PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

REAL TIME WATER REMOVAL FOR UNDERWATER PHOTOGRAMMETRY WITH DEPTH INFORMATION

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR:

Sebastian Jesus Torres Padilla

ASESOR:

Cesar Alberto Carranza De La Cruz

Lima, Octubre, 2024

Informe de Similitud

Yo, **Cesar Alberto Carranza de la Cruz**, docente de la Facultad de **Ciencias e Ingeniería** de la Pontificia Universidad Católica del Perú, asesor de la tesis titulada "**Real Time Water removal for underwater photogrammetry with depth information"**, del autor **Sebastián Jesús Torres Padilla**,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 12 %. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 11/08/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: Lima, 11 de agosto de 2024

Apellidos y nombres del asesor: Carranza de la Cruz, Cesar Alberto	
DNI: 09641576	Firma 
ORCID: 0000-0003-1222-0118	

Resumen

En la actualidad, la restauración de imágenes submarinas sigue siendo un tópico complejo en el ámbito de la imagenología óptica subacuática. Esto se debe a que los efectos atmosféricos y de reflectividad producidos por la escena suelen generar una degradación significativa a la fotografía y limitar el rango capturado debido a la borrosidad producida en el ambiente. Con la finalidad de solucionar este problema, la siguiente tesis implementa un algoritmo de retrodispersión que busca eliminar el efecto de degradación generado por el agua. Este proceso utilizara una imagen extraída de una cámara RGB y un mapa de profundidad extraído por software de fotogrametría como puntos de entrada.

Sin embargo, dada la complejidad computacional que involucra ejecutar, el tiempo de ejecución aumenta exponencialmente, especialmente para imágenes de alta resolución y matrices generados de mapas de profundidad. Debido a esto, es indispensable diseñar un algoritmo paralelo e implementado en una GPU para poder acelerar significativamente el tiempo de procesamiento.

Abstract

Nowadays, underwater image restoration remains as one complex issue in underwater optical imaging. The reason of this is because the atmospheric and reflective effects produced by the scene that can severely degrade the photograph and limits the captured range because of the blurriness produced in the environment. To solve this topic, this thesis makes an implementation of a backscatter algorithm that tries to remove the degradation effect produced by the water. This process will utilize a RGB photograph and a depth map extracted from a photogrammetry software as inputs.

However, given the significant computational complexity involved, the execution time increases considerably, especially for high resolution images and depth map matrices. For that reason, it is necessary to design a parallel algorithm and implement it in a GPU to be able to significantly accelerate the processing time.

Acknowledgement

The author of this thesis wishes to FONDECYT through its national program PROCIENCIA (161-2020- FONDECYT) for providing the data for the research and development of this project.

Este trabajo está especialmente dedicado a mis abuelos Alfredo y Rubén por ser de gran inspiración y motivación durante todo mi periodo académico.

A mis Padres Félix y Nidia por todo el apoyo brindado y ser los pilares fundamentales durante este periodo académico.

Contents:

List of Figures:

List of Tables:

1. Chapter 1: Motivation, State-of-the-art, Justification and objectives.

1.1. Motivation:

The reconstruction of underwater images using computer vision is an important subject in images processing because of its numerous applications, such as identification of underwater objects, rescue of sunken ships, constructions of underwater facilities like mining systems and ocean oil platforms, and an easier way to drone handling vehicles as the case of the remotely operated vehicles (ROVs) [8]. But there are some drawbacks of taking photographs in underwater environments like the degradation of the image, which remains as one challenging problem. One of the main reasons for this degradation is because of lights scattering, the dust presence and the objects that cause noise between the camera and the object due to the particle reflectance (figure 1.1 (a)). At the same time, there are other factors that degrade the photography resolution, like the depth of the object, because the wavelengths of the image are attenuated depending on the distance between the surface and the object, this effect is called direct transition component (DTC). Moreover, the DTC effect varies the number of visual light wavelengths (400 to 700 nm) according to the depth of the object such that the red color is more likely to be absorbed in the first 5 meters range (figure 1.1 (b)) than the green and blue color that are able to keep the light for around 60 meters range before becoming imperceptible or losing the 93.5% of color perception (figure 1.1 (b)) [8]. Thus, the image suffers from various types of distortions, like quality degradation, blur, foggy, hazing and light scattering and color distortion due to the light attenuation generated by the water depth [8].

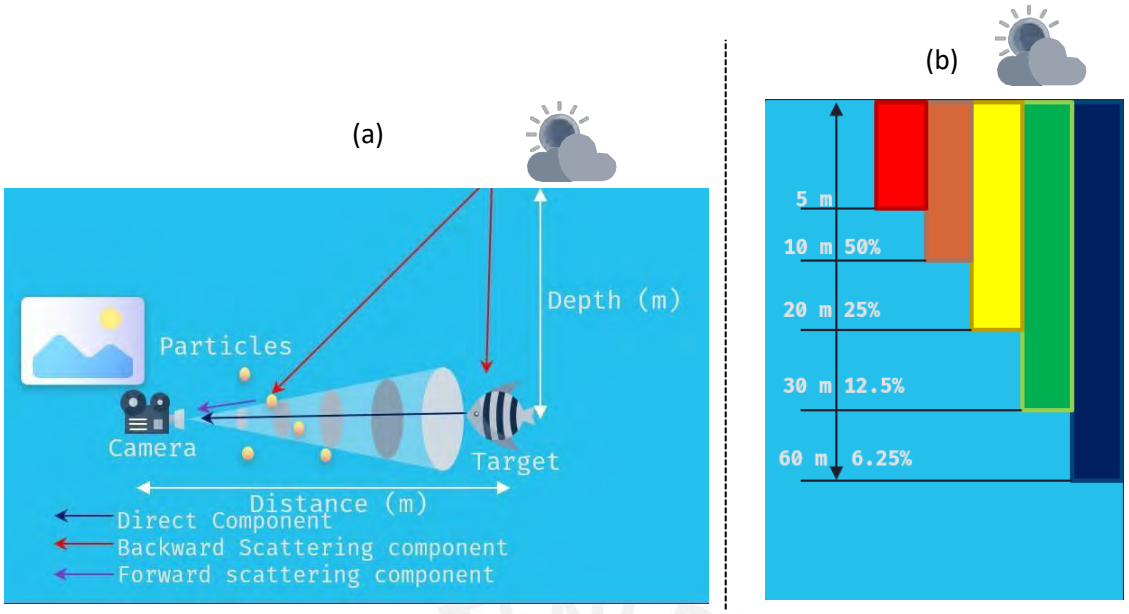


Figure 1.1: Underwater image. (a) light scattering. (b) light absorption in different rates [1]

Due to the limited resolution obtained in the underwater photographs, different techniques about image reconstruction have received a lot of attention. These technicities generally use different types of tools, such as the RGB cameras, which allows the capture and interpretation of colors, using three different channels red, green and blue. This device has the capacity of catching the light in the visible spectrum, between the object and the camera. The main problem of this equipment remains in the attenuation of the wavelength in the object due to the underwater depth. As a consequence, the image starts losing a percentage of colors depending on the range of the distance of the object to the surface (Figure 1.1 (b)). Another tool used in image restoration is the RGB-D camera, which can generate depth maps using the distance between the sensor and the object. The depth map data plays a crucial role in underwater image enhancement and water removal, due to the ability of extracting 3d information which can be used as an input to provide a statistical allocation of the space used by the picture to remodel the image object volume [9]. But this tool has some limitations such as the bad resolution due to the particle presence in the space and the bad quality image generated by some RGB-D cameras, that can't provide an accurate depth map to make a 3D reconstruction.

The image color recovery focusses its solution by improving its quality, increasing the resolution and reducing the attenuation in the image [10]. But the restrictions generated by the effects of the light and the water prevent the conventional RGB cameras from recovering the photography without noise and water, so many efforts have been made to find a solution to this problem. There have been a lot of research about image restoration, that attempts to provide a visually recovered image using the scene radiance of the light, which restores the image colors [8]. However, despite the researched methods an RGB camera can't enhance the image to its completely original colors. So, the research focuses its attention on removing the light scattering and the particles in such a way that the photograph looks closely to a water removal image.

On the other hand, the other tool used in water removal and image enhancement is the RGB-D camera, that uses time-of-flight (ToF) technology sensors that measure the round-trip time of the infracted signal and the object [11], that make it acquire the depth map information. These types of inputs use the depth information and the light information to reconstruct the 3D volume of depth probability, that is occasionally called the cost volume of stereo vision [9]. This principle allows the subject to reconstruct the 3d model of the photography using the depth map and the RGB camera image. However, the options to take an underwater depth transmission map are limited due to the depth sensors, such as the time-of-flight (ToF) or the stereo vision sensors, that are affected by the water depth which reduce the depth map resolution and affect the 3d reconstruction. So, there are different methods to increase the depth resolution using filters that take advantage of the image resolution to register high-quality information to provide important information about the photograph. Finally, this information is yielded to the water removal using the filtered depth map and the RGB picture. However, there are a few methodologies that utilizes the depth map and the photograph as inputs and are capable to work in a short

range of time. The next section will make an explanation of some existing methodologies for the water removal process.

1.2. State-of-the-art

The image reconstruction or the water removal by computer vision, allows the subject to use different kinds of techniques or algorithms, which has the main function of yielding the image to a better resolution in a water depth environment. Generally, the RGB-D camera lets the user have two input information, which are the Red-Green-Blue (RGB) sensors and the depth sensors. The RGB cameras send the color information with the attenuated scene radiance or the DTC [8], so the input yields a distorted underwater image. At the same time, the second input is the depth sensor, that lets the user make a 3d reconstruction of the scene. But, due to the limited resolution and the underwater environment particles, the sensor can't provide an accurate 3d map reconstruction. So, to keep the local depth structure the algorithms usually use guided filters to eliminate this deficiency [11] to have an accurate depth and RGB inputs for the water removal or water enhancement methods.

