}(\frac{MAX^2}{IV - MSE}) \tag{5}$$

MAX is once again the maximum possible value that a pixel in an image can take, and IV-MSE is the immersive video mean squared error. The difference between IV-MSE and normal MSE is that it takes the difference between the value of the selected pixel and the pixels within a specified window such that the error is minimized, thus choosing the most similar pixel of the window.

In general, it outperforms PSNR in immersive video, and it can also be used for other applications that are not in the field of immersive video, virtual reality, etc.

The tool used for measuring each one of the synthesized images can be found in the following repository [45]. It can perform PSRN, IV-PSNR and WS-PSNR, but only the first two will be taken into account. Graphics for each approach will be shown, representing the drop of quality in PSNR or IV-PSNR as the distance increases.

3.4.2 Objective Assessment of View Synthesis using Colmap Depth Maps

In the case of *Colmap*, two graphics are presented: one for PSNR and another one for IV-PSNR. The six different approaches have been grouped together, so they can be compared more easily.

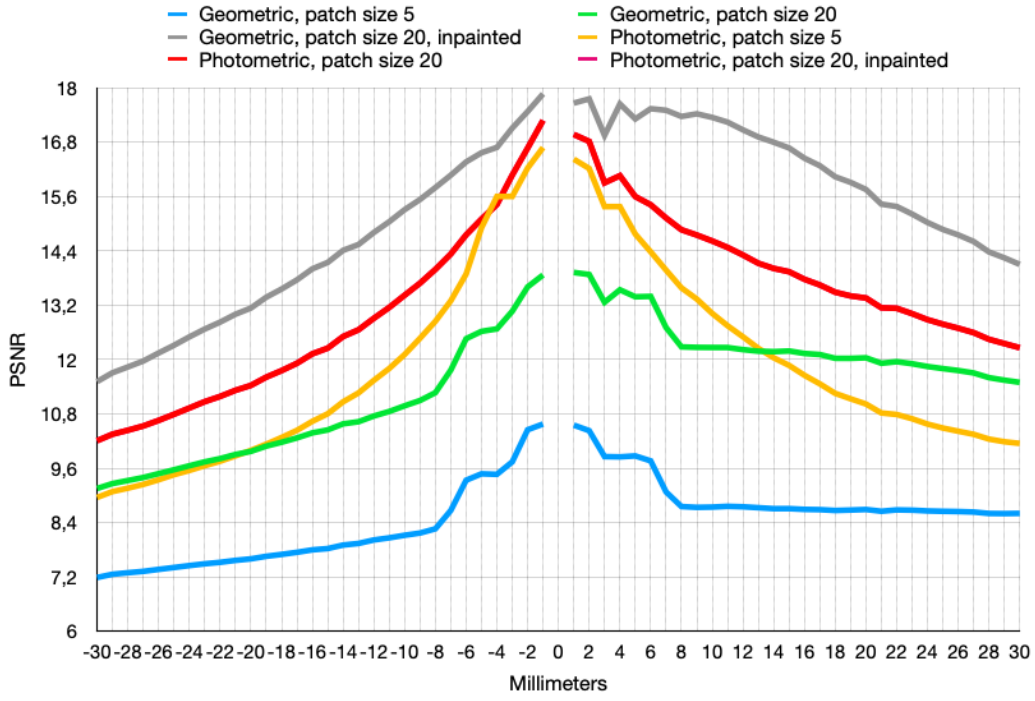


Figure 39: Colmap, PSNR

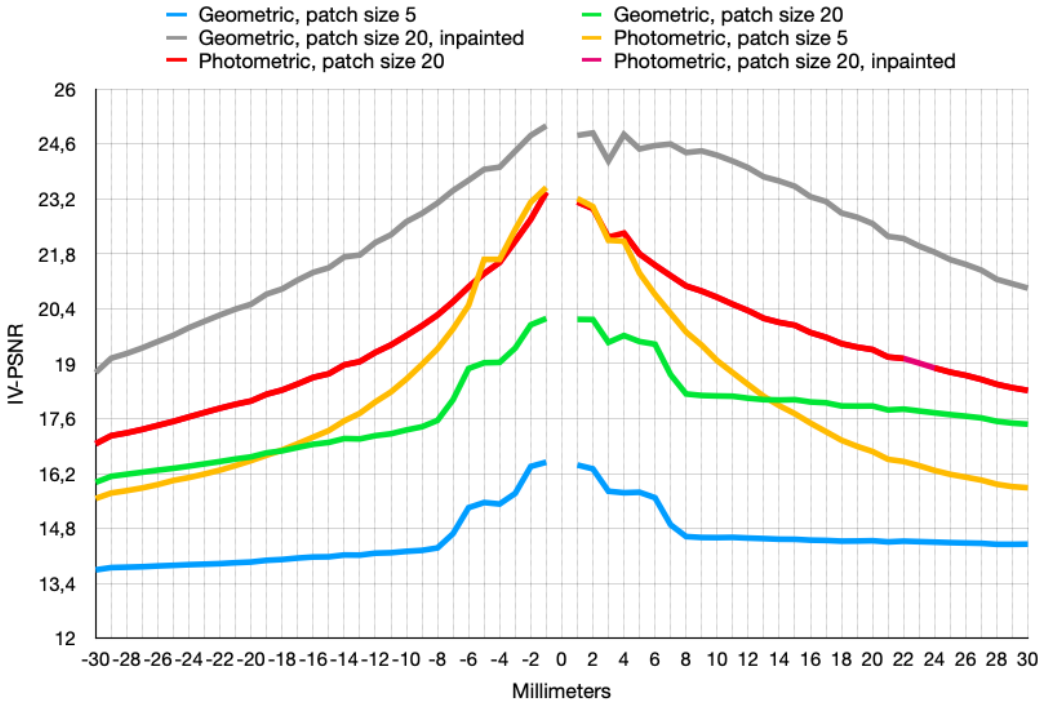


Figure 40: Colmap, IV-PSNR

As expected, both graphs are almost equal, mostly sharing the shape. IV-PSNR values are higher, since the metric is more permissive than regular PSNR. Regarding the results, the best one is yielded by geometric inpainted depth maps, with window size patch of 20. This matches the visual assessment, since subjectively they also have the best quality. In the case of photometric depth maps with patch size 20, there is no difference in the metrics, both in PSNR and IV-PSNR, since both lines have the same value. That means the inpainting has not had any positive effect on the view synthesis.

On the other hand, photometric depth maps clearly outperform geometric depth maps without inpainting, since photometric, even with window size 5, greatly outperforms geometric with size 20. One possible reason for this is that geometric depth maps have more missing data than photometric, thus when synthesising views that missing data will lower the metric. Objectively, it seems it is better to have the noise of different colours generated by the use of photometric depth maps than the missing data, even though after the visual check it would be preferable to have missing data instead of generalized noise. On the other hand, synthesizing to the right seems to have a slightly better performance, both in PSNR and IV-PSNR, specially noticeable in geometric depth maps.

In general, the results are quite low, being the maximum value of geometric inpainted depth maps of 17.87 and 24.82 for PSNR and IV-PSNR respectively. This is due to the missing data in the synthesized views in textureless areas, meaning heavier processing of the depth maps is needed in order to generate high quality virtual views.

3.4.3 Objective Assessment of View Synthesis using DERS Depth Maps

In the case of *DERS*, we have the following graph:

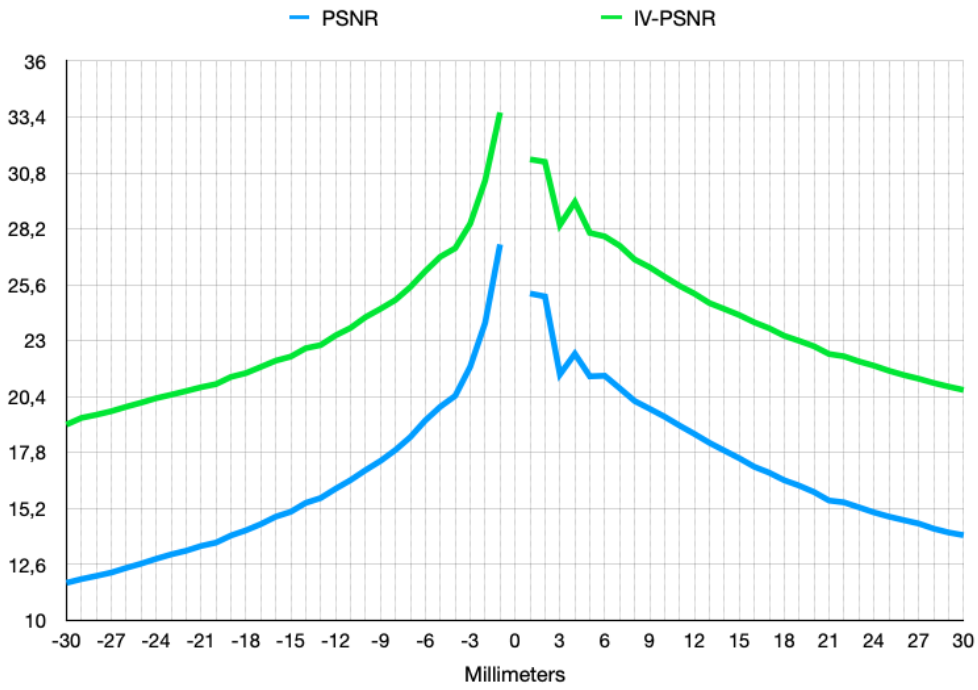


Figure 41: DERS, PSNR and IV-PSNR

In this case, both PSNR and IV-PSNR values are higher when the synthesized view is close and on the left of the reference view. On the other hand, the right hand side has a slightly better performance when generating views far from the reference. The values themselves are quite higher, compared to the ones obtained after using *Colmap*'s depth maps, being the maximum values of 27.48dB and 33.61dB for PSNR and IV-PSNR respectively. Another thing to highlight is that, on the left side, the quality drops very quickly as the distance increases, referring to close distances (between 1mm and 5mm). That is also the case in the right side, but between 1mm and 3mm. The main reason for that is that the disocclusion artifact forming the "shadow" of the unicorn is starting to appear.

When synthesizing to the right, the drop of quality in closer distances is more irregular, since there are views that are further away than others but have higher values, and quality by extension. This might be result of the physical structure of the scene, yielding a difference in performance depending on the direction of the synthesis.

3.4.4 Objective Assessment of View Synthesis using RxLive Depth Maps

The PSNR and IV-PSNR values for the views synthesized using RayTrix's native depth are represented in Figure 42. Note that it is not the same image as the one shown in Figure 41, there are subtle differences.

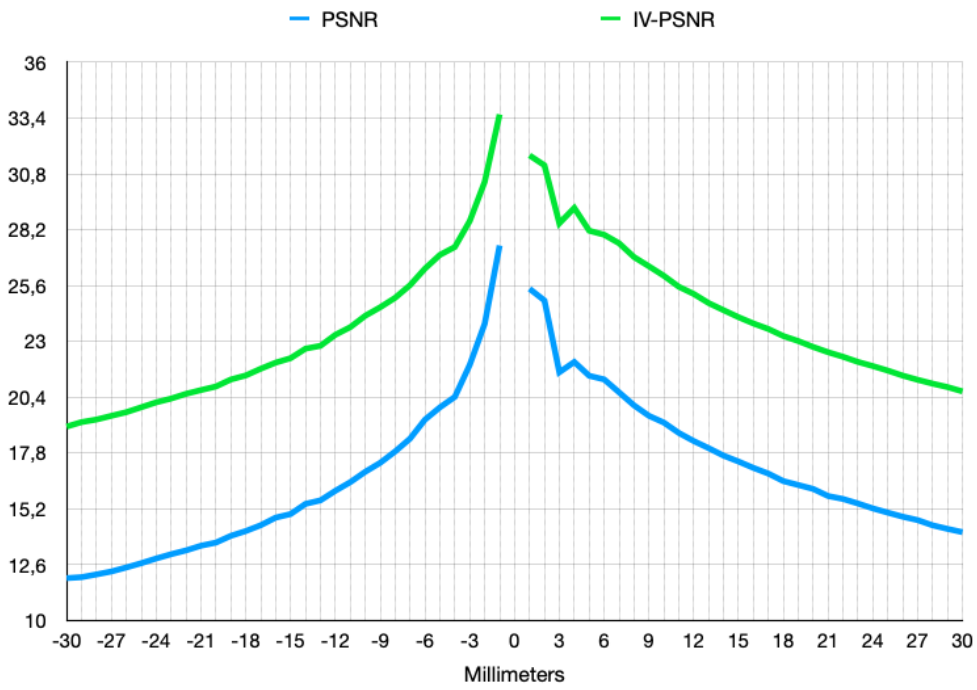


Figure 42: RayTrix, PSNR and IV-PSNR

As observed in the figure, the curves are almost exactly the same as the *DERS* ones. Maximum values correspond to the view distanced 1mm to the left of the reference: 27.47db for PSNR and 33.57dB for IV-PSNR. Those values are extremely close to the ones obtained with *DERS*, and quite higher compared to *Colmap*. The reason behind such similarity in values between *DERS* and RayTrix might be the disocclusion artifacts that appear in *DERS* synthesized images in the form of a "shadow" of the unicorn. Indeed, the performing both PSNR and IV-PSNR in the pixels of that area, the result will drop significantly. In the case of RayTrix it will remain higher because the synthesized views do not have big disocclusion artifact, they are much smaller, and the area has been warped to fit in, yielded much higher values after calculating the metrics for those regions.

We can arrive at the same conclusions we arrived with *DERS*: the right side is more irregular, probably due to the physical characteristics of the scene captured, and on the left side the performance decreases faster with the distance, being specially noticeable in the closes range of proximity (5 to 1 millimeters).

After the objective quality assessment, apparently the depth maps generated by RayTrix are of a quality equivalent to the ones generated by *DERS*, although they will need some processing to adjust the scale of the values to the one of the camera parameters. But, subjective quality assessment says otherwise: *DERS* synthesized views are much sharper in edges, specially as the distance increases (see next section). On the other hand, *Colmap* produces good quality depth maps, but they are very limited due to the fact that they have a lot areas with missing data in zones were the texture is uniform. Taking into account that *Colmap* is an open source software, there is room for evolution and improvement, making it possible to address this issue in the future. Another solution might be to perform heavier processing in the depth maps than the simple inpainting that has been performed, leading thus to better results, since the basics like edge detection and right depth detection are there.

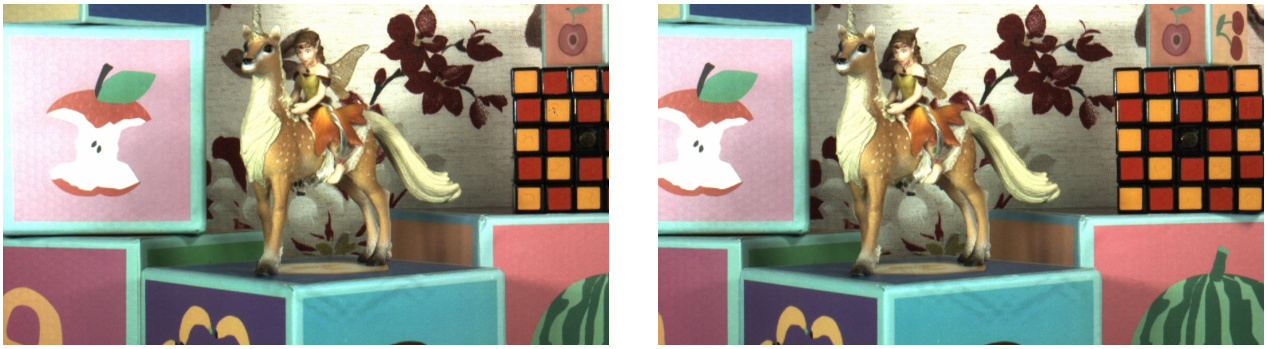
3.5 Subjective Quality Assessment

Subjective quality assessment will be performed by visualization of each one of the synthesized views, paying special attention to specific areas such as objects with much detail and texture, surfaces with not much texture and edge.

3.5.1 Subjective Assessment of View Synthesis using Colmap Depth Maps

In the case of *Colmap* we have a total of six possibilities: photometric and geometric, with window patch size of 5, 20 and 20 with inpainting. With that we can check the quality of each one of the possible depth maps, and then later decide which one is more suitable depending on the application.

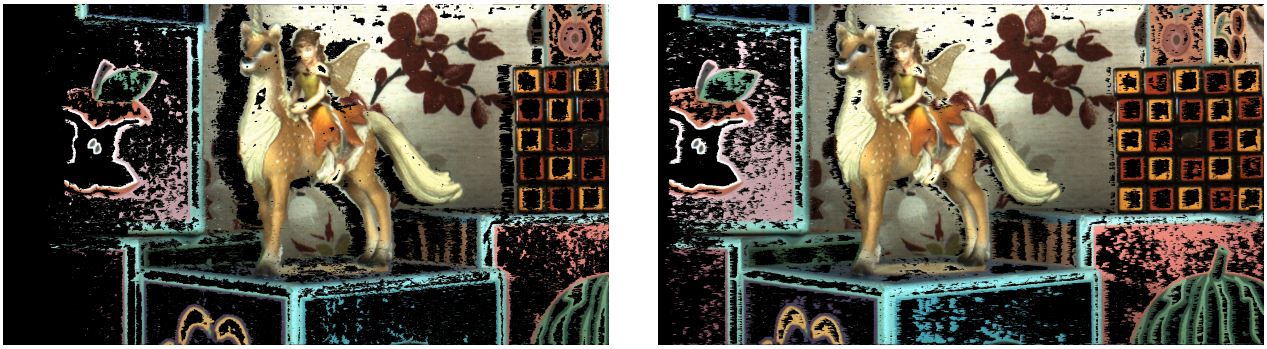
Two images to the left direction will be shown using each approach, with distances of 5 millimeters and 30 millimeters.



(a) 30 mm to the left

(b) 5 mm to the left

Figure 43: Ground truth (Colmap)



(a) 30 mm to the left

(b) 5 mm to the left

Figure 44: Geometric, window patch size 5

(a) 30 mm to the left

(b) 5 mm to the left

Figure 45: Geometric, window patch size 20

(a) 30 mm to the left

(b) 5 mm to the left

Figure 46: Geometric, window patch size 20, inpainted depth map

(a) 30 mm to the left

(b) 5 mm to the left

Figure 47: Photometric, window patch size 5

(a) 30 mm to the left

(b) 5 mm to the left

Figure 48: Photometric, window patch size 20

(a) 30 mm to the left

(b) 5 mm to the left

Figure 49: Photometric, window patch size 20, inpainted depth map

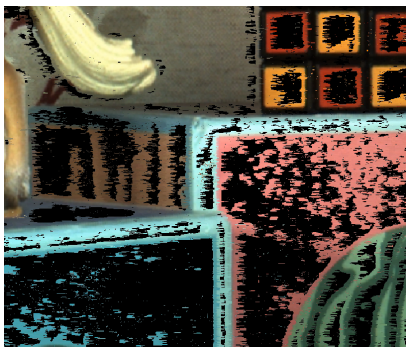
Magnified regions are shown next in order to better show detail in areas with a lot of texture (unicorn), edges and areas with little texture (edges and faces of cubes):