Huimin Lu and Yujien Li [3] found that the image enhancement methods are classified in 6 different groups. These classifications are divided by their distance estimation, their backscatter radiance of the scene, their different ways to enhance the image and their different ways to calculate the parameters to make a precise image enhancement. This different groups are the next ones:

> **a- Polarization. –** These techniques consist in restoring the image by taking different RGB images at different angles to estimate the distance data, so these parameters can be used to enhance the image using polarization filters attached to the RGB camera. In this category, Li Dekui [12] proposed in 2022

that the polarization information is affected by any kind of wavelength when the information estimates the transition to the target detection, causing some noisy outputs, as a result the image is distorted. So, the algorithm estimates the polarization state of light and it can track the parameters of the reflectance polarization information [12].

In 2021, Jiamin Qian and Jianxin Li [13], proposed a spectral polarization method that uses the degree of polarization and the angle of polarization to estimate the parameters that are used to recover the image and show enhanced visibility [13]. However, these methods can't provide a high attenuation to the transference radiance, especially in time variations, because this effect is averse to the brightness and visibility conditions [3].

b- Turbidity medium. – This type of method analyzes a scene using multiple images to find the parameter needed to enhance an image. In 1997, Fabio Cozman and Eric Krotkov [14] proposed a method that consists in analyzing a scene by taking numerous images from different angles, so it can estimate the distance to the object and the depth map. This data is used by an algorithm which restores the image by analyzing the static scene [14]. Nevertheless, it can't be applied in real time, because it needs multiple images to calculate itsparameters.

c- Multi-lighting. – This category consists in estimating the depth parameters inunderwater images using multiple artificial lights. In 2014, Chourmouzios Tsiotsios and Maria E. Angelopoulou [15] proposed the use of multiple artificial light sources to estimate the medium absorption of the image and the scattering components from the underwater scene [15]. The direct components are measured by the light beam traveling distance, which returns to the camera

sensor due to the reflection effect [15]. At the same time, the backscatter components help to predict the total backscatter brightness integrating the multiple depth positions obtained by the multiple light sources. However, to avoid light saturation and potential graduate errors, this multi-lighting technique [15] uses different light sources to take an accurate description of the scene depth. Nevertheless, the presence of particles in underwater environments prevents the sensor from taking a precise measurement of the transmission depth map, so it's more difficult to make an accurate dehazed image.

d- Scene depth. – This category focuses on calculating the depth map to solve the visibility ambivalence produced by the color distortion and the particle noise that causes images attenuation [3]. Chiang J. and Chen Y. proposed to find the information by estimating the distance of the object to the camera, soit can provide the scene depth map. Thus, the data can be used to make an artificial light removal to compensate for the artificial light presence by the surface underwater depth scenario [2]. However, estimating the depth information in underwater environments could provide erroneous information due to the particles, the water and the concentration of haze in the image scenario, which make difficult to estimate the distance between the object and the camera, so it could need another input to calculate the depth data [3] to make an accurate estimation.

e- Fusion. – This category focuses on restoring the image by combining the Laplacian contrast, white balance, and gamma correction to obtain the water removed image. Codruta O. Ancuti and Cristophe de Vleeschouwer [4] proposed that the image can be enhanced by using a white-balance section to

compensate the color loss and improving the image aspect due to the depth, which is parallelized in two section that try to compensate the brightness, which then are combined in a multiscale fusion section. Thus, the output is a completely restored image with its original color that were attenuated by the underwater depth [4]. However, this process provides a light saturation that affects the output images.

f- Prior. – This category works with different ways to estimate transmission algorithms over underwater images. This includes estimating the scene radiance and the transmission map using different dehazing algorithms like the Dark Channel Prior (DCP). In 2012, Martina Martin and Gunjan Pandey [16] proposed that the parameters can be initialized by making a transmission map estimation using DCP, which helps to make a distance estimation, so the algorithm makes a transmission range map assessment to computerize the image [16]. These parameters are used to evaluate the color computation so the algorithms are able to find the different shape attributes that yield to a dehazed and clear image [16]. In consequence, the output image colors are restored and dehazed. However, using DCP to estimate parameters could yield an erroneous approximation in the parameters, which affect the final result. So, using RGB-D cameras provide more accurate information to estimate the parameters and restore the photograph.

As it was mentioned in the previous paragraphs, most of the water enhancement categories use different estimation methods to find the distance data because the RGB images inputs can't provide a depth transmission map. In consequence, the enhanced image output has a low quality due to the inexact estimation of the distance between the camera and the object. However, the RGB-D cameras don't need to calculate the depth

estimation, because it can provide distance information without an algorithm, so it is easier to apply a water removal than using the RGB image.

The water removal or underwater enhancement using depth information uses two inputs. These inputs require two different kinds of algorithms, due to structural differences, that focus on compensating the light scattering or color change the distortion in color images [2] and filtering the noise in depth data to increase the depth output resolution. The next paragraphs will explain about some experimental methods that improve the quality of the estimated photograph and how the effectiveness of some of them work.

The Underwater Image Enhancement by Wavelength compensation and Dehazing method was proposed in 2012 by John Y. Chiang and Ying-Ching Chen, and combines algorithms for wavelength compensation and dehazing water (WCID), which consist in removing distortion caused by light scattering and color depth degradation [2]. This method combines WCID techniques to remove the distortion caused by de light reflection using depth derivation methods to approximate the distance between the object and the camera [2]. The light intensity, in the foreground and the background, is used to determine the artificial light source used in the process [2], if the artificial light is detected, the illumination produced by de light reflection is removed to avoid overcompensation of the light. After all this process, the program uses algorithms and wavelength compensation to remove the noise produced by the water haze and the particles between the object and the camera. In the original proposed method, the residual energy ratio of the RGB colors is used to make an approximation of the underwater depth map in the scene [2]. In the case of using an RGB-D camera, it's not necessary to use scene derivation methods to calculate the distance between the camera and the object, because in this case the camera provides two inputs, which can be used to generate the depth data, so the program focuses on the water removal algorithm instead of the depth estimation.

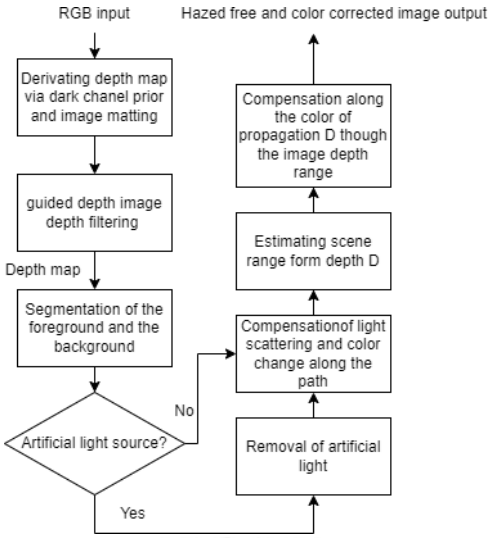


Figure 1.2: Flowchart of the Image Enhancement by Wavelength compensation and Dehazing method using depth information [2].

As it is shown in (Figure 1.2) the RGB-D system uses the depth data as an input, so the algorithm doesn't make a depth estimation using the Dark-channel-prior (DCP), an existing depth derivation method, to estimate the distance of the object to the camera [2]. The algorithm uses the depth map to segmentate the foreground and background areas of the image. And then, the program tries to find the artificial light source in the photo to avoid overcompensation, if some light source is detected, the program proceeds to remove it. After that the dehazing method removes the haze effect and color change due to the underwater transmmision of the camera [2]. Finally, the energy compensation for the three color channel (RGB) is carried out to restore the bluish and reconstruct the image to its natural colors.

Figure 1.3: Comparative results of the wavelength compensation and image dehazing method[2].

As a result, (Figure 1.3) shows that the wavelength compensation and dehazing method tries to recover the image by removing the artificial light from a turbid picture. This method uses the RGB image light as an input, so the technique directly tries to find the existence of a supplementary source of artificial light that has to be removed. The influence of the artificial light provided by the camera is a function of the amount of luminance given by the light and the surface reflectance [2]. However, the amount of luminance provided by light is inversely proportional to the distance range between the object and the camera. So, the method does a distribution of luminance between the three-color channels (Red, Green and Blue), removing the artificial light instead of removing the water, so this doesn't look so effective enhancing a picture or removing water from it.

The second method is called "Color Balance and Fusion of Underwater Images" [4], and it was based on a model called "The Contrast Enhancement for Images in Turbid Water" method, which was proposed by Huimin Lu in 2015 [3]. This technique divides the input image in three divisions, that estimate the distance of the object to the camera and use a filter refinement. The first section is called locally adaptive color correction (LACC) (Figure 1.4) [17], and it is constituted by an attenuated image section, an underwater attenuated image section, and a redefined color channels section [17]. This segment allows the algorithm to make a depth map estimation, and a local filter refinement, so the input image has a color clarification. In the case of using the RGB-D camera, the depth map estimation is already made (Figure 1.4), so instead of using a depth map estimation section, it could use a bilateral filter to increase the depth map resolution.

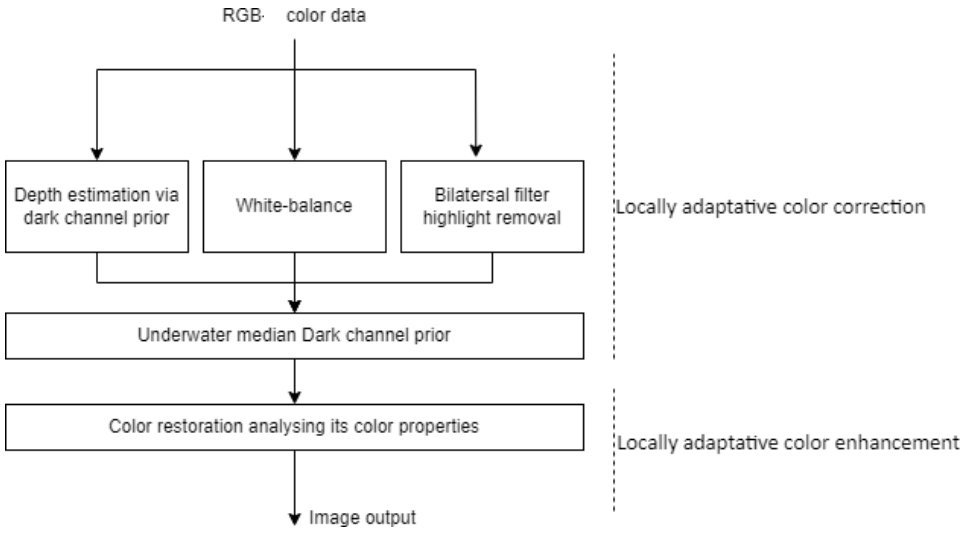


Figure 1.4: Flowchart of the contrast enhancement for images in turbid water method [3].