(a) 30 mm to the left

(b) 5 mm to the left

Figure 50: Geometric, window patch 5, unicorn

(a) 30 mm to the left



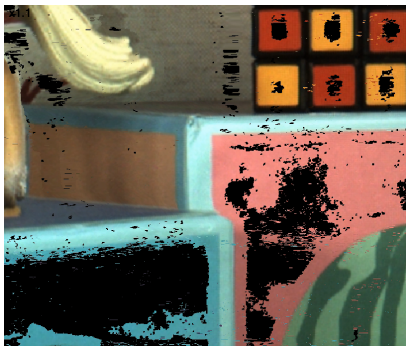
(b) 5 mm to the left

Figure 51: Geometric, window patch 5, edges and flat areas

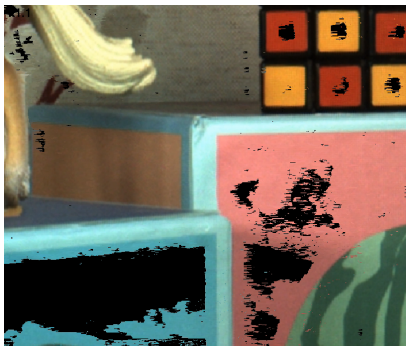
(a) 30 mm to the left

(b) 5 mm to the left

Figure 52: Geometric, window patch 20, unicorn



(a) 30 mm to the left



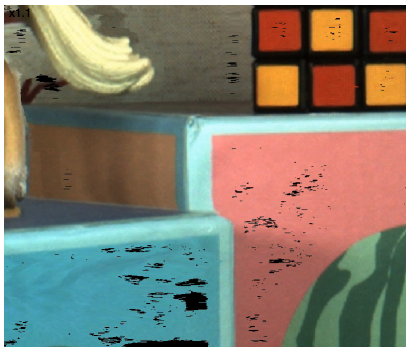
(b) 5 mm to the left

Figure 53: Geometric, window patch 20, edges and flat areas

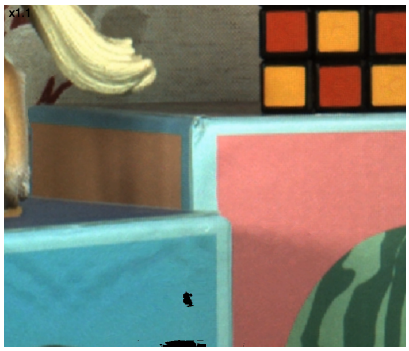
(a) 30 mm to the left

(b) 5 mm to the left

Figure 54: Geometric, window patch 20, inpainted depth map, unicorn



(a) 30 mm to the left



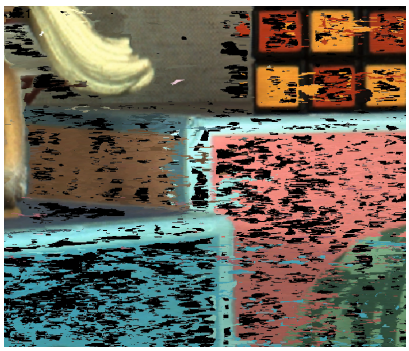
(b) 5 mm to the left

Figure 55: Geometric, window patch 20, inpainted depth map, edges and flat areas

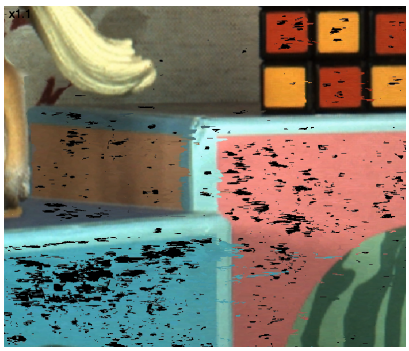
(a) 30 mm to the left

(b) 5 mm to the left

Figure 56: Photometric, window patch 5, unicorn



(a) 30 mm to the left



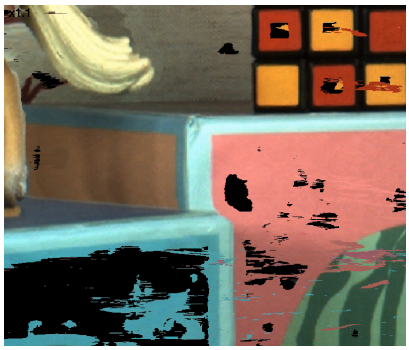
(b) 5 mm to the left

Figure 57: Photometric, window patch 5, edges and flat areas

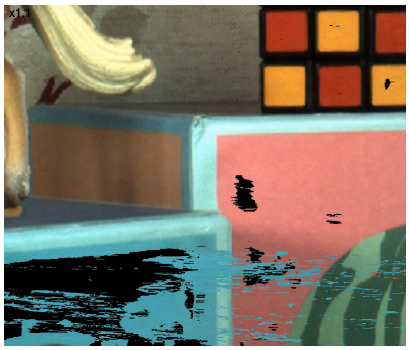
(a) 30 mm to the left

(b) 5 mm to the left

Figure 58: Photometric, window patch 20, unicorn



(a) 30 mm to the left



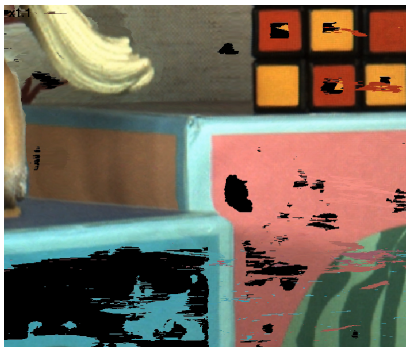
(b) 5 mm to the left

Figure 59: Photometric, window patch 20, edges and flat areas

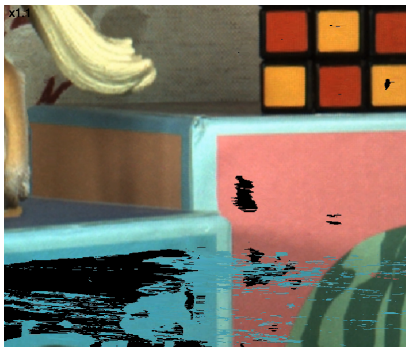
(a) 30 mm to the left

(b) 5 mm to the left

Figure 60: Photometric, window patch 20, inpainted depth map, unicorn



(a) 30 mm to the left



(b) 5 mm to the left

Figure 61: Photometric, window patch 20, inpainted depth map, edges and flat areas

As seen in the images, and being compared with the ground truth, we can say, subjectively, that the quality of the synthesis increases with the window patch size, specially in the case of the inpainted result, where all the objects are almost perfect. On the other hand, the performance of the geometric depth maps is better in general, since photometric maps produce noise in the form of ”moved colour” in the synthesized views, as it can be appreciated specially in Figure 49a. It is also worth noting that areas with no texture still remain uncolored. That happens because the depth maps have no values in there.

On the other hand, as the distance is increased, the quality of the synthesis decreases. This becomes more noticeable with the black band on the left of the images, as well in the ”shadow” of the unicorn that is very noticeable in 30 millimeters away synthesized views. The black band on the left is there because the reference image does not have information about that area, since it is not in the picture. Regarding the unicorn ”shadow”, its reason of appearance it disocclusion, since that area was occluded in the reference by the unicorn and now it is not, hence the artifact takes the form of the object that was occluding it, the unicorn. These artifacts appear not only using *Colmap*'s depth maps, but also using *DERS* and *RxLive* depths.

While the evaluation remains subjective, it seems that *Colmap* performs well when generating depth maps for view synthesis, specially when using geometric depth maps. Its main drawback is the lack of data when dealing with texturless areas, as it can be observed in the images. But, it is able to correctly identify borders and assess correctly the depth of the objects, as long as it has enough reference images to perform the estimation.

3.5.2 Subjective Assessment of View Synthesis using DERS Depth Maps

In the case of *DERS*, we no longer have that many options, there is only one choice, which is using the unique output the software gives.

(a) 30 mm to the left

(b) 5 mm to the left

Figure 62: Ground truth (DERS)

(a) 30 mm to the left

(b) 5 mm to the left

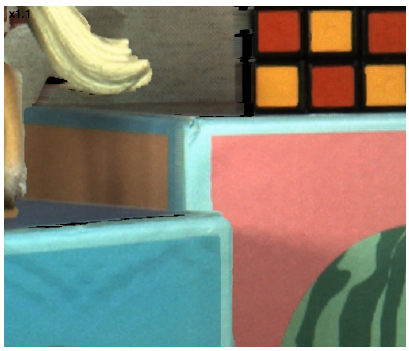
Figure 63: DERS

Magnified regions are shown next in order to better show detail in areas with a lot of texture (unicorn), edges and areas with little texture (edges and faces of cubes):

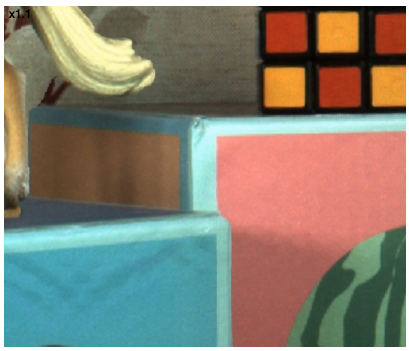
(a) 30 mm to the left

(b) 5 mm to the left

Figure 64: DERS, unicorn



(a) 30 mm to the left



(b) 5 mm to the left

Figure 65: DERS, edges and flat areas

Using *DERS* as a tool to generate depth maps, we can visually check that the synthesized views are almost perfect. The only noticeable artifacts are the disocclusion of the background behind the unicorn and the missing data on the left. It seems that, after subjective evaluation, *DERS* produces high quality depth maps, as it is implied since it is a tool has been in development for quite some time and belongs to the MPEG-I standard.

3.5.3 Subjective Assessment of View Synthesis using RxLive Depth Maps

In the case of RayTrix native depth maps, when exporting coloured depth maps from *RxLive*, the user can modify several parameters, such as the depth algorithm, the maximum and minimum depth or the number of iterations among other things, in order to obtained the best quality depth map, and this can be performed real-time or offline. In the case of this work, it has been offline, after the capture of the dataset.

(a) 30 mm to the left

(b) 5 mm to the left

Figure 66: Ground truth (RayTrix)

(a) 30 mm to the left

(b) 5 mm to the left

Figure 67: RayTrix

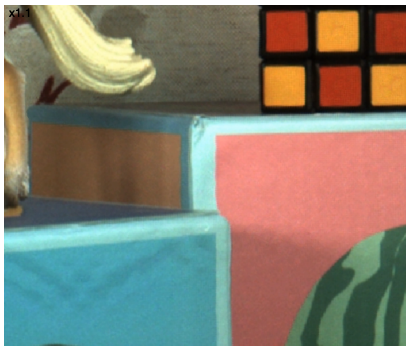
Magnified regions are shown next in order to better show detail in areas with a lot of texture (unicorn), edges and areas with little texture (edges and faces of cubes):

(a) 30 mm to the left

(b) 5 mm to the left

Figure 68: RayTrix, unicorn

(a) 30 mm to the left



(b) 5 mm to the left

Figure 69: RayTrix, edges and flat areas

As observed in the figures, the quality of the synthesized views is quite high, specially when the distance is not high. As the distance increases, the straight lines are not straight anymore, but almost every detail of the objects is kept. Again, there is the black band on the left, and, in the case of the disocclusion artifacts, there is no black "shadow" of the unicorn. Instead, the pattern on the background is stretched to fill the gaps, even though some small holes remain. This might happen because *RVS* tries to inpaint holes in synthesized views if the those holes are not big. Again, after subjective assessment, it seems that RayTrix native depth maps are of good enough quality, even

though visually their results seem inferior to those obtained using *DERS*, specially with bigger distances.

3.6 View Synthesis for Virtual Reality

In this section, a more practical approach will be taken to assess the quality of depth maps generated by the three different ways mentioned before. For virtual reality, more than one view must be used as reference in order to avoid missing data artifact like the black bands on the sides seen previously, as well as avoiding disocclusion artifacts like the ”shadow” of the unicorn.

In this experiment, image synthesis has been performed using multiple views as reference. The central view of the matrix has ben synthesized using the corners of the multiview matrix as reference. Figure 70 shows it schematically.

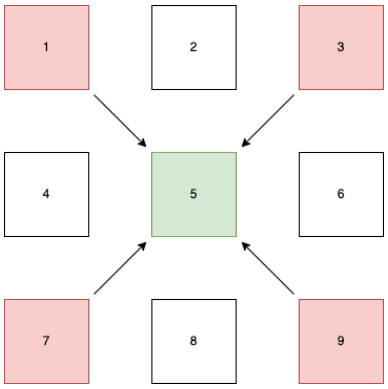


Figure 70: Multiview synthesis for view 5 scheme

Once again, there are six synthesized images using *Colmap*'s depth map, one for *DERS* and one for RayTrix native depth.

Figure 71: Reference view 5, ground truth

(a) Geometric

(b) Photometric

Figure 72: Colmap, window size patch 5

(a) Geometric

(b) Photometric

Figure 73: Colmap, window size patch 20

(a) Geometric

(b) Photometric

Figure 74: Colmap, window size patch 20, inpainted depth map

(a) Geometric

(b) Photometric

Figure 75: Colmap, window size patch 20, inpainted depth map, inpainted synthesized view

(a) DERS

(b) RayTrix

Figure 76: DERS and RayTrix

Magnified regions are shown next in order to better show detail in areas with a lot of texture (unicorn), edges and areas with little texture (edges and faces of cubes):

(a) Geometric

(b) Photometric

Figure 77: Colmap, window size patch 5, unicorn

(a) Geometric

(b) Photometric

Figure 78: Colmap, window size patch 20, unicorn

(a) Geometric

(b) Photometric

Figure 79: Colmap, window size patch 20, inpainted depth map, unicorn

(a) Geometric

(b) Photometric

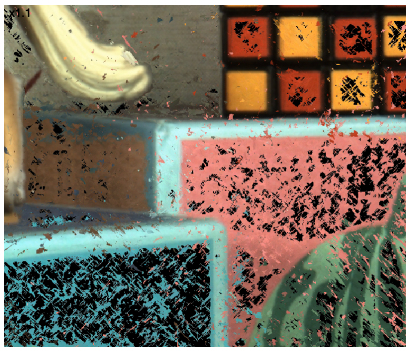
Figure 80: Colmap, window size patch 20, inpainted depth map, inpainted synthesized view, unicorn

(a) DERS

(b) RayTrix

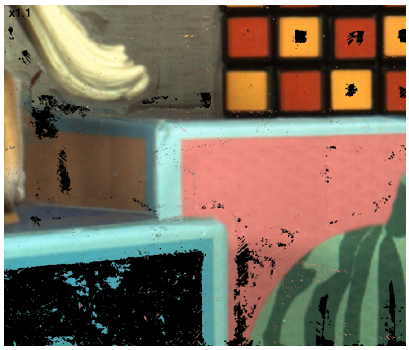
Figure 81: DERS and RayTrix, unicorn

(a) Geometric



(b) Photometric

Figure 82: Colmap, window size patch 5, edges and flat areas



(a) Geometric

(b) Photometric

Figure 83: Colmap, window size patch 20, edges and flat areas

(a) Geometric

(b) Photometric

Figure 84: Colmap, window size patch 20, inpainted depth map, edges and flat areas

(a) Geometric

(b) Photometric

Figure 85: Colmap, window size patch 20, inpainted depth map, inpainted synthesized view, edges and flat areas

(a) DERS

(b) RayTrix

Figure 86: DERS and RayTrix, edges and flat areas

In the case of *Colmap*, the image quality keeps increasing with the image patch size, although surrounding the unicorn there are some noticeable artifacts that are due to the loss of sharpness in the depth maps. Even after inpainting depth maps of patch size 20, in the synthesized views there are still some small areas with missing data. That has been solved inpainting the synthesized views once again, as shown in Figures 75a and 75b. With small patch size, the holes in the synthesized views are very big, as expected, but the objects are very well defined, with no noticeable artifacts in or around them. In the case of photometric depth maps, there is some noise in the form of the wrong colour, same issue as with just one reference view, so the conclusion would be to always use geometric depth maps if possible, or find a way to improve the quality of photometric depth maps

after they have been generated by *Colmap* but before using them for view synthesis with *RVS*. Note that the inpainting method for synthesized views is working because the gaps are very small and within regions of the same colour that are well delimited. It is worth mentioning that in Figures 75a and 75b colour is different from the rest because, in order to inpaint them, they were transformed to *PNG*, since *OpenCV* does not support *YUV* format.