The second section is called locally adaptive color enhancement (LACE) (Figure 1.4) [17]. This section was made to obtain a color correction to the color derivations induced by the influence of the blue and green color channels of the image. Its first step consists in removing the abortion effect produced by light reflectivity and the scattering, which can be removed by removing the artificial light terms using the transmission map and the reflectivity [3], that were estimated in the last section. Its second step proposes a color correction method to rectify the spectral effect produced by the camera (Figure 1.4), that also affects the colors of the object [3] before reaching the enhanced photograph.

As it was mentioned paragraphs before, the next method is called “Color Balance and Fusion for Image Enhancement”, which was proposed in 2018 by Codruta O. Ancuti [4], and it was born as an alternative to “The Contrast Enhancement for Images in Turbid Water Method” (2015) [3]. This underwater technique basically consists in combining the white balancing image and the image fusion so this can increase the resolution of the image [4]. The white balancing tries to compensate the colors, which were attenuated by the underwater depth light effect. These two sections are fused to enhance the image and enhance the color, restoring the picture like if the water was mitigated.

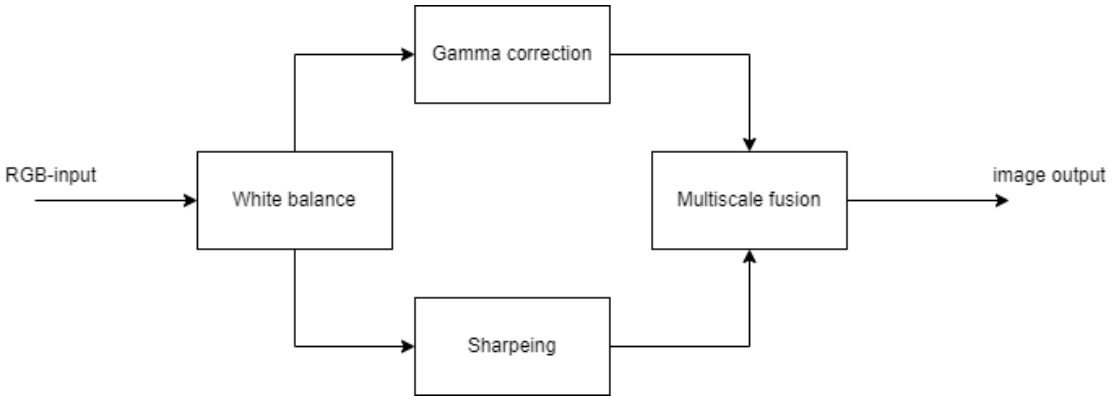


Figure 1.5: Flowchart of the color balance and fusion method [4].

As it's mentioned before the white balancing section (Figure 1.5) aims to increase the clarity of the picture, by removing the light scattering produced by the underwater depth attenuation effect. In this section, the white balance tries to restore the color loss by compensating the red channel, which is not well preserved by the presence of underwater depth compared to the green and blue channels. This section uses the green channel information to provides a proportional compensation to the red channel, so the color attenuation is reduced while keeping a natural appearance in the background of the image [4].

At the same time, the scale fusion seen in (Figure 1.5) is used to enhance the color contrast and the margin shapeless of the white-balance picture [4] of the previous section. As it can be seen in (Figure 1.5) the scale fusion receives information from two inputs, the first input is called gamma correction, which aims to correct the brightness produced by the white-balance, that tends to appear too bright [4]. Finally, the second input (Figure 1.5) writes to the sharpening correction, this helps in reducing the scattering effect produced by the light.

Figure 1.6: (a) first input underwater hazed image. (b) second input depth data. (c) output dehazed image using color balance method in Matlab [4].

As a result, (Figure 1.6) exposes that using RGB information in color balance method shows the dehazed image. This color restoration is caused by the white-balance effect which compensates for the color loss using the red channel [4]. However, the white balance section causes a light saturation, and so the image tends to be too bright in the clearer in some parts of the photo, like the white papers in the image. Using depth data in (Figure 1.6) has the same effect in reducing the scattering effect instead of the sharpening correction section, and takes less time because it uses a second input to obtain the information, so the program focuses on the white-balance color correction alternatively to the sharpening correction.

Finally, the last technique is called sea-thru method, and it was proposed in 2019 by Derya Akkaynak and Tali Treibitz [5] as one of the first techniques that recovers an image using parameters provided from the RGB-D camera to make an image modeling that recovers the original colors and removes the water. This technique focuses on that the hazed image is the addition of the direct signal constant and the backscatter constant, these coefficients require the range of the distance between the camera and the object in the scene, the unattenuated scene, and the light fixture [5] to define the camera image parameters summation equation. The RGB-D camera has an important role in this technique, because it provides the parameters information by sharing the depth map data, and the RGB image which gives the reflectance to the object in the scene that estimates the image constants

and removes the scattering coefficient from the image. Therefore, Sea-thru method results in removing the backscattering and attenuation coefficients that are limited by the visual light wavelength effect (400 to 700 nm) [5], which gives an image without noise and light scattering effect produced by the particles in the environment.

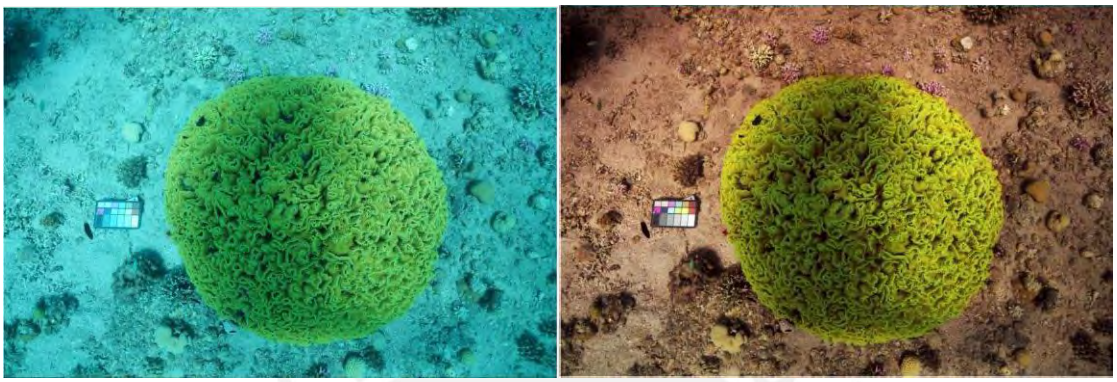


Figure 1.7: Sea- thru method comparative results [5].

As a result, (Figure 1.7) shows that using RGB-D information to remove the water on an underwater image successfully subtracts the attenuation coefficient from the original photograph and the attenuated information of the original scene, making a reconstruction of the picture and showing more resolution details than the other state-of-art algorithms [5].

1.3. Justification:

Table I shows that the state-of-art methods don't make a real time image reconstruction, because most of the cases the algorithms take more than 10 seconds to enhance the image, and in some cases the color reconstruction is not so effective, like in the WCDI method that takes less time (6.935 seconds), but it's not so functional in restoring the original colors like the Color balance and the Sea-thru technique. On the other hand, Sea-thru shows to be the most effective method removing the water in the image, but it is which takes much more processing time. Therefore, there is not an algorithm which removes the water effectively in a short time range.

Table I: State-of-the-art methods time comparison

	Program	Image Width x Height (in pixels)	Running time (per image, in seconds)	Processor
Color Balance	Matlab	4000x2250	14.90269	AMD RIZEN 5 5600H
WCDI	Matlab	4000x2250	25.91087	AMD RIZEN 5 5600H
Sea-Thru	Python	4000x2250	354.7842	AMD RIZEN 5 5600H

1.4. Objectives:

1.4.1. General objective:

Design and implement a new parallel algorithm, that can remove the water in real time of an underwater image.

1.4.2. Specific objectives:

a. Create a database with underwater RGB photographs and their respective depth information.
b. Algorithm selection to remove the introduced distortion when taking a real underwater scene. Also, the potential for parallelization is considered.
c. Speedup1: Port the algorithm implementation from interpreted code (MATLAB/Phyton) into compiled code (C/C++). Sequential version.
d. Speedup2: Design and implement (C/C++/CUDA) the parallel algorithm based on the sequential algorithm. Also, computational complexity is calculated.
e. Evaluate the performance in terms of running time and speedup.

2. Chapter 2: Theoretical framework:

As it was mentioned before a robust color recovery and water removal using depth data in turbid water environments involves removing the particle presence between the object and the camera and estimate the backscatter light coefficient of the color loss in the R, G, B channels. At the same time, the depth data works as one support to make a more accurate estimation of these coefficients. The next chapter will make an explanation of the water removal process used to recover the image and an introduction to the function parallelization to reduce the processing time in the water removal program.

2.1. Compute Unified Device Architecture (CUDA):

The graphic processing unit (GPU) gives a higher instruction throughput and bandwidth capacity than the central processing unit (CPU). The difference between these dispositive remains on the CPU and the GPU capabilities because they were invented with different purposes. While a CPU is designed to use an excel at executing a sequence of operations, which are called threads, because of its lower number of cores that make it difficult to parallelize the function [18]. At the same time, the GPU is capable of executing thousands of threads simultaneously and reducing the process with a program parallelization.

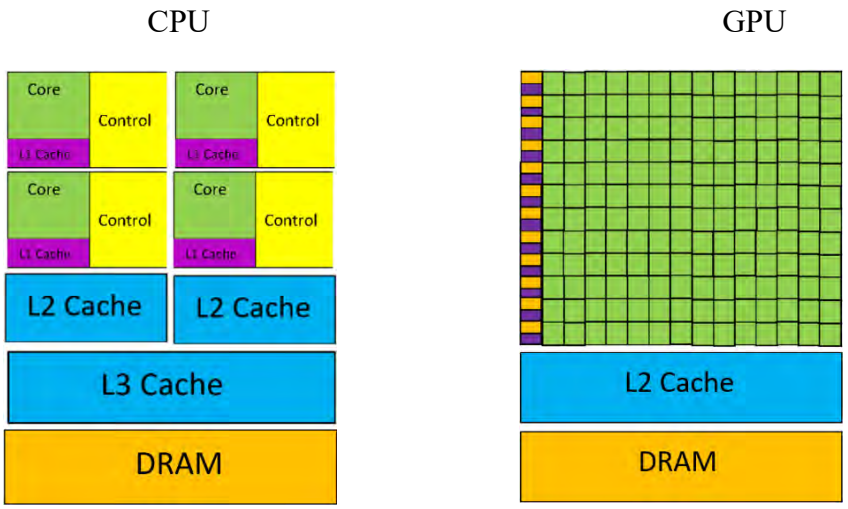


Figure 2.1: CPU and GPU distribution used in the parallelization process

As Fig 2.1 shows the GPU is specialized in parallel computation of the program because of the number of cores. This computation maximizes the performance by accelerating the process by making a communication between the CPU and the GPU, where the CPU allocate the matrix or array data in the graphic processing unit by giving the information from a host to a device, and the GPU parallelizes the process using blocks and threads that allocate the array position in the GPU and calculates the function, which accelerate the process. Fig 2.2 shows a better example of the communication process between the CPU with the GPU [18].

To summarize, the process has a mix of parallel parts and sequential parts, so the program works with a CPU to GPU communication in order to maximize the performance and reduce the time [18].

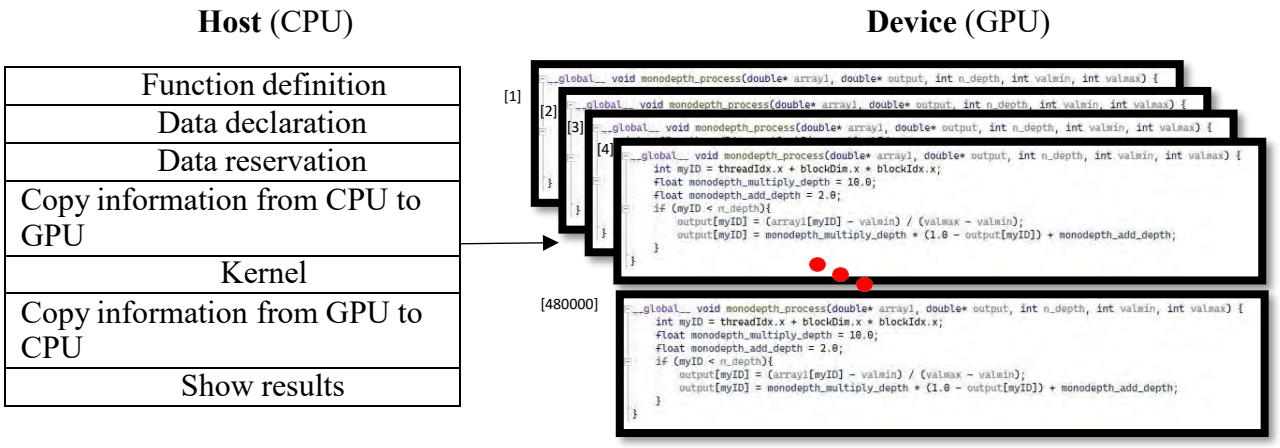


Figure 2.2: CPU function structure and GPU function parallelization (1 to 480000 from an array)

As Fig 2.2 shows that the communication process in the GPU starts in the central processing unit, where the function is defined and the information is declared, this data is translated to the GPU reserving the memory space using “cudaMalloc”, which allows allocating the memory size in the GPU to parallelize the process, after this step the program take this information to the device using “cudaMemcpy” which copies the data between the host and the devise pointed in a space of memory area, after all this process the function enters to a kernel to parallelize the algorithm using a number of blocks and threads, this means that some elements in an array are calculated at the same time instead of using a sequential loop that slow down the process. The blocks are defined as the number of spaces used in a function to accelerate the function which gives a number of threads that is used to simultaneously estimate the value of an array. Finally, the information returns to the CPU and it’s copied in a space of memory to return the value and after all the process the function can show the output results.

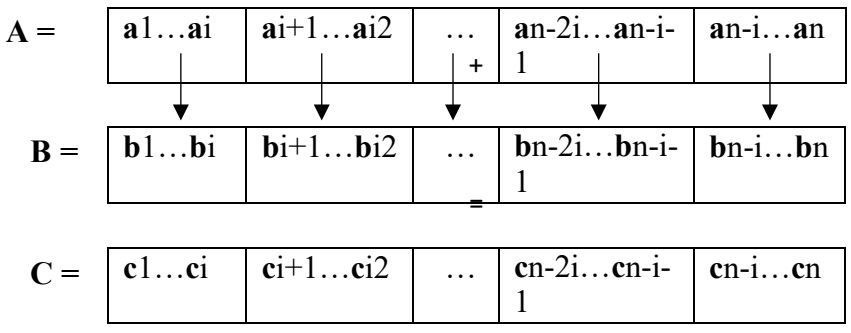
The main task of designing a parallel algorithm using the communication between the CPU and the GPU could be a challenging labor compared to the sequential process, for that reason Joseph Jaja proposed in 1992 different types of parallel techniques designed as general guidelines used to directly apply this technique [19], in this case the algorithm

is focused in applying one of these methods, which is called “Partitioning” and it’ll be explained in the next subsection.

a. Partitioning:

Partitioning techniques is dispersing the problem into a number of subproblems in an array with the same parameters like equal size and then solving the number of subproblems at the same time. In its basic form, this method divides the input into a number of processor pieces, and then calculates problems associated with the number of processor pieces currently [19]. Let’s put an example where a function is composed by two arrays which are A[n]= [A1, A2, A3, …, An] and B[n]= [B1, B2, B3, …, Bn] where n is the number of elements included in the sequential array. This section considers merging the two arrays into a third array called C[n]= [C1, C2, C3, …, Cn]. The main function of this method is replacing the serial algorithm with one parallel function that is related in partitioning A and B arrays into subproblems that will be merged into a third solution using the number of processor pieces [19]. The next part will show an example of partitioning application where A and B are the representation of arrays and C is the output array obtained in the addition of the values:

A + B= C
A = (a1, a2, a3, a4, …, an)
B = (b1, b2, b3, b4, …, bn)



2.2. Anisotropic diffuse depth map filter:

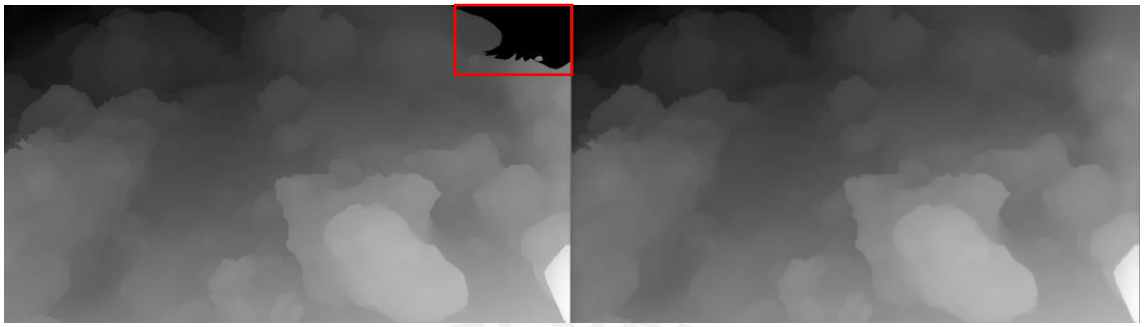
The importance of an anisotropic diffuse filter before the water removal method remains in the depth map image. The newest RGB-D cameras are capable to provide a higher resolution than other tools. However, these parameters are limited due to the light absorption in the scene [20]. For that reason, the RGB-D depth resolution utilized in underwater environments is limited by the reflection and other factors that generate erroneous and incomplete information by omitting data from the original depth map.

The anisotropic diffuse depth map filter is classified as a denoiser and inpainting filter, which tries to reconstruct the depth by using supplementary data for the depth signal for the optimization of this process [20]. This means, that this technique increments the resolution of the depth information parameters by attaching the edge signals from the RGB image to be able to use it in the reconstruction of the depth map. At the same time, the depth parameters are mapped to the intensity camera by getting a depth map projection containing a sparse set of base depth points in the RGB space image [21].

The anisotropic diffuse technique applied utilizes the depth model is adjusted so they can have the same scale then the process uses an algorithm called Conjugated gradient to find a better representation of the depth model resulting in the photograph with the completed information from the original photograph [20]. So, the filter model utilizes the RGB image to compute the 3d surface normal in every pixel location by sampling neighboring pixels within a depth limit and attaching the least square stage [22].

The process starts with two inputs, the RGB photograph and the depth map, which are classified or separated in a process called segmentation based on their depth and color gradients. After this process, the parameters measured are utilized to create a hierarchical segmentation and estimate the support structure [22]. After all this process the columns

and the rows from the original photograph are replaced for a new matrix reshaped with the new values which are reshaped depending on the height and width from the depth map. Fig.2.3 shows the result of applying this filter.



(a) (b)

Figure 2.3: (a) Original Depth map with some imperfections like the black hole (“0” distant Value) in the image. (b) Depth map after using the filter.