Using *DERS*, visually the results are good in general. The only artifacts that appear are surrounding objects, specially in the around the unicorn. This is happening because in the depth maps, even though their quality is good, in each view it has a different value. One possible solution would be post-process the depth maps and put certain pixels to zero, so that *RVS* does not take those values into account when synthesizing the view. The pixels that should be zero are obtained comparing the synthesized view with the ground truth, and then making a mask which is the result of the subtraction of the values of one image to the values of the other, being those values above a certain threshold.

In the case of RayTrix native depth estimation, the result is not good. It clearly suffers of the so called ghosting effect. That type of artifact happens when the views are not correctly aligned, so it puts the objects in several places at the same time. Being not correctly aligned means there are calibration issues, thus concluding that the camera parameters that have been generated by *Colmap* have some small error that is causing the ghosting effect. This is only happening using *RxLive* exported depth maps because those are the only ones that use their own scale, *DERS* and *Colmap* generated depth maps use those same camera parameters, thus they are on the same scale. In order to remove the ghosting effect, another calibration method to obtain camera parameters should be used. In [40], they make use of *OpenCV* calibration and they obtained good results. It has not been performed in this work because calibration is out of the scope of the thesis, multiview view synthesis has been performed as a complementary experiment.

Objective measurement of the synthesized images using PSNR and IV-PSNR has also been performed. The results are shown in the following table, being ordered from higher to smaller values, taking precedence IV-PSNR:

Method	**PSNR**	**IV-PSNR**
DERS	26.2700 dB	33.3505 dB
Colmap, geometric, inpainted synthesis	26.0597 dB	32.4207 dB
Colmap, photometric, inpainted synthesis	25.3011 dB	30.9426 dB
RayTrix	20.9254 dB	27.1370 dB
Colmap, geometric, inpainted depth	23.6127 dB	26.0597 dB
Colmap, photometric, inpainted depth	16.6326 dB	22.5722 dB
Colmap, photometric, size 20	16.6326 dB	22.5722 dB
Colmap, geometric, size 20	12.7498 dB	19.2263 dB
Colmap, photometric, size 5	12.4751 dB	18.2232 dB
Colmap, geometric, size 5	8.5058 dB	14.6883 dB

Table 1: PSNR and IV-PSNR values for the synthesized views.

Objectively, in terms of PSNR, the best result is yielded by *DERS*, followed by *Colmap*'s geometric depth map with window patch size of 20 inpainting both the depth map and the output image. RayTrix depth falls behind both *DERS* and *Colmap*'s best performances, although this is expected, since it has ghosting effect. With respect to *Colmap*'s depth map with small patch size or without inpainting, the PSNR values are very low, but it is expected since it has quite a lot of missing values. Contrary to what visually can be said, photometric depth maps yield better PSNR values when using

low patch size or no inpainting than geometric depth maps.

With respect to IV-PSNR values, we get to the same conclusion: *DERS* comes on top, followed closely by *Colmap*'s inpainted geometric depth map with window size 20 and inpainted result. The remaining methods mantain the same positions as with regular PSNR. As seen before, IV-PSNR values are higher than regular PSNR. The values themselves are not very high either, not reaching 34dB in any case. The only IV-PSNR value below 30dB is the one corresponding to the RayTrix. In general, the use of the three different approaches is usable in multiview synthesis, but needs further refining to obtain top-quality synthesized views.

4 Comparison of Depth Maps by RayTrix, and Azure Kinect vs DERS for DIBR

In this section, a comparison with another depth-sensing device will be addressed. We will be comparing the performance of the RayTrix camera against the Azure Kinect.

In the master thesis "Evaluation of the Azure Kinect depth sensor for view synthesis" [46], Hoet et. al. perform an evaluation of the Kinect for view synthesis, very similar to the work explained in this manuscript. Indeed, they use the same metrics for objective quality evaluation, and the captured dataset is the same: 3x3 matrix, 58 1 millimeters separated views and 35 extra pictures for calibration using *Colmap*. Note that the dataset captures the same scene captured in this work, and in the exact same positions for the matrix and the 1mm distance, using the same robot. Indeed, that will enable the comparison.

Figure 87: Capturing setting

The objective of this comparison is to address the strengths and weaknesses of each device compared to the other, since it is not possible to perform a completely objective comparison because the two cameras have a very different field of view: the one of the RayTrix is very narrow, whereas the Kinect's one is very wide. Thus, we will focus on the difference of IV-PSNR values of synthesized views using the native depth maps with respect to the ones obtained using *DERS*, always using only one view as reference for the synthesis, as explained previously. Note that here we are using *DERS* as a quality measure. On the other hand, we will also check subjectively certain areas of the synthesized images, such as edges, flat textureless areas and object detail.

First, we will address IV-PSNR values. As it can be observed in Figure 88, IV-PSNR values both for RayTrix and Kinect depth synthesized views are shown, along with *DERS* synthesized views using the two different cameras. As shown previously, the quality for *DERS* (RayTrix) and the RayTrix depth maps have an extremely similar quality, thus almost overlapping. In the case of the Kinect, the virtual views generated using Kinect's depth map have worse quality than the ones generated using *DERS* depth maps. This difference is represented in Figure 89. The difference in quality between

RayTrix and *DERS* is almost 0, whereas in the case of the Kinect the loss of quality is much higher, specially in views close to the reference. On the other hand, the performance of the Kinect is very robust, since the loss of quality of far away views with respect to closer ones is very low: 28.75dB with respect to 32.5dB on the left side, and 29.01dB with respect to 32.5dB on the right.

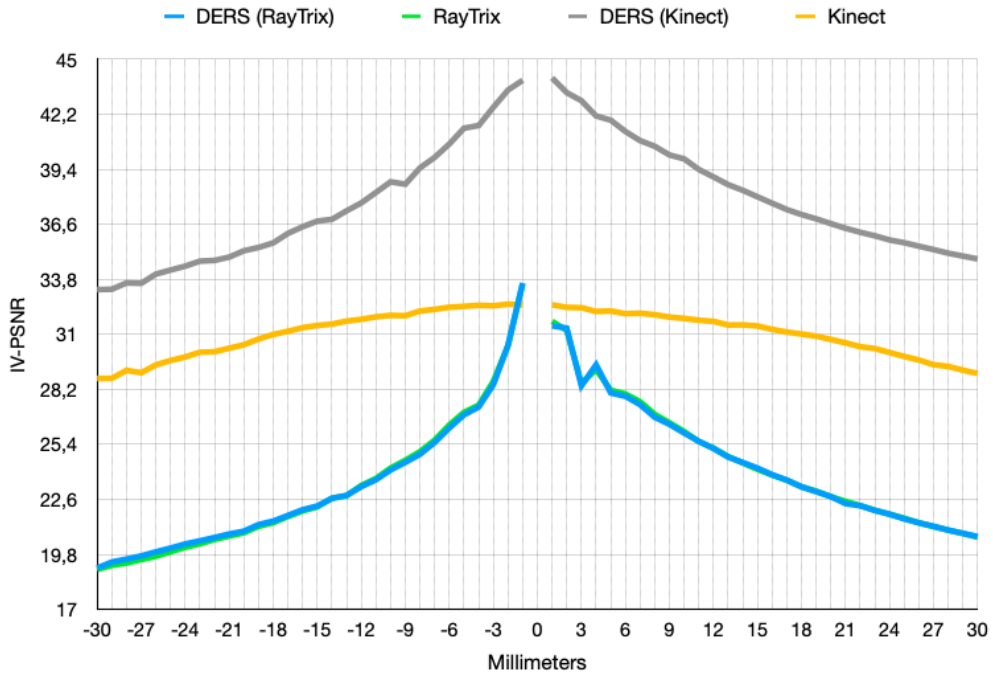


Figure 88: RayTrix, Kinect and DERS, IV-PSNR

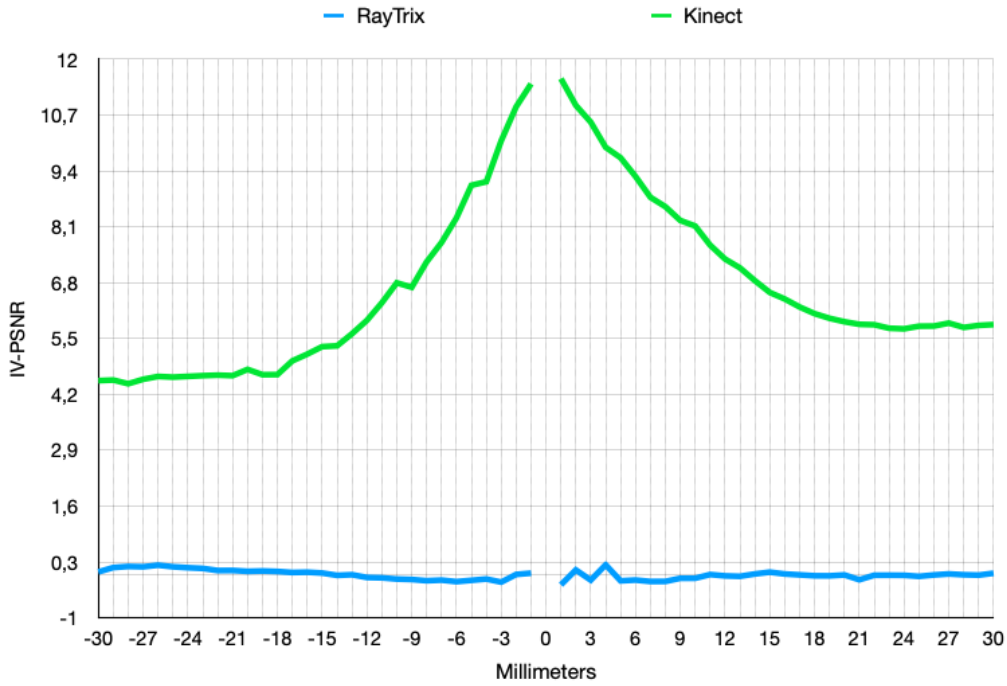


Figure 89: Loss of quality between RayTrix and Kinect with respect to DERS

Regarding the big difference in quality of *DERS* using RayTrix as capturing device and *DERS* using Kinect, it is probably due to the difference in the field of views of both cameras. Since the

RayTrix FoV is much more narrow, it will capture only the objects of the scene, and with much more detail. On the other hand, the Kinect, with a much wider FoV, will also capture the surrounding area of the scene, at the cost of close detail in objects. We must also take into account that the image sizes are not the same: RayTrix images are of size 1920x1080 pixels, whereas Kinect's are 2048x1536 pixels. Note that the Kinect images have been cropped from 4096x3072 to half that size, the original images have a FoV of 120°. This cropping has been made in order to make it easier to perform the visual comparison with RayTrix-captured images. Another big reason for such difference in quality is the aforementioned disocclusion artifacts that appear in the rendered views from the RayTrix, specially in the case of using *DERS* depth maps. That also makes the loss of quality between RayTrix and *DERS* almost 0.