2.3. Flowchart of water removal process:

To restore an underwater picture the RGB-D image requires a process which aims to reconstruct the photograph. This process consists in using a preprocessing depth map, which allow the algorithm to recover the image by helping in the backscatter coefficient research, and estimating the wideband attenuation, like in the figure 2.4, so the depth map is used as a support to the RGB image, because this data provides an accurate information to estimate the attenuation level and helps in the research of some coefficients. At the same time the RGB input provides the image formation modeled by the main equation Eq.1, where Ic (c = R, G, B) is confirmed by the original image including the light attenuation and the wavelength color distortion, and Dc is the direct signal that has the attenuation data and Bc is the effect produced by the light reflection, that attenuate the image, which is called backscatter [5]. The main reason of water removal methods is removing the backscatter effect and the degradation produced by the direct signal, resulting in the image output without light attenuation or color degradation.

$$Ic = Dc + Bc \qquad (1)$$

As it was mentioned in the last paragraph the main function of this program is subtract the degradation and the light attenuation. So, to make this possible the algorithm needs a process that helps to find this coefficient and subtract it from the original image. Fig 2.4 shows the process used to remove the water from the original RGB image using the depth data.

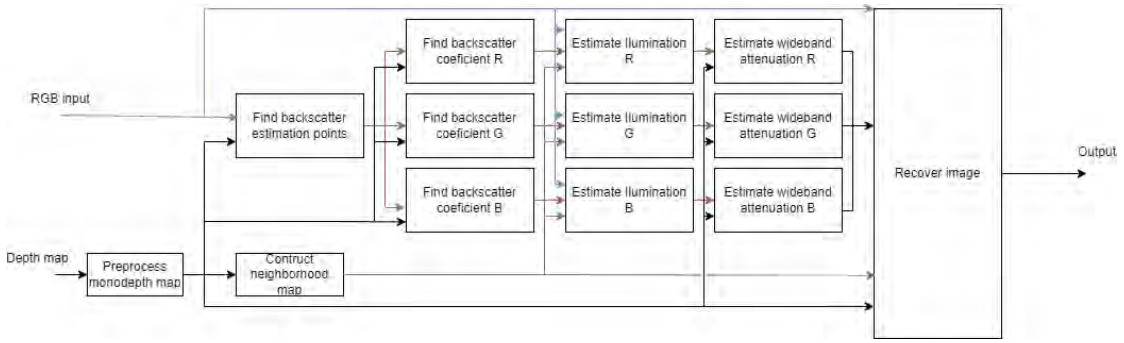


Figure 2.4: Flowchart of the theoretical framework based on the Sea-thru method using depth information [5]

As Fig 2.4 shows the subtraction of the backscatter and the light attenuation is constituted by a process that restores the depth data and estimates the coefficients before recovering the images. The next paragraphs will make a general explanation of this process.

a. **Preprocess mono depth map.-** This section proposes a task approximation as a picture restoration problem, and tries to make a prediction of the depth per pixel from a single depth image ,which is read as a matrix for the program,this section works with a distance range pixel, which is conformed between [0- 255], and it's used to estimate the distance between the object, '0' is the longer distance, but in some cases is interpreted so the depth map shouldn't go that far, and 255 which is the nearest distance between the object and the camera.

b. **Construct neighborhood map. –** Consist in neighboring every pixel of the depth 1 channel image matrix (Height x width) in order to reduce the impact in a single temporal noise. [5] define it as 4-connected pixels in the height perwidth range of the photograph, which are closer to it than a radius approach.

c. **Find backscatter estimation point. –** The main function about this part is de-scattered the points of the image, and works as a backscatter removal section,which depends of the RGB data and the depth map information, and it classifies the backscatter coefficients depending on the RGB

channel, that are red, green and blue. And at the same time, it estimates the backscatter points that are used to subtract the degradation provoked by the color degradation inthe input RGB image, as it can be seen in the Eq.1.

d. **Find the backscatter coefficient. -** This section research for the backscattering coefficient, also called 'Veiling light' [23], this constant is located in the Bc attenuation effect from the Eq.1, so this component doesn't carry any object to camera scene data, so this coefficient yields to a color degradation and a picture attenuation in the original scene [23].

e. **Estimate illumination. -** This section tries to find complementary model of underwater pictures, in the dimensional space of the image [1] the illuminationis used in the image to enhance the color quality, and it helps to make an illumination adjudgment in the original picture, this helps to make a color reconstruction of the image that is used to recover the image.

f. **Estimate wideband attenuation. -** The attenuation coefficient for the underwater image makes a research of the organic elements located in the spatial 3-dimensional space in the image, this part includes the light reflectionprovided by the underwater particles between the object and the camera, and the illumination provided by the underwater environment that helps in the underwater photograph quality.

g. **Image recovery. -** This is the last process before removing water from the image, the recovery section requires the estimated backscatter light coefficients and the color attenuation coefficients, this section uses these parameters to subtract the degradation from the original image. As a result, the algorithm provides the original photograph without the degradation coefficients, which can be used in photogrammetry for 3d

modulation.

2.4. Preprocess mono depth map:

In the case of the mono depth map reconstruction is described as a method that suggest an incrementation on the depth accuracy predicting the depth distance of the camera, which yields to the conversion of the normalized depth information into the distance of the object and the camera. Fig 2.5. shows processed depth map in python.

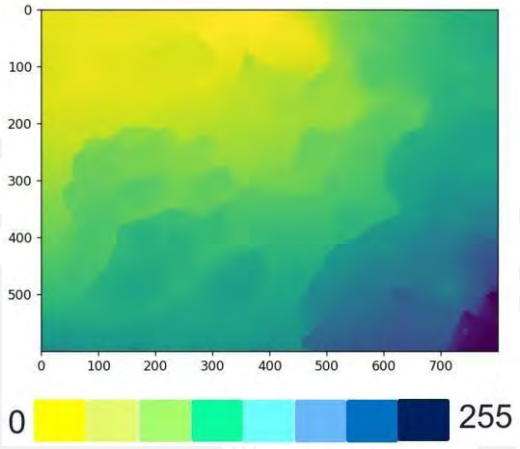


Figure 2.5: filtered depth map with a distance range between [0-255]

The preprocessed mono depth map algorithm utilizes the normalization process in every pixel of the image by modifying the photograph. The implemented data works with a depth range [0 - 255] which represents the distance between the object and the camera like Fig 2.5 shows, with 0 as the longest distance and 255 as the shortest distance. However, the information should be inverted so the longest value could represent the longest distance while “0” represents the shortest distance.

The function requires the depth map as an input, which is modified by normalizing the pixels to a range of [0 - 1] and then invert the pixels so “0” could represent the shortest distance and “1” the longest one. And finally, the normalized values are multiplied by an estimated distance constant using the depth map, giving as a result a range of distance in meters that can be used in the estimation of the parameters of the process.

2.5. Construction of the neighborhood map:

Most of the images require the backscatter coefficient to estimate the photograph attenuation and take away that value to estimate the image. However, the process also requires to make a color cast and a color compensation to restore the original colors. For that reason, it is useful to estimate the contras of the image in order to use it on the estimation of the illumination map. This process divides the image into different sections that estimate the contrast on the images. This technique works with the next equations:

$$a'c(x,y) = \frac{1}{Ne} * \sum_{Ne} ac(x', y') \quad (2)$$

$$ac = Dc(x,y) * p + a'c(x,y) * (1-p) \quad (3)$$

Eq.2 and Eq.3 shows the local space average color value method, where x and y are the coordinates of the position in the matrix, p is defined as the area computed of the image that varies depending on the height and the width of the RGB image, Dc is the direct signal in the RGB image, and Ne is described as the neighborhood map, which is described as the 4-connected pixels around an element in the image matrix (x, y) [5]. In the case of the initial value in the ac(x, y) will be interpreted as zero so the starting value would be insubstantial.

2.6. Find the backscatter coefficient:

As it was mentioned in section 2.3 in Eq.1 the backscatter component B(r, g, b) carries the no data added in the input image, which degrades the color of the image generating a blur degradation the image. The backscatter component depends depth map to estimate every backscatter value of this matrix. The process to estimate the backscatter component is to make a research of the darkest or the shadowed pixels in

the image matrix, similar to the dark channel prior process (DCP), but instead of estimating the depth map this process works with the depth input data [5].

The backscatter coefficient research starts with the range map division of the 10 clusters which has the minimum and maximum value of the preprocessed depth map which is represented by a 'z'. In every range the original image makes a research for the RGB triplets to the base 1 percentile [5], which means that the function will store the darkest pixels and will use it to estimate the parameters in Eq.4:

$$B(r,g,b) = Jc(r,g,b) * e^{-\beta_c^D * z} + Ac * (1 - e^{-\beta_c^B * z}) \quad (4)$$

Where B(r, g, b) is the backscatter coefficient, z is the distance value, Jc and Ac are binary constants which depend of the preprocess value and the z max and min value which can get the [0,1] range value, and the β_c^D and $\beta_c^B \in [0,10]$ [24]. So, with all these values the backscatter constant is estimated for every pixel depending on the distance depth map value and the direct signal component obtained in the general value Eq.1.

2.7. Estimate Illumination:

The backscatter removal from the photograph yields to a darker matrix. For that reason, the linear illuminance map is used to take out the qualities from the chromatic color of the three-channel scene in the image [25]. The illumination model was based on the atmospheric effect produced in the visual light wavelength range, generating the reflectance model of the illumination spectra, that span the range of the atmospheric parameters [25]. However, it's impractical to modulate the illumination spectra in sea-thru method because this modulation requires multiple images to make the reconstruction of the image object. For that reason, the algorithm in image reconstruction requires taking the backscatter coefficient from the original image

where the image is improved. However, there is a red-light attenuation presence in the picture that generates a color distortion making it look greenish and bluish [24]. The main function of this section is removing this color effect to reduce the color distortion and compensate the reflectance in the three channels of the image.

According to the Land and McCann studies (1967) the direct signal component is composed by the reflectance component and the luminance effect from an image [24], as shown in the Eq.5:

$$D(r, g, b) = R(r, g, b) * E(r, g, b) \quad (5)$$

$$E(r, g, b) = f * ac \quad (6)$$

Eq.5 show E(r, g, b) which is the estimated illumination map and it's used to remove the color distortion in the image to compensate the red light attenuation, this function is composed by the multiplication of the geometry scaling constant (f = 2) and the local space average (ac) [5], as Eq.6 shows.