To illustrate the difference in FoV of both cameras, even after cropping Kinect's images, Figures 90a and 90b are shown, both being the central view of the 3x3 matrix, being the cameras placed at exactly the same position. The difference is clear: the RayTrix camera captures a fraction of the image captured by the Kinect, but with much more detail.

(a) RayTrix (b) Kinect

Figure 90: Central view

Next, we will perform subjective assessment, visually. In order to do it, images that are 5 millimeters and 30 millimeters, to the left, will be compared, using RayTrix or Kinect depth maps. First it will be shown the whole image, then textureless flat areas, followed by edges and finally focusing on the unicorn, since it is the object with most details.

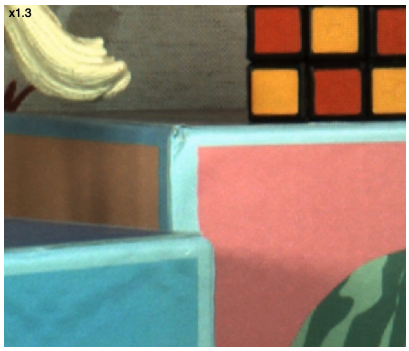
(a) RayTrix (b) Kinect

Figure 91: 5mm to the left

(a) RayTrix

(b) Kinect

Figure 92: 5mm to the left, unicorn



(a) RayTrix

(b) Kinect

Figure 93: 5mm to the left, edges and flat areas

With only 5mm distance, the RayTrix camera clearly comes on top. Despite having the missing information black band on the left, the rest of image is almost visually perfect. Edges are smooth and well defined, and it is able to capture and synthesize a lot of detail, as seen in the unicorn. On the other hand, Kinect, has some issues in the edges, having some small artifacts in some of them, while other are very sharp. In the case of the detail of the unicorn, quite a lot of detail is also kept, but in the fairy that is riding it there are some artifacts, probably due to the reprojection error because of the displacement between the RGB and depth sensors. Also, there are some artifacts in the white and blue cloth in which the objects of the scene are placed. These artifacts are due to the limitations of the depth sensor, since in that area the projected rays do not refract back to the sensor correctly.

(a) RayTrix

(b) Kinect

Figure 94: 30mm to the left

(a) RayTrix

(b) Kinect

Figure 95: 30mm to the left, unicorn

(a) RayTrix

(b) Kinect

Figure 96: 30mm to the left, edges and flat areas

In bigger distances, 3cm in this case, RayTrix performance decays quite a lot, but the Kinect maintains a similar performance. In the case of RayTrix, disocclusion artifacts start appearing next to the unicorn (black areas on the left). Also, the edges start blurring and are not straight anymore, similar to what happened with the multiview synthesis. In any case, most of the detail is kept, specially in the unicorn: its edges are a bit blurry, but one can clearly recognize it without major issues. In the case of the Kinect, edges have the same quality as before: some are very sharp and well defined, but other contain artifacts, but detail in areas with a lot of texture (the unicorn) is lost, generating major artifacts affecting to the shape and details.

In general, we can conclude that RayTrix plenoptic 2.0 cameras are more suited for capturing detail. On the ther hand, Kinect is more suited to capture wider angles at the cost of detail. Depending on one's needs and restrictions, one must choose one over the other. It must also be taken into account the cost of each device: the RayTrix camera costs several tens of thousand of euros, whereas one may acquire the Kinect with several hundreds.

Finally, it is worth noting that in this case, colour correction has not been performed. It is clear that the colours of the compared images captured by different devices are different, even though the scene and light conditions were the same during capture. In a future and deeper comparison, it should be corrected, using the colour correction tool proposed to the MPEG-I standard [47,48]. It would be specially necessary in the case of using multiple cameras, and it would also help in the subjective comparison, since the objects would have the exact same colours.

5 Conclusions and Future Works

5.1 Summary and Conclusion

During the elaboration of this masther thesis, assessment of several approaches for generating depth maps for view synthesis has been performed. A scene has been captured using a RayTrix plenoptic 2.0 camera, generating a dataset of 67 useful images for view synthesis, 9 of them ordered in a 3x3 matrix, separated by 3 centimeters each, and the rest ordered in a straight line, distanced from each by 1 millimeters. Then, depth maps have been generated using three different approaches: *DERS*, *Colmap* and RayTrix. With those three possibilities, view synthesis with *RVS* of the 1 millimeter-separated images has been performed using one view as reference. Also, a little experiment using multiple reference views has been performed. Finally, a comparison between two depth-sensing devices, Azure Kinect and RayTrix camera, has been done.

After completing the experiment, we observed that the RayTrix plenoptic 2.0 camera is able to generate depth maps of high-enough quality for DIBR real-time applications, since RayTrix is capable of generating those depth maps at a maximum rate of 30 frames per second. Regarding *Colmap*, we can asseverate it is also a great tool that is available to anybody since it's open source, but it has several limitations when generating depth maps, specially when it is encountered with textureless surfaces. Those depth maps are sharp feature detection and depth assessment, but are not completely filled, so they would need further processing in order to achieve quality similar to one achieved with *DERS* or RayTrix depth maps. Finally, as expected of a standarization tool, *DERS* is yielding the best quality depth maps of the three.

In the case of real applications, we observed that it is very important to perform correctly camera calibration when using RayTrix as a depth sensing device. If not performed correctly and the depth maps scales properly, the synthesized images will suffer from the ghosting artifacts, as shown in this work. On the other hand, *DERS* and *Colmap* perform better with simple camera calibration, since those depth maps have been generated using the same camera parameters, unlike RayTrix depth maps, that have their own scale.

With respect to the comparison of the RayTrix camera with the Azure Kinect, we came up to the conclusion that RayTrix performs better in capturing detail at the cost of a smaller field of view, thus the captured are more focused on the main scene. On the other hand, Kinect looses a bit of detail and sharpness in objects, but it is better suited to capture bigger environments or scenes. It is also important to highlight the difference in monetary cost of each device: Kinect costs several hundreds euros, whereas the RayTrix camera is several tens of thousands euros, being a difference in price in the order of 100.

Finally, as an academic conclusion, the elaboration of this work has allowed me to further increase my knowledge in the view synthesis and DIBR subjects, giving me experience on the field. Having the possibility of making use of a commercial plenoptic 2.0 camera is specially valuable, since those are such specialized devices not easy to find. Also, this work has allowed me to put to good use the knowledge acquired last year during the elaboration of my MA1 project, simulation of a plenoptic camera by means of a normal camera, since the tools used in both projects are the same, as well as the knowledge I acquired was useful for doing this master thesis.

5.2 Future Research Opportunities

Further research will be needed, repeating the same experiment with different scenes to address if the results are consistent whatever the captured scene is. On the other hand, the same experiment can be repeated using several reference views instead of only one when synthesizing virtual views. That approach will be closer to real applications, but more precise camera calibration will be required in the case of using depth maps exported from *RxLive*. Performing precise camera calibration with a plenoptic 2.0 camera is much more difficult than doing so with a regular camera because of the unique characteristics of this kind of camera.

The experiment can also be reproduced using multiple cameras to acquire multiview content at the same. In order to perform it correctly, it would also require colour calibration in order to ensure every image has the exact same colours. This colour calibration can be performed when doing another comparison with the Azure Kinect too, in order to ensure all the objects in the scene have the exact same colour, which would help with visual assessment, making the comparison as fair as possible, taking into account the limitations of each capturing device and their inner characteristics.

Different models of plenoptic cameras, including non-RayTrix devices, also give rise to new experimentation and assessment. Comparisons between different types of plenoptic cameras could be made, and even with other depth sensing devices such as the Azure Kinect or Intel Realsense LiDAR L515 [49], even though this last one has been discontinued.

6 References

[1] Tanimoto, M., Teratani, M., Fujii, T., Yendo, T. (2012). FTV for 3-D Spatial Communication. Proceedings of the IEEE, 100(4), 905-917. https://doi.org/10.1109/JPROC.2011.2182101.

[2] Tanimoto, M., Teratani, M., Fujii, T., Yendo, T. (2011). Free-Viewpoint TV. IEEE signal processing magazine, 28(1), 67-76. https://doi.org/10.1109/MSP.2010.939077.

[3] RayTrix. https://raytrix.de. Last visit: 29-12-2022.

[4] Mat Levoy, Teratani, Pat Hanrahan. (1996). Light field rendering. SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 31-42. https://doi.org/10.1145/237170.237199.

[5] Microsoft Azure Kinect DK. https://azure.microsoft.com/es-es/services/kinect-dk/. Last visit: 8-1-2023.

[6] Lafruit, G., Teratani, M. Virtual Reality and Light Field Immersive Video technologies for Real-World applications: IET, the Institute of Engineering and Technology. 2022. Chapter 4. ISBN: 978-178561578.

[7] COLMAP. https://colmap.github.io/. Last visit: 8-1-2023.

[8] Johannes Lutz Schönberger, Jan-Michael Frahm. Structure-from-Motion Revisited. 2016. Conference on Computer Vision and Pattern Recognition (CVPR). http://dx.doi.org/10.1109/CVPR.2016.445.

[9] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. 2016. European Conference on Computer Vision (ECCV). http://dx.doi.org/10.1007/978-3-319-46487-9_31.

[10] OpenCV. https://opencv.org. Last visit: 28-12-2022.

[11] Edward H. Adelson, John Y. A. Wang. Wang. Single lens stereo with a plenoptic camera. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, num. 2, February 1992, p. 99-106. DOI.org (Crossref), https://doi.org/10.1109/34.121783.

[12] Ren Ng, Marc Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz, Pat Hanrahan. Light Field Photography with a Hand-held Plenoptic Camera. Stanford Tech Report CTSR 2005-02, February 2005.

[13] Todor Georgiev, Andrew Lumsdaine. Focused Plenoptic Camera and Rendering. Journal of Electronic Imaging, vol. 19, num. 2, April 2010, p. 021106. DOI.org (Crossref), https://doi.org/10.1117/1.3442712.

[14] Shu Fujita, Sho Mikawa, Mehrdad Teratani, Keita Takahashi, Toshiaki Fujii. Extracting multi-view images from multi-focused plenoptic camera. International Forum on Medical Imaging in Asia 2019, editat per Hiroshi Fujita et al., March 2019, p. 24. DOI.org (Crossref), https://doi.org/10.1117/12.2521355.

[15] Mehrdad Teratani, Shu Fujita, Wenzhe Ouyang, Keita Takahashi, Toshiaki Fujii. 3D Imaging System using Multi-focus Plenoptic Camera and Tensor Display. 2018 International Conference on 3D Immersion (IC3D), December 2018, p. 1-7. DOI.org (Crossref), https://doi.org/10.1109/IC3D.2018.8657863.

[16] Daniele Bonatto, Sarah Fachada, Takanori Senoh, Jiang Guotai, Xin Jin, Gauthier Lafruit, Mehrdad Teratani. Multiview from micro-lens image of multi-focused plenoptic camera. 2021 International Conference on 3D Immersion (IC3D), January 2021. https://doi.org/10.1109/IC3D53758.2021.9687243.

[17] RLC software. https://gitlab.com/mpeg-dense-light-field/rlc. Last visit: 28-12-2022.

[18] Sarah Fachada, Armand Losfeld, Takanori Senoh, Gauthier Lafruit, Mehrdad Teratani. A Calibration Method for Subaperture Views of Plenoptic 2.0 Camera Arrays. 2021 IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP), October 2021, p. 1-6. DOI.org (Crossref), https://doi.org/10.1109/MMSP53017.2021.9733556.

[19] Fachada, Sarah, Armand Losfeld, Tankanori Senoh, Danielle Bonatto, Gauthier Lafruit, Mehrdad Teratani. [LVC] [DLF] A new calibration tool for multi-plenoptic 2.0 cameras. October 2021, https://doi.org/10.5281/zenodo.4488243.

[20] RPVC software. https://gitlab.com/mpeg-dense-light-field/RPVC. Last visit: 28-12-2022.

[21] Python. https://www.python.org. Last visit: 8-1-2023.

[22] Lafruit, G., Teratani, M. Virtual Reality and Light Field Immersive Video technologies for Real-World applications: IET, the Institute of Engineering and Technology. 2022. Chapter 11. ISBN: 978-178561578.

[23] Sarah Fachada, Yupeng Xie, Daniele Bonatto, Gauthier Lafruit, Mehrdad Teratani, "RabbitStamp Test Sequence", 2021.

[24] Sarah Fachada, Yupeng Xie, Daniele Bonatto, Gauthier Lafruit, Mehrdad Teratani, "[DLF] Plenoptic 2.0 Multiview Lenslet Dataset and Preliminary Experiments [m56429]", 2021.

[25] Sarah Fachada, Yupeng Xie, Daniele Bonatto, Gauthier Lafruit, Mehrdad Teratani, "[LVC] Update for RabbitStamp: Plenoptic 2.0 Multiview Lenslet Dataset [m57100]", 2021

[26] Sarah Fachada, Yupeng Xie, Daniele Bonatto, Gauthier Lafruit, Mehrdad Teratani, "[LVC] Exploration Experiments using RabbitStamp Multiview Lenslet Images [m57101]", 2021.

[27] Lafruit, G., Teratani, M. Virtual Reality and Light Field Immersive Video technologies for Real-World applications: IET, the Institute of Engineering and Technology. 2022. Chapter 12. ISBN: 978-178561578.

[28] Masayuki Tanimoto, Toshiaki Fujii, Kazuyoshi Suzuki. "Reference Software of Depth Estimation and View Synthesis for FTV/3DV" [M15836] ISO/IEC JTC1/SC29/WG11, October 2008.

[29] Masayuki Tanimoto, Toshiaki Fujii, Kazuyoshi Suzuki, Mehrdad Panahpour Tehrani, Menno Wildeboer. "Depth Estimation Reference Software (DERS) 3.0" [M16390] ISO/IEC JTC1/SC29/WG11, April 2009.

[30] Masayuki Tanimoto, Toshiaki Fujii, Mehrdad Panahpour Tehrani, Menno Wildeboer. ”Depth Estimation Reference Software (DERS) 4.0” [M16605] ISO/IEC JTC1/SC29/WG11, June 2009.

[31] Masayuki Tanimoto, Toshiaki Fujii, Mehrdad Panahpour Tehrani, Menno Wildeboer. ”Depth Estimation Reference Software (DERS) 5.0” [M16923] ISO/IEC JTC1/SC29/WG11, October 2009.

[32] Ségolène Rogge, Daniele Bonatto, Jaime Sancho, Rubén Salvador, Eduardo Juarez, Adrian Munteanu, Gauthier Lafruit. MPEG-I DEPTH ESTIMATION REFERENCE SOFTWARE. 2019. International Conference on 3D Immersion (IC3D). December 2019. https://doi.org/10.1109/IC3D48390.2019.8975995.

[33] Jaime Sancho, Takanori Senoh, Ségolène Rogge, Daniele Bonatto, Rubén Salvador, Eduardo Juárez, Adrian Munteanu, Gauthier Lafruit. ”RDE Fine-tuning to achieve DERS 8.0 performance” [M52135] ISO/IEC JTC1/SC29/WG11, January 2020.

[34] RxLive. https://raytrix.de/downloads/. Last visit: 7-1-2023.

[35] Tim Michels, Arne Petersen, Luca Palmieri, Reinhard Koch. SIMULATION OF PLENOPTIC CAMERAS. 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON). June 2018. https://doi.org/10.1109/3DTV.2018.8478432.

[36] Sarah Fachada, Daniele Bonatto, Arnaud Schenkel, Gauthier Lafruit. DEPTH IMAGE BASED VIEW SYNTHESIS WITH MULTIPLE REFERENCE VIEWS FOR VIRTUAL REALITY. 2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), June 2018, p. 1-4. DOI.org (Crossref), https://doi.org/10.1109/3DTV.2018.8478484.

[37] Daniele Bonatto, Sarah Fachada, Gauthier Lafruit. RaViS: Real-time accelerated View Synthesizer for immersive video 6DoF VR. Electronic Imaging, vol. 32, num. 13, January 2020, p. 382-1-382-89. DOI.org (Crossref), https://doi.org/10.2352/ISSN.2470-1173.2020.13.ERVR-382.

[38] Sarah Fachada, Daniele Bonatto, Mehrdad Teratani, Gauthier Lafruit. View Synthesis Tool for VR Immersive Video.. Chapter from book 3D Computer Graphics.

[39] RVS software. https://gitlab.com/mpeg-i-visual/rvs. Last visit: 28-12-2022.

[40] Razavi Khosroshahi, H., Sancho, J., Rosa, G., Salvador, R., Juarez, E., Lafruit, G., Teratani, M. (2023). ”Assessment of Multi-Plenoptic 2.0 Camera Depth Maps for DIBR”. In 2023 International Workshop on Advanced Image Technology (IWAIT) and International Forums on Medical Imaging in Asia (IFMIA), SPIE (9-11 January, 2023, Jeju, Korea).

[41] RayTrix 3D cameras. https://raytrix.de/products/. Last visit: 9-1-2023.

[42] OpenCV inpainting. https://docs.opencv.org/3.4/d7/d8b/group__photo__inpaint.html #gga8c5f15883bd34d2537cb56526df2b5d6a892824c38e258feb5e72f308a358d52e. Last visit: 10-1-2023.

[43] FFmpeg. https://ffmpeg.org. Last visit: 9-1-2023.

[44] A. Dziembowski, D. Mieloch, J. Stankowski, A. Grzelka. IV-PSNR – the objective quality metric for immersive video applications. June, 2022. IEEE Transactions on Circuits and Systems for Video Technology. https://doi.org/10.1109/TCSVT.2022.3179575.

[45] IV-PSNR software. https://gitlab.com/mpeg-i-visual/ivpsnr/-/tree/master. Last visit: 9-1-2023.

[46] Dorian Hoet, Gianluca Bontempi, Gauthier Lafruit. “Evaluation of the Azure Kinect depth sensor for view synthesis”. MA thesis. 2023.

[47] Teratani, M., Ishikawa, A., Sakazawa, S., Koike, A. (2010). ”Iterative colour correction of multicamera systems using corresponding feature points”. Journal of visual communication and image representation, 21(5-6), 377-391. https://doi.org/10.1016/j.jvcir.2010.03.007.

[48] Takanori Senoh, Nobuji Tetsutani, Hiroshi Yasuda. ”[MPEG-I Visual] Proposal of Trimming and Color Matching of Multi-View Sequences” [M47170] ISO/IEC JTC1/SC29/WG11, March 2019.

[49] Intel Realsense LiDAR L515. https://www.intel.es/content/www/es/es/products/sku/201775/intel-realsense-lidar-camera-l515/specifications.html. Last visit: 8-1-2023.

[50] YUView. https://github.com/IENT/YUView. Last visit: 8-1-2023.

[51] CMake. https://cmake.org. Last visit: 28-12-2022.

Appendices

Here, tools and software that are essential to reproduce the results of thesis are listed and their usage is explained. Some words on installation for each tool are in order, as well as giving some insight in their functionalities. In the case of *RxLive*, some basic functionalities are explained, whereas for *RLC*, *DERS* and *RVS* the most important parameters for configuration are described. In the case of *Colmap*, it is detailed, step by step, how to obtain camera parameters and depth maps. Finally, there are some words and insight on camera calibration using two different approaches: *Colmap* and *OpenCV*, as well as for calibrating the RayTrix camera using *RxLive*.

A YUView

YUView [50] is an open-source cross-platform tool used for analysis and visualization of *YUV* files. The *YUV* format is simply a format for storing colour images. Since it is not a common format used by the average user, a regular PC cannot open it with the default tools, so it needs special treatment.

It has been used to visualize *YUV* images, as well as to convert them into *PNG* files in order to facilitate handling and visualization.

It can be downloaded from this repository.

B Installation and operation

In this appendix, installation and operation of several tools used in the elaboration of this thesis will be presented. It could be considered as a basic step by step guide, along with some advice, to operate correctly the different software, always in conjunction with the rest, as if it was a pipeline.

B.1 RxLive

RxLive can be downloaded from the RayTrix website. Note that it will need a license, that will come with the purchase of a RayTrix camera, in order for the user to exploit all the software's functionalities, including the most basic ones such as exporting images. That license will be in a USB dongle, so it must be plugged in before starting the program for proper detection. In this work, *RxLive 5.0* has been used.

Installation is really simple, one must simply follow the installation wizard and the program will be ready.

Using it is also simple and quite user-friendly. Several tutorials can be found on their website. Here only the basic functionality will be explained.

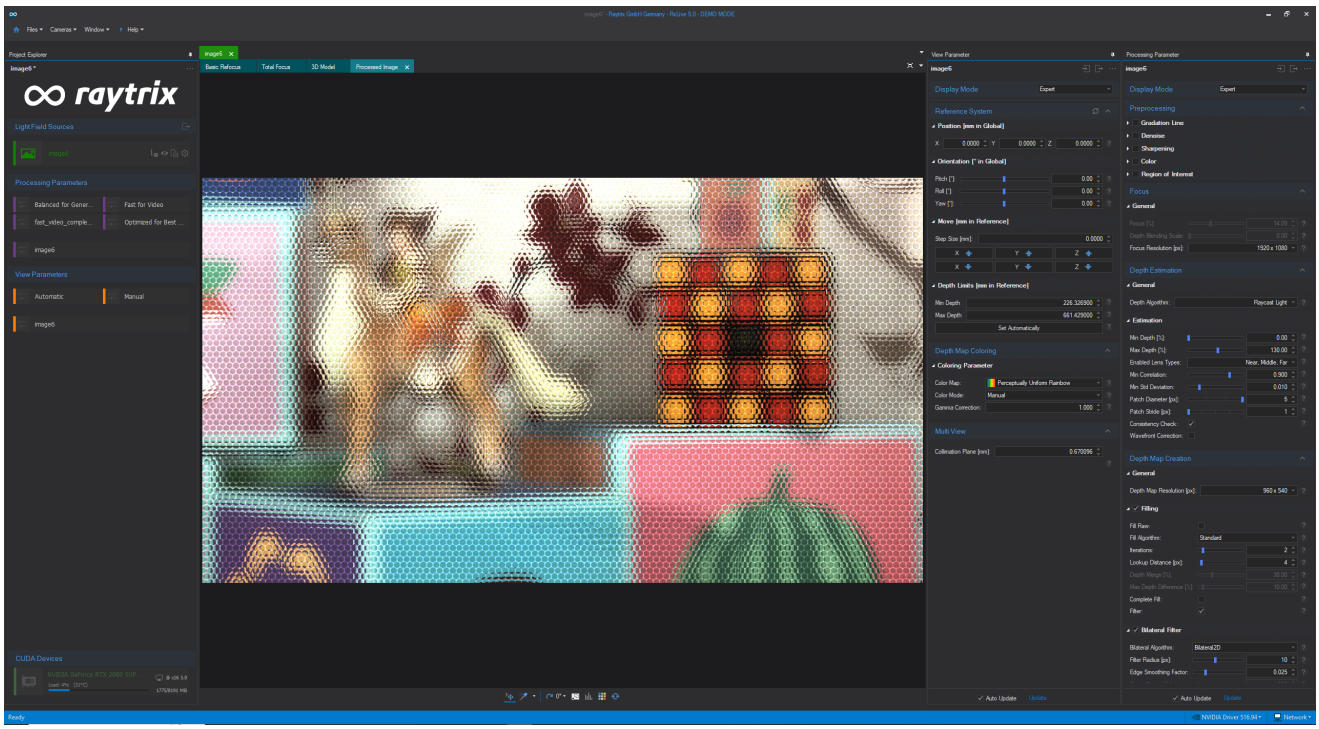


Figure 97: RxLive user interface

To proceed to capture date, the camera must be connected and calibrated. Then, the user must choose the camera to visualize the scene. The camera can be chosen on the bottom left corner, after *RxLive* detects it. Then, the program will move to the main menu.