2.8. Estimate wideband attenuation:

In section 2.1 was mentioned that the image is originally composed by two elements which are defined in Eq.1, that are the addition of the direct signal of the attenuated data (Dc), and the backscatter coefficient (Bc), which is composed by the coefficients that has no information about the input RGB picture, so this doesn't provide information and degrades the image. So, in other words, the Direct signal (Dc) provides the information carried by the special environment between the object and the camera, this signal is composed of some coefficients that help providing the image information, this coefficient is called 'wideband attenuation' [5]. The original Dc is composed by the original colors and an attenuation coefficient that degrades the underwater photograph like in the Eq.7 which is the Direct signal composition, provided by [23].

$$D(r,g,b) = J(r,g,b) * e^{-\beta_c^D(Vd)*z} \qquad (7)$$

As Eq.7 shows the Direct signal is composed by different elements, where J(r, g, b) is the original scene without any attenuation or distortion, z is the distance between the object and the camera that is provided by the depth camera and it's located around the [0 - 255] range, and β_c^D(Vd) is the wideband attenuation coefficient, where B is a continuous value of the same scene. Eq.8 shows that wideband attenuation coefficient requires the z constant, the J(r, g, b) and the D(r, g, b).

$$\beta_{D_c} = -\ln\left(\frac{J(r,g,b)}{D(r,g,b)}\right) * \left(\frac{1}{z}\right) \qquad (8)$$

The algorithm was implemented in python and uses the depth and the estimated illumination maps as inputs to estimate the wideband attenuation coefficient, this algorithm tries to make an equivalent function to the Eq.8 and utilizes the illumination and the depths to estimate the D(r, g, b) and the J(r, g, b) values , and at the same time depths is also used as the z value using the preprocessed depth map.

2.9. Image recovery:

Fig 1.1 shows the image formation model which is based on the McGlamery (1980) and Jaffe (1990) studies [24], relative to this model the original RGB captured image is the inclusion of three different elements that are implemented in the next equation:

$$I(r,g,b)(x,y) = D(r,g,b)(x,y) + B(r,g,b)(x,y) + F(r,g,b)(x,y) \quad (9)$$

Eq.9 shows the image model represented by an I(r, g, b), where I represent the original image with the attenuation effect and the backscatter coefficient and (x, y) denotes the image matrix coordinates. At the same time, D(r, g, b) represents the direct signal, B(r, g, b) the backscatter attenuation and F(r, g, b) represent the forward backscatter coefficient, which has no significant value compared to the direct signal and de backscatter coefficient, so it's omitted [24], which yields to the next equation:

$$I(r,g,b)(x,y) = D(r,g,b)(x,y) + B(r,g,b)(x,y) \quad (10)$$

Eq.10 shows the addition of the components that are contained in the image, but these components are comforted by coefficients that has an effect in the degradation of the image, which are the wideband attenuation (β_C^D) and the backscatter coefficient (β_C^B), respectively. These components are expressed in the Eq.11 [5]:

$$I(r,g,b)(x,y) = Jc * e^{-\beta_C^D * z} + Bl * (1 - e^{-\beta_C^B * z}) \quad (11)$$

Where Jc is defined as the undegraded image without attenuation of the scene, Bl is the undegraded light and z as the preprocessed depth map [24] After knowing that the Jc is the unattenuated scene caption, which is the desired result, the main function could be represented by this modification to the Eq.10: D(r, g, b) = I(r, g, b) - B(r, g, b). at the same time this equation can be replaced by the Eq.11 to find the unattenuated value of the function, which is represented by the Eq.12.

$$Jc = Dc * e^{\beta_C^D * z} \quad (12)$$

Where Dc was obtained after estimating the illumination map that is used to remove the color distortion [24], this value is used with the wideband attenuation coefficient and the depth map to find the original scene without attenuation.

(a) (b)

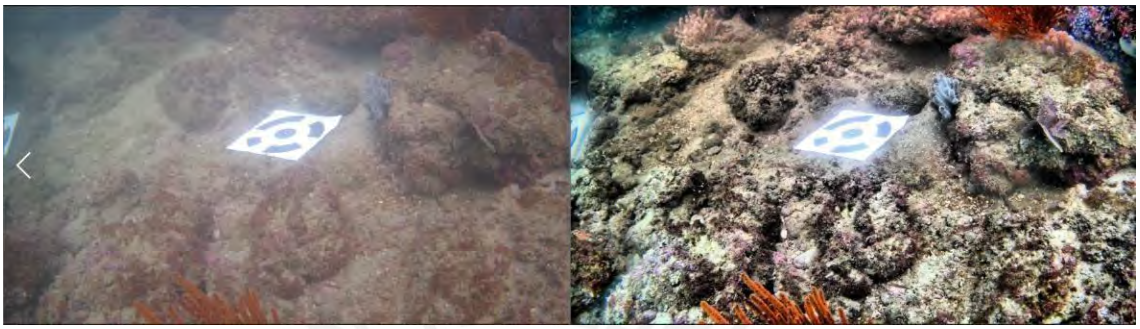


Figure 2.6: (a) Original image with attenuation and backscatter effect. (b) Original Image without degradation.

Fig 2.6 shows the obtained result after removing the backscatter and the color degradation in a Paralenz photograph, which has 4000 x 2250 pixels (Width X Height) which means that is attached by 27’000’000 of element in the RGB image, leaving only the undegraded scene.

3. Chapter 3: Methodology:

The water removal process requires the RGB image and the depth information to restore a photograph by removing the color degradation and the backscatter coefficient. However, this process needs of some steps to restore the accurate data from the picture. This chapter tries to make an explanation about the methodology used for the water removal process, which includes the image acquiring, the image processing where I make an explanation of the filters and the water removal process used and the 3D modulation.

3.1. Processing:

The water removal process uses the image formation model to subtract the backscatter coefficient and estimate the illumination to find the color degradation [5]**.** However, what I'm proposing is that instead of using an RGB-D image or estimation the depth using dark channel prior like [5], I'm going to use the depth map information provided by photogrammetry. Nevertheless, using a photogrammetry depth map doesn't provide an accurate measurement about the depth data, which yields to an erroneous image reconstruction. For that reason, it's necessary to add some algorithms to reconstruct the depth map erroneous information and increase the resolution before utilizing the water removal process.

In this section I'll make an explanation of some of the specifications needed for the water removal process, the proposal real time method and an explanation of the followed methodology using the block diagram of the water removal process.

3.1.1. Specification:

The algorithm needs to follow some condition to work properly. Some of these conditions must be applied in the water removal process:

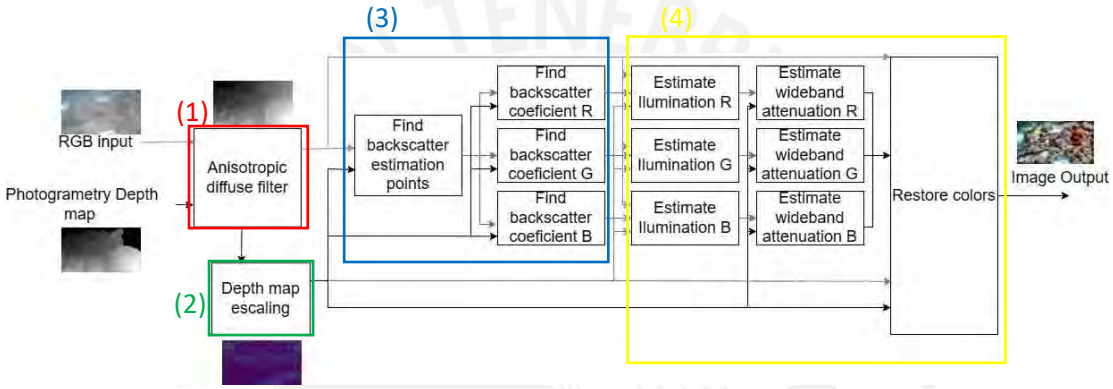
- The algorithm works only with ".png", ".jpg" and ". raw" files images. these images can be used at the same time. But they need to have the same height and matrix to be restored, this is because the depth map has to provide the same information about the RGB image. So, the depth data should have the same height and width, but the difference is that the image works with 3 channels and the depth map with one.
- The code should be able to run a lot of images in one compilation, so the program works as a loop where the program processes the image after it finishes with another. Therefore, the program can remove the water from various images in one compilation.

3.1.2. Proposal

Table I shows that the underwater images, especially which are composed by a large number of pixels in the height and the width, takes much time to remove the water. For that reason, I'm proposing to use the water removal algorithm but with a parallelization process. So, the program could process the image in real time by transforming the image matrix into a vector and use the partitioning technique to parallelize the process. Therefore, the processing time takes around [20 - 50]ms to reconstruct the image.

3.1.3. Block diagram of the proposed solution

The water removal process works with a RGB input and a depth map provided from the photogrammetry edition. However, the photogrammetry process provides some erroneous depth data. So, the algorithm works with a filter based on the [22] studies, that tries to fill this erroneous data by adding information with the geometric structure inferred from the depth [22] After this process the algorithm works with the water removal process as Fig 3.1 shows:



(1) Depth map reconstruction

(2) Normalization of the depth map and reconstruction and refinement of the neighborhood map

(3) Backscatter removal

(4) Color compensation and color cast removal

Figure 3.1: Block diagram used in the water removal process

Fig 3.1 shows the block diagram where it combines the water removal process with the anisotropic diffuse filter to reconstruct the underwater picture. So, it could use the photogrammetry depth information as an input. After this process, the depth map is processed and is used to construct the neighborhood map. This information is used to find the backscatter coefficients and estimate the color degradation. And finally, the algorithm subtracts the backscatter coefficient and the color degradation from the original

photograph to recover the picture. As a result, the output shows the photograph dehazed and with the original colors.

The block diagram was divided in 4 sections that represent different actions made by the program, the next sections will make an explanation of how this process work and how it's going to be used in the implementation.