In the main menu, the scene will be viewed in the center. One can choose several visualization options (lenslet, total focus, basic refocus, 3D rendering, coloured depth map...) by clicking on the eye button the selecting what is needed. Using the red camera button, one can capture an image, and with the red circle button a video. The shutter speed can also be modified, to let more or less light in the camera. The type of file and format to be exported can be chosen from the saving button, next to the eye button. The name of the file and the path to where it will be saved can also be specified.

Below the area of where one can name, capture and export the data, the are several presets. These are default configuration files that give certain values to the processing and visualization parameters. The user can create their own presets.

On the right side of the screen, there are the processing and visualization parameters. In the processing tab, there are parameters to change the focus, adjust depth and choose the algorithm for depth estimation. These last two have a big impact on the depth maps generated, so they must be manipulated carefully to obtain good results.

In the view tab, one can choose the colouring palette for depth maps, change brightness and gamma of images, and the maximum and minimum values of depth. These values can be adjusted automatically, and are very important when using depth maps, since these values are the ones used to adjust its scale.

The program can also export *.ray* files. These are light field files, that contain all the information of the image/s that has been captured. These files can be loaded into *RxLive* to visualize the captured data, process it using the processing and visualization tabs, and export it.

B.2 RLC

To install it, one must simply download the files from the repository and use the *CMake* [51] program to compile and generate the executable program.

To make use of *RLC*, one needs a lenslet image, the intrinsic camera parameters and the configuration file for *RLC* to work. The configuration file contains several parameters whose explanation and options can be found in the repository (along with an example file).

```
viewNum             5
rmode               1
pmode               0
mmode               2
lmode               1
Calibration_xml     .././TestDataset/R5_fujita/CalibData.xml
RawImage_Path       .././TestDataset/R5_fujita/img%03d.png
Output_Path         .././Output/fujita/Res_%03d
Debayer_mode        0
Isfiltering         0
isCLAHE             0
Gamma               1.0
Lambda              0.05
Sigma               0
input_model         0
output_model        0
start_frame         1
end_frame           2
height              2048
width               2048
```

Figure 98: RLC configuration file

In this appendix, only the most important will be discussed (the rest can be left with default values, or the same ones as in the example file):

- *viewNum*. It refers to the number of views it will generate. It can be either 5 (to generate a 5x5 matrix) or 7 (to generate a 7x7 matrix).
- *Calibration_xml*. It is the path to the intrinsic camera parameters, in *xml* format. One can check the example files to how it is formatted.
- *RawImage_Path*. Path to the lenslet image.
- *Output_Path*. Path to where the output must will be left.
- *start_frame*. Starting frame, interesting for video content.
- *end_frame*. Ending frame, interesting for video content.
- *height*. Height of the image, in pixels.
- *width*. Width of the image, in pixels.

To execute it, one can use the following command line: *RLC parameter.cfg*, where *parameter.cfg* is the configuration file. Note that *RLC* must be added to be path to call it using its name. Otherwise, one must execute the *converting* executable file (which is in the folder generated by CMake during installation) with *parameter.cfg* as argument.

B.3 DERS

To install it, use the *CMake* program to compile and generate the executable program. Note that this software is not available to the public as of the date this thesis is being written.

DERS only needs the reference views, including the view one want to generate its depth map, the camera parameters file and a configuration file in *JSON* format. In Appendix B.5 one can find how to obtain the camera parameters from *Colmap* in the adequate format for use in *DERS*.

DERS_config_image5.json

```
{
    "Version": "2.0",
    "InputCameraNames": [
        "image1",
        "image2",
        "image3",
        "image4",
        "image6",
        "image7",
        "image8",
        "image9",
        "image5"
    ],
    "InputCameraParameterFile": "final_camera_parameters.json",
    "OutputFiles": [
        "image5_1920x1080_grey16le_w9.yuv"
    ],
    "Start_frame": 0,
    "Fps": 1,
    "NumberOfFrames": 1,
    "DepthLength" : 16,
    "SearchRangeType": 1,
    "MinimumValueOfDisparitySearchRange": 60,
    "MaximumValueOfDisparitySearchRange": 120,
    "MinimumValueOfDisparityRange": 60,
    "MaximumValueOfDisparityRange": 120,
    "NearestDepthValue": 9,
    "FarthestDepthValue": 16,
    "NearestSearchDepthValue": 9,
    "FarthestSearchDepthValue": 16,
    "NumberOfDepthSteps": 1000,
    "BaselineBasis": 2,
    "Precision": 1,
    "VerticalPrecision": 1,
    "SearchLevel": 1,
    "ReliabilityThreshold": 1,
    "SmoothingThreshold": 48,
    "SmoothingCoefficient": 1.0,
    "SmoothingCoefficient2": 0.1,
    "Threshold": 5.0,

    "ViewImageNames": [
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image1_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image2_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image3_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image4_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image6_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image7_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image8_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image9_1920x1080_420p.yuv",
        "D:\Vicent_master_thesis\Raytrix_final_pictures\YUV\image5_1920x1080_420p.yuv"
    ],
    "VirtualCameraNames": [
        "image5"
    ],
    "VirtualCameraParameterFile": "final_camera_parameters.json"
}
```

Figure 99: DERS configuration file

The important parameters of the configuration file are the following:

- *InputCameraParameterFile.* Path to the file where the camera parameters of the reference views are.

- *VirtualCameraParameterFile*. Path to the file where the camera parameters of the views to be synthesized are. It can be the same file as in *InputCameraParameterFile*.
- *InputCameraNames*. Names given in the camera parameters file to the reference views. The view whose depth map is going to be generated must be listed here, and must be the last one.
- *VirtualCameraNames*. Name given in the camera parameters file to the view whose depth map must be generated. It can only generate one depth map at a time.
- *ViewImageNames*. Path to where the reference image files are. Admits *YUV* format. The view whose depth map is going to be generated must be listed here, and must be the last one.
- *OutputFiles*. Path to the output file, including the name of the file. Admits *YUV* format.
- *Start_frame*. Starting frame, interesting for video content. First frame is 0.
- *Fps*. Number of frames per second, for video content.
- *NumberOfFrames*. Total number of frames, for video content.
- *NumberOfDepthSteps*. Number of iterations to perform. 1000 usually yields very good results.
- *NearestDepthValue*. Maximum possible depth value to be given to the generated depth map.
- *FarthestDepthValue*. Minimum possible depth value to be given to the generated depth map.
- *NearestSearchDepthValue*. Maximum possible depth value to search. Normally has the same value as *NearestDepthValue*.
- *FarthestSearchDepthValue*. Maximum possible depth value to search. Normally has the same value as *FarthestDepthValue*.
- *MinimumValueOfDisparityRange*. Maximum possible disparity value to be given to the generated depth map.
- *MaximumValueOfDisparityRange*. Minimum possible disparity value to be given to the generated depth map.
- *MinimumValueOfDisparitySearchRange*. Maximum possible disparity value to search. Normally has the same value as *MinimumValueOfDisparityRange*.
- *MaximumValueOfDisparitySearchRange*. Maximum possible disparity value to search. Normally has the same value as *MaximumValueOfDisparityRange*.

Once it has been installed, one can use *DERS* by issuing the following command: *DERS parameter.json*, where *parameter.json* is the configuration file. Note that *DERS* must be added to be path to call it using its name.

B.4 RVS

In order to install and compile, one must download the files from the repository and use the program *CMake* to generate the executable file.

To use *RVS*, one needs the reference views to use and their respective depth maps, as well as the camera parameters of both the virtual view to be synthesized and the reference views. In Appendix B.5 one can find how to obtain the camera parameters from *Colmap* in the adequate format for use in *RVS*. It also needs a configuration file in *JSON* format.

view_synthesis_image5.json

```
{
    "Version": "2.0",
    "InputCameraParameterFile": "final_camera_parameters.json",
    "VirtualCameraParameterFile": "final_camera_parameters.json",
    "InputCameraNames": [
        "image1",
    "image3",
    "image7",
    "image9"
    ],
    "VirtualCameraNames": [
    "image5"
    ],
    "ViewImageNames": [
        "image1_1920x1080_420p.yuv",
    "image3_1920x1080_420p.yuv",
    "image7_1920x1080_420p.yuv",
    "image9_1920x1080_420p.yuv"
    ],
    "DepthMapNames": [
        "image1_1920x1080_grey16le_w9.yuv",
    "image3_1920x1080_grey16le_w9.yuv",
    "image7_1920x1080_grey16le_w9.yuv",
    "image9_1920x1080_grey16le_w9.yuv"
    ],
    "OutputFiles": [
    "out_image5_1920x1080_420p.yuv"
    ],
    "StartFrame": 0,
    "NumberOfFrames": 1,
    "Precision": 1.0,
    "ColorSpace": "RGB",
    "ViewSynthesisMethod": "Triangles",
    "BlendingMethod": "Simple",
    "BlendingFactor": 5.0,
  "QualityType": "TriangleSide"
}
```

Figure 100: RVS configuration file

The parameters of that configuration file will be discussed:

- *InputCameraParameterFile*. Path to the file where the camera parameters of the reference views are.
- *VirtualCameraParameterFile*. Path to the file where the camera parameters of the views to be synthesized are. It can be the same file as in *InputCameraParameterFile*.
- *InputCameraNames*. Names given in the camera parameters file to the reference views.
- *VirtualCameraNames*. Names given in the camera parameters file to the views to be synthesized. It can synthesize more than one view at a time.

- *ViewImageNames*. Path to where the reference image files are. Admits *YUV* format.
- *DepthMapNames*. Path to where the depth maps files of the reference image are. Admits *YUV* format.
- *OutputFiles*. Path to the output file, including the name of the file. Admits *YUV* format.

To execute it, one can use the following command line: *RVS parameter.json*, where *parameter.json* is the configuration file. Note that *RVS* must be added to be path to call it using its name.

B.5 Colmap

Colmap can be downloaded and installed from its documentation website. It offers support for Windows, Mac and Linux. In the case of Linux, CUDA support must be manually installed.

To start the program, one must launch the *.bat* file on Windows, or run the application on Mac and Linux.

In order to export camera parameters:

1. Create a new project (File →New project).
2. Create a new database and give it a name.
3. Choose the folder where the images are (they must be encoded with less than 64 bits, otherwise *Colmap* won't be able to read them).
4. Perform feature extraction (Processing →Feature extraction, or button with the half colored and half grey image). Change the camera model to SIMPLE_PINHOLE and then click "Extract".
5. Perform feature matching (Processing →Feature matching, or button with the black square grid). Leave the default settings and then click on "Run".
6. Start reconstruction (Reconstruction →Start reconstruction, or button with blue "Play" arrow). One can reset the current reconstruction by clicking on Reconstruction →Reset reconstruction. That will allow to create a new reconstruction.
7. In the case any step fails, note that *Colmap* uses more than 30 images (normally) to work well. Another solution could be to make the parameters less restrictive (Reconstruction →Reconstruction options, or button with grey building with a small pencil).
8. When the reconstruction finishes correctly, export the parameters (File →Export model as text). This will create three files: *images.txt*, *cameras.txt* and *points3D.txt*.
9. Use script *colmap_to_json.py* to obtain the *JSON* file with the parameters ready to use (*python colmap_to_json.py -c path_to_cameras.txt -I path_to_images.txt -o outputFile.json*). The script can be found on the *RPVC* GitLab repository. Note that the script may need some modification such as changing the depth map colour space to *YUV400* or changing the depth range.