3.1.3.1. Depth map in-panting

Underwater conditions allow to the capture of digital images that can be utilized to generate a three-dimensional representation of the environment through the use of photogrammetry programs like Methashape. These software applications analyze multiple images and establish correlations between them to create a model. Additionally. Photogrammetry is employed to extract depth information from the resulting model. However, it should be noted that the depth map produced by a photogrammetry program may not be entirely accurate. This situation arises due to certain areas containing a Null value, represented as black holes, which negatively impacts the final image output. Consequently, these erroneous data directly influence the RGB representation.

Therefore, the primary purpose of this filter is to identify correlations among neighboring pixels in the single-channel depth map. By doing so, it becomes possible to utilize the filter to fill the inaccurate or missing data in the photograph.

To make this process possible I developed a new algorithm based on pixel correlation, which tries to estimate the real values of the depth map using some values of the image. The next paragraph will explain the functioning of this algorithm.

a. Flowchart of the pixel correlation algorithm:

The algorithm was divided in two sections, the first one is called “Finding holes depth map” and this subsection tries to find the number of pixels in blank located in a section on the row of the image and the pixel value situated at the opposite location of the hole. This subsection would make possible to estimate the values of a row by replacing the blank values with correlated data between the opposite locations in the hole.

“Finding hole depth map” function will locate the holes in a row of the image matrix and will count the number of pixels with the Null value and will save the initial value in some space of the image matrix. This is an example of how the function works, it will detect the 0 value and will count until it finds the 240 value. After it finds the 240 the function will save the count and the first pixel value like the diagram below, where the table represents the row of the image matrix where the “0” represent the hole between 150 (the first value) and 240 (the last value).

- First Value = 150
- Count = 0

+ ⟶

150	0	0	0	0	0	0	0	0	0	0	0	0	0	240

- Count = 13

⟶

150	0	0	0	0	0	0	0	0	0	0	0	13	150	240

So, the function will run the row pixel by pixel and will count the ceros until it finds a value different to cero, and after it finds this value the count and the first value will be saved in the matrix. To make this process possible the function operates the following manner:

1- The function will run the image in a loop, it firstly will run the height. Where it will assign the value 0 to a constant called “Hole” which will detect the cero value in a row, if it detects a 0 it will change its value to 1 indicating that a Null was

detected. And a constant called “count” which count the number of pixels with Null value located in the hole.

2- The next part is a loop that will work with every pixel located in the width, this will be represented as a “for” function, which starts in the first value “0” and will end in the last pixel value “W - 1”.

3- The next part will make an evaluation of the state of the pixel. If the state of the Hole is 0 it means that no hole was detected before and the value of the Depth-map pixel is 0 which means that the function had detected a hole, so it will change its state as “1” and it will start the “count”.

4- If it is already on the Null section the “Hole” will be in “1” and the depth map value will be “0” so it will continue with the count to estimate the number of values with cero.

5- If the “Hole” value is “1” but the number of depth map pixel is different to 0 means that loop already left the loop so the count and the initial pixel must be saved in the matrix. So, the data would be written in the two last pixels, so the other function could know the initial pixel and the number of cero values. And then the functions initialize the values “Hole”, “count” and “initial pixel” so they can be used in another whole.

The next graphic is a representation of the flowchart followed by this process, where at the end the function will try to save the data in the last two values of the hole so the first value and the count could be known by the “Filling depth map function”:

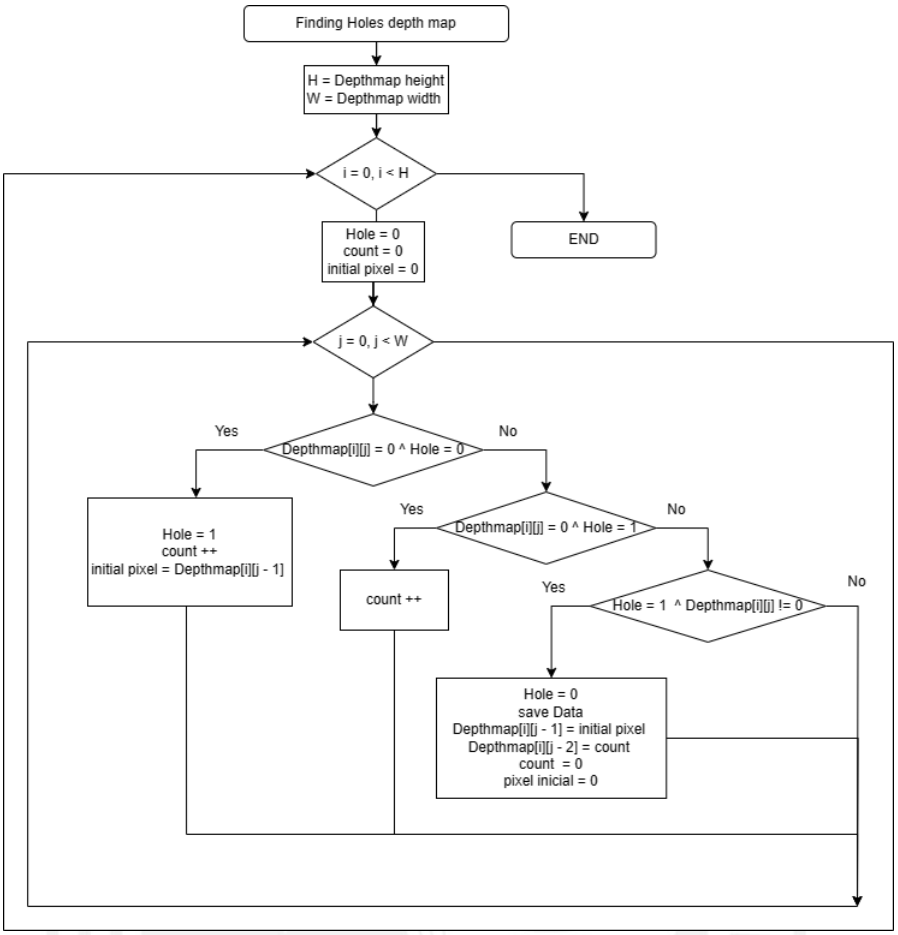


Figure 3.2: Finding holes depth map flowchart

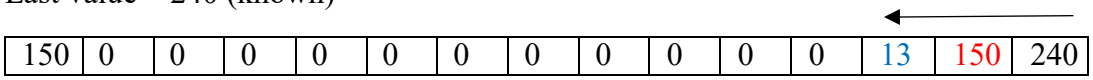
After this process the algorithm will use the information provided by “Finding Holes Depth map” to replace the missing information in the ceros. To make this process possible the “Filling depth map function” will read the information from right to left, in order to use the make research of the empty pixels, and after the hole is found the algorithm reads the last two pixels where the initial value and the number of pixels with zero where stored.

After the function finds the hole, the algorithm will store the values of the last and the penultimate pixels in two constants called “initial” and “count” so the function could work with the values of the initial pixel and the number of empty pixels. The next graphic makes a representation of the process.

Number of pixels with 0 = X (unknown)

First value = Y (unknown)

Last value = 240 (known)

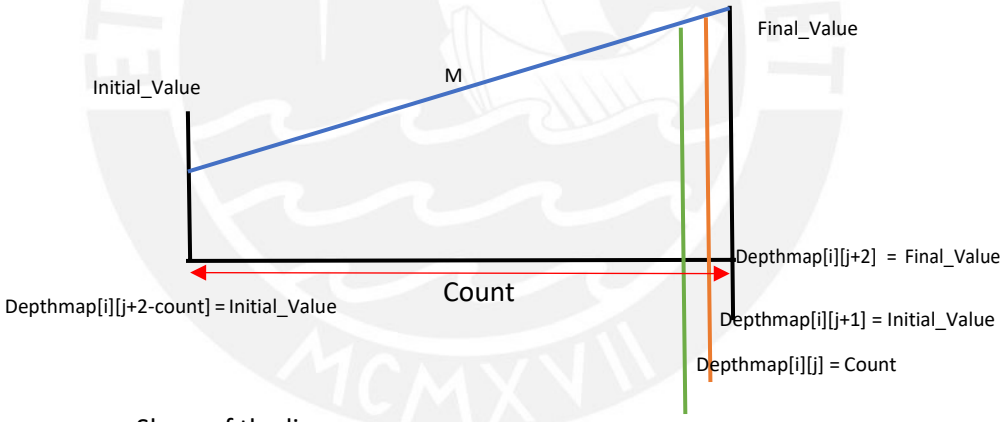


Number of Pixel with 0 = 13

First Value = 150

Last Value = 240

So, after the function knows the first value and the number of pixels, the algorithm will use these constants in a line function where it would be easier to estimate the slope "m" and the initial and the last point of the line. So, after this process the algorithm will use the line function to estimate the value for every pixel position in the hole. The next image makes a representation of the line function process followed to estimate every value in the image.



Slope of the line:

$$M = \frac{FinalValue - InitalValue}{Count}$$

Initial Position (Estimated when the algorithm detect the hole):

$$InitialPosition = j + 2 - count$$

Line Function:

$$Depthmap[i][j] = M * (j - InitialPosition) + InitialValue$$

The function operates the following way:

1- The function will estimate the value of the heigh and width and will assign it to two constants “H” and “W” respectively.
2- The algorithm will run in a loop where it can work with every row of the image and then initialize the constants “Hole” that detect the 0 value in the array, “Value” makes reference to the first value of the array, “count” refers to the number of ceros and “m” refers to the slope of the estimated line.
3- Now it will enter to a loop where it evaluates every pixel in the width of the image but in this case, it will be executed from right to left.
4- If the function detects a “0” the “Hole” changes its state to “1”, the function will estimate the first value and the count will be saves in a constant. After this process the algorithm will estimate the slope value and replaces the last 2 pixels with the respective line function value.
5- If depth-map keeps detecting ceros the depth map values will be replaced with the line functions values.
6- If the algorithm detects that the new pixel is cero, the constants will be initialized and the process will repeat until all the ceros in the depth map are replaced by the estimation values.