Example of the obtained camera parameters after adapting them using the *RPVC* script:

final_camera_parameters.json


```
{
    "Version": "2.0",
    "Content_name": "Colmap dataset",
    "Fps": 1,
    "Frames_number": 1,
    "Informative": {
        "Converted_by": "colmap_to_JSON.py",
        "Original_units": "m",
        "New_units": "m"
    },
    "cameras": [
        {
            "Name": "image1",
            "Position": [
                -1.1745805324306768,
                0.37473708659862254,
                -1.9459889256011818
            ],
            "Rotation": [
                0.7313370552792698,
                -0.5773175881953206,
                0.06127881289893649
            ],
            "Depthmap": 1,
            "Background": 0,
            "Depth_range": [
                9,
                16
            ],
            "Resolution": [
                1920.0,
                1080.0
            ],
            "Projection": "Perspective",
            "Focal": [
                5348.45,
                5348.45
            ],
            "Principle_point": [
                960.0,
                540.0
            ],
            "BitDepthColor": 8,
            "BitDepthDepth": 16,
            "ColorSpace": "YUV420",
            "DepthColorSpace": "YUV400"
        },
        {
            "Name": "image2",
            "Position": [
                -1.2245842240531852,
                -0.03953757957019628,
                -1.9418404126671756
            ],
            "Rotation": [
                0.6151230695144356,
                -0.5615275418565995,
                0.0263444650661808
```


Figure 101: Camera parameters

To obtain depth maps:

1. Finish the reconstruction (same steps 1-6 as when extracting camera parameters).
2. Perform a dense reconstruction (Reconstruction →Dense reconstruction, or square button with the black-grey gradient).
3. Choose a folder (preferably empty) for the data to be stored.
4. Click on button “Undistortion” and wait for it to finish.
5. Click on button “Stereo” and wait for it to finish.
6. On the grid, all images with their respective depth maps can be seen (click on the buttons for the depth maps to see them, with the possibility of saving). The files are also stored as binary files in the folder previously selected. If the resulting depth maps present holes in them (black areas), one can partially solve the problem by repeating the process in a new empty folder, but in the “Options button” increase the field “window_radius” (max value is 20 in *Colmap 3.6*). It can also help reducing the field “filter_min_ncc” or increasing the “max_image_size”, but the “window_radius” field has the biggest impact. Note that this will reduce sharpness in edges.

7. Use the script called *dense_to_exr.py* (*python dense_to_exr.py -d path_to_binary_depth_maps –t geometric/photometric*) in the *RPVC* GitLab repository to transform the binary depth maps to *EXR* depth maps that can be used directly with *RVS*. These depth maps may need some preprocessing, since it is common that they have no data in certain areas which have a plain basic texture (like a simple colour).

8. To inpaint the depth map, one can use the script *inpaint_colmap_depth.py*, modifying it so that it inpaints the depth maps the user wants. Normally the best result is obtained inpainting the depth map obtained with window_radius=20 using the *OpenCV* inpainting algorithm "TELEA".

C Camera calibration

Camera calibration is basic when performing view synthesis using multiple reference to generate virtual views. There are several ways to perform camera calibration. In this Appendix, only *Colmap* and *OpenCV* will be discussed. These approaches allow to obtain both the intrinsic and extrinsic parameters.

On the other hand, the RayTrix plenoptic camera must also be calibrated. This is a different kind of calibration, since it is used so that the RayTrix uses the right units to estimate distances and depth.

C.1 Using Colmap

Colmap requires at least 30 images to perform camera calibration correctly. Those images must capture the same scene, but must not be too similar among them, otherwise *Colmap* will reject them.

The software is able to perform this calibration without the need of a known pattern, it will do it using 3D reconstruction and feature matching. The parameters will be "up to scale". That is, the unit of the parameters is not known, although they might coincide with real units if studied properly, but they will vary from one dataset to another.

In order to perform the calibration, the images must be inputted to *Colmap*. Then start feature extraction, followed by feature matching and finally perform the 3D reconstruction. Regarding the intrinsic parameters, they will depend on the chosen camera model. *Colmap* offers simplified versions of the camera models implemented in *OpenCV*. One must choose accordingly depending on oneselves needs and knowledge. Once the 3D reconstruction has been finished, one can export the model (the camera parameters) in several formats, including text. Then the user must adapt them to the adequate format if they are planned to be used in another application. Further information can be found on the official documentation, and on Appendix B.5.

C.2 Using OpenCV

OpenCV also is able to perform camera calibration, in order to obtain intrinsic and extrinsic camera parameters. In this case, it will use a known patter, such as a chess or charuco board.

It can obtain intrinsic and extrinsic parameters separately. In the case of intrinsic parameters, it requires around 30 different images of the pattern. Those images must be of high quality so that *OpenCV* is able to detect the features correctly and assess the distances. The images must also be

different. To achieve that, the user must rotate the board and take pictures of it in several different positions, making sure the pattern is always visible. Once 30 or more images have been captured, they can be given to *OpenCV* to perform intrinsic calibration. It will output the intrinsic parameters matrix with the principle point and the focal distance, as well as the matrix with the distortion coefficients. These intrinsic parameters can be reused as many times as one wants, as long as the camera that is going to be used is the same one. In the case of RayTrix, the RayTrix own calibration must no be changed (see Appendix C.3).

For the extrinsic parameters, one must simple capture the views of which the user wants to obtain the position and rotation of the camera. Then, using the intrinsic parameters, which can be obtained using *OpenCV* or could be known by the camera specifications, must be used. After that, *OpenCV* outputs the extrinsic parameters matrix, with the translation and rotation of the camera.

C.3 RayTrix calibration using RxLive

Before using any RayTrix camera, it first must be calibrated with *RxLive*. The program offers a calibration wizard to help the user with the calibration of the camera. This calibration also makes use of a pattern, being, in general, a dot pattern, where the diameter of the dots and the space between them must be known.

The minimal calibration required for a RayTrix to work properly is the calibration of the micro-lens array (MLA) and the metric calibration).

For the MLA calibration using the wizard, the user must first open the MLA calibration wizard, then follow the instructions: first, one must choose a main lens, if it was already registered, or create a new one. Next, choose an existing configuration to modify it, or create a new one. Then, the user must input the focal lens. With prime lenses, minimum and maximum values are the same. In the case of zoom lenses, they correspond to the maximum and minimum values of the zoom. Then after giving the focus distance, the calibration filter must be put. Next, a grey image must be captured. This grey image (literally an image of grer or white colour) will help to adjust the light intensity and the shutter time, as well as the MLA array. For light intensity and shutter time, the wizard will show a RGB histogram. It should at least reach 90% illumination. After that the main lens should be adjusted to fit the aperture of the MLA. Micro-lenses images should be touching, without overlapping. With that, the MLA calibration is complete. If the aperture or the main lens are changed, the process must be repeated.

Regarding the metric calibration, a wizard is also available. In this case, the dot pattern will be used. The first step is to input the point pitch. In order to avoid parallax errors, one can measure the pitch of 10 points and divide by 10. Then, the calibration target must be put in front of the camera, tilted, no more than 45 degrees. An image of the pattern is captured, and then one must check if the overlay correctly matches the black dots. If so, they image can be kept, otherwise, it has to be discarded. This process must be repeated several times, having the dot pattern in different positions. The capturing can be stopped when there are point the three depth zones (green, blue and red) on the histogram. After that, it can be switched to 3D mode to check if the grey points as overlapping with the coloured points. If so, the user can choose a calibration level (3 by default) and calibrate. Then the wizard will show the results, rating them with stars (maximum is 5) and showing the deviation. The deviation must be as low as possible, being 0% the minimum. If the quality is not high enough, one can take more images, check if the point pitch distance is correct or repeat the whole procedure.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**				x
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**			x	
ODS 9. **Industria, innovación e infraestructuras.**	x			
ODS 10. **Reducción de las desigualdades**.				x
ODS 11. **Ciudades y comunidades sostenibles**.				x
ODS 12. **Producción y consumo responsables.**				x
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**			x	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En el caso de mi TFM, *Depth Image-Based Rendering for Multiview Plenoptic Camera*, no tiene mucha relación con los ODS, salvo con el ODS 9: Industria, innovación e infraestructuras. El TFM trata sobre la técnica Depth Image-Based Rendering para el renderizado de escenas sacadas del mundo real para aplicaciones de realidad virtual. La realidad virtual es una tecnología emergente que tiene mucha perspectiva de mejora y que se puede aplicar en muchos ámbitos: entretenimiento, robótica, comunicaciones, transporte, etc., por lo que puede ser una tecnología clave en un futuro no muy lejano en muchos ámbitos de nuestra vida, tanto cotidiana como profesional. Va especialmente ligada con la innovación, puesto que se pueden reinventar tecnologías y metodologías actuales para que se incluya la realidad virtual: operar máquinas de manera remota como si uno estuviera allí, o facilitar la inmersión en un entorno diferente de en el que físicamente se está son solo un par de ejemplos de posibles aplicaciones. Esto ofrece la posibilidad a nuevas empresas a hacerse un hueco en el mercado vendiendo nuevos productos innovadores relacionados con la realidad virtual. Por otro lado, el TFM también trata las cámaras plenópticas. Este tipo de cámaras hace ya años que se inventaron, pero actualmente tienen muy poco mercado y, por tanto, pocas empresas las fabrican y las venden. Este tipo de cámaras tienen muchas ventajas respecto a una cámara convencional, especialmente en aplicaciones en tiempo real, por lo que dar a conocer las bondades de este tipo de cámara es crucial para que en un futuro crezca su popularidad, propiciando la aparición de nuevos fabricantes y mejorando la competencia dentro de su propio sector. Esto también va relacionado, aunque en menor medida, con el ODS 8: Trabajo decente y crecimiento económico. El crecimiento económico está claro: la posibilidad de aparición de nuevas empresas en el sector de la realidad virtual, así como un posible aumento de la competencia dentro del sector de las cámaras plenópticas. En cuanto al trabajo decente, esto dependerá de la legislación vigente de cada país, pero lo que está claro es que estas nuevas empresas necesitarán contar con personal cualificado para el desarrollo de su actividad, y, siendo personal que no es fácil de encontrar (al menos hoy en día) deberían ofrecer unas condiciones de trabajo bastante buenas para mantenerlos en su fuerza de trabajo. Por último, el ODS 17: Alianzas para lograr objetivos, también tiene algo de relación, aunque no directamente con el TFM en sí. En primer lugar, destacar el acuerdo de doble titulación entre la Universitat Politècnica de València y la Université Libre de Bruxelles, que me ha permitido terminar mis estudios en Bruselas, realizando allí el TFM con algunos de los mayores expertos en el campo, que han desarrollado tecnologías de estandarización en el campo de la realidad virtual. En segundo lugar, el hecho de trabajar allí me ha permitido ver de primera mano la colaboración entre su grupo de investigación de realidad virtual con otras universidades, como la Universidad Politécnica de Madrid o la Universidad de Nagoya, en Japón, lo cual evidencia que, dentro del campo de la realidad virtual, es crucial la colaboración entre investigadores y expertos, independientemente de su lugar de procedencia o su lugar actual de trabajo.