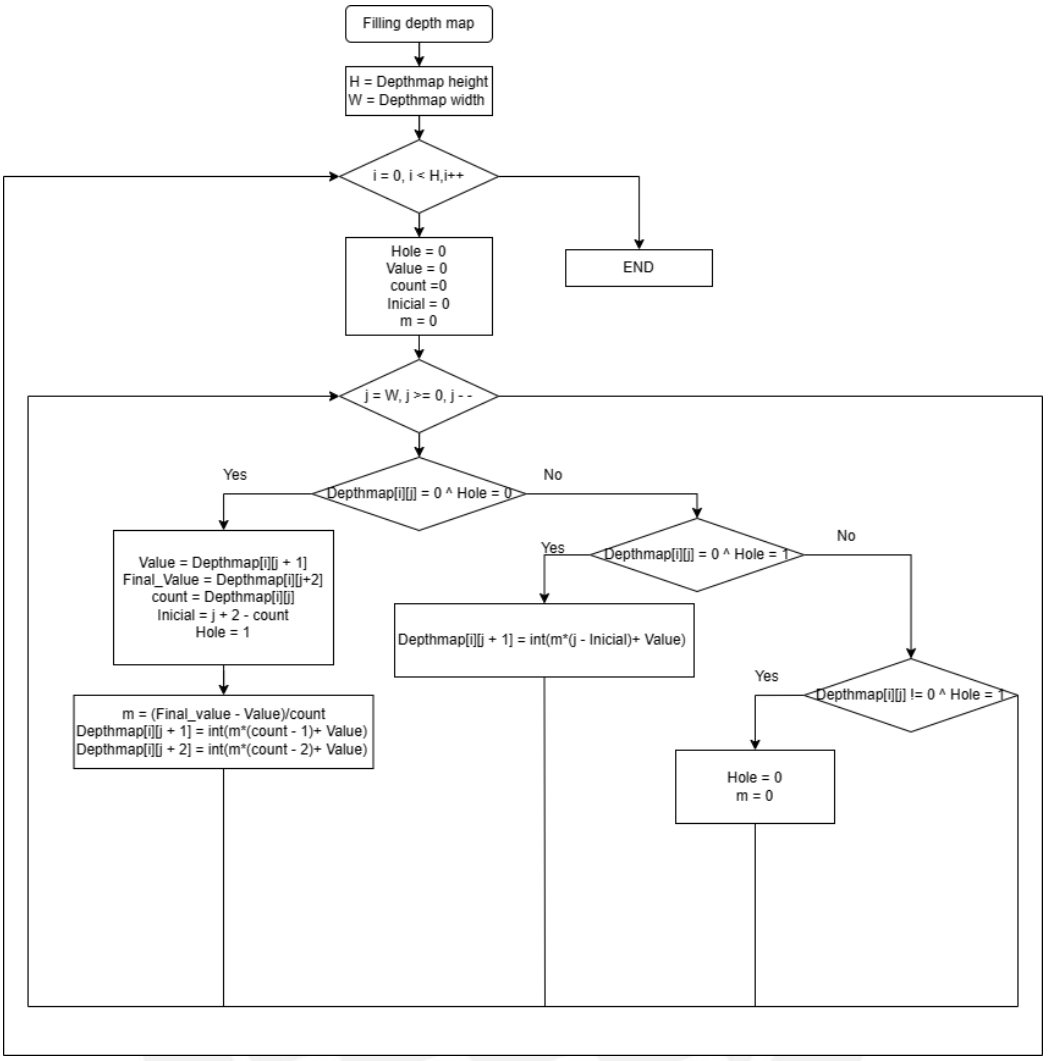


Figure 3.3: flowchart of Filling depth map reconstruction

To summarize the code is composed by to part, one that detect the holes and save the first value and the number of pixels and other one that uses the saved information to reconstruct the photograph using a linear function to estimate the values in the empty areas of the photograph.

3.1.3.2. Construction of the neighborhood map

The main function of this process is to classify the different contrast areas that could be found in the depth image. This process helps in the estimation of the illumination map, this is because the provided information result in the reflectance sections in the original photograph. The function smooths out the depth changes in an image.

The function construct neighborhood map takes an input depth image and constructs a neighborhood map where each pixel in the output corresponds to the size of its local neighborhood. The neighborhood size is determined by counting the number of pixels in a circular region around each pixel with a given radius. The output map can be used for various image processing task specially to find the contras in a depth map.

To make this process possible the program will estimate a constant that will define the range of values as a condition in the map, if it doesn't follow the condition it means that the area has found other contrast that doesn't follow the condition and will pass to another area. To summarize, the function will locate a random value to be located and will assign a value called n_neighborhood, which is a unique constant that won't be repeated after is used in an area. The function will expand the n_neighborhood value around all the located area until it finds another contrast that doesn't follow the conditions, in that case the function will leave the area and will choose another location to expand the value. The next algorithm makes an explanation of the followed process.

a. Serial algorithm:

Input: The one channel matrix utilized by the Depth map input and the epsilon value used as a constant to find the contrast areas in the map.

Output: Neighborhood map matrix.

begin

Max_depths : Find the maximun value of a depth map.

Min_depths : Find the minimun value of a depth map.

Eps = (Max_depth – Min_depth)*epsilon

Comment: ► Initialize the number of neighborhooods to 1

n_neighborhoods = 1

Comment: ► Initialialize an empty neighborhood map

nmaps = zeros(depths)

Comment: ► While nmap has values similar to 0

```
While(nmaps == 0):
        Locs_x = 0 coordinates in nmaps x
    Locs_y = 0 coordinates in nmaps y
    Comment: ►Choose a random value around the locs_x values
    Start_index = random(locs_x)
    Comment: ►Stablish coordinates
    Start_x = locs_x[start_index]
    Start_y = locs_y[start_index]
    Comment: ►Stablish coordinates inside a list called q
    q = [Start_x, Start_y]
    Comment: ►While q is not empty
    While q != 0:
        X = q[0]
        Y = q[1]
        If(depths(X,Y) – depths(Start_x, Start_y)) <= eps:
            Comment: ►fill the values of the matrix nmaps with the constant
        n_neighborhoods
            nmaps [x, y] = n_neighborhoods
```

Comment: ► The function searches the neighborhood values in the corners and the bases, which fills the values and provides values to q, ones the function doesn't detect this conditions the list will start to reduce itself.

```
if(0 <= x && x < depth.rows - 1):
        x2 = x +1
        y2 = y
        If (nmaps[x2,y2] == 0):
                q = [x2,y2]
if(1 <= x && x < depth.cols):
        x2 = x -1
        y2 = y
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
if(0 <= y && x < depth.rows - 1):
        x2 = x
        y2 = y + 1
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
if(1 <= y && x < depth.cols):
        x2 = x
        y2 = y – 1
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
```

end

The last algorithm was made in serial so it can work in the CPU. However, to make this process possible in the GPU the function the algorithm was based on a PRAM machine, and have the preliminary "start_index" values in an array so it could start on those values instead of making the process randomly. The next function shows the parallel algorithm.

b. Parallel algorithm:

Input: Depths matrix with height y width values, and epsilon values. In this case it's also considered a seed array where the function memories the start index values, so it could start around those values instead of doing the whole function, which was called random_seed.

Output: Neighborhood map matrix

```
Start_index = random_seed[i]
Comment: ►Stablish the coordinates knowing the start index values
Start_x = locs_x[start_index]
Start_y = locs_y[start_index]
Comment: ►Stablish the list q
q = [Start_x, Start_y]
Comment: ►While q is not empty
While q != 0:
    X = q[0]
    Y = q[1]
    If(depths(X,Y) – depths(Start_x, Start_y)) <= eps:
    Comment: ►fill the values of the matrix nmaps with the constant n_neighborhoods
    nmaps [x, y] = n_neighborhoods
```

Comment: ► The function searches the neighborhood values in the corners and the bases, which fills the values and provides values to q, ones the function doesn't detect this conditions the list will start to reduce itself.

```
    if(0 <= x && x < depth.rows - 1):
        x2 = x +1
        y2 = y
        If (nmaps[x2,y2] == 0):
                q = [x2,y2]
     if(1 <= x && x < depth.cols):
        x2 = x -1
        y2 = y
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
    if(0 <= y && x < depth.rows - 1):
```

```
        x2 = x
        y2 = y + 1
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
    if(1 <= y && x < depth.cols):
        x2 = x
        y2 = y – 1
        if (nmaps[x2,y2] == 0):
                q = [x2,y2]
end
```

3.1.3.3. Preprocessing depth map

The backscatter removal process is related to the degradation between the object and the camera, for that reason this process requires to use the Depth map to estimate the backscatter components. However, the depth map is composed by one channel pixels and doesn't give a reliable information of the distance, because this has a range of [0 - 255] uchar pixels, where the nearest distance is represented by 255 and the longest by 0, which means that the depth map needs to be processed before using. For that reason, it is necessary to make a preprocess of the depth map before using, this preprocess should give as a result a double representing the distance in meters. To make this process possible it is necessary to make a normalization of the process, which means that the algorithm will take the greater and the lower value of the depth map to operate in a range of [0 - 1]. Nevertheless, normalizing the matrix is not enough to utilize the distance, so after this process the matrix will invert its values. This is because the closer pixels have a maximum value, which could be represented as they are farther than the other pixels. At the same time, the values are multiplied by a coefficient that tries to represent the distance where the camera is operating.

$$Depth = \frac{Depth - Min\,Value}{Max\,Value - Min\,Value} \qquad (13)$$

Eq(13) makes a representation of the formula used to normalize the values between [0 - 1] where Max Value represents the greater value obtained by the depth map and the minimum value represents the lowest value in the depth map. The next algorithm makes a representation of the followed process to estimate the distant values using the depth map and the normalization process.

Input: Depth map matrix, const coefficient

Output: Preprocessed depth map

begin

(W, H) <- size(Depth map)

Min Value <- min(Depth map)

Max Value<- max(Depth map)

For i in range(H):

    For j in range(W):

        Depth map[i][j] = (Depth map[i][j] – Min Value)/(Max Value – Min Value)

        Comment: ►then the values are inverted to work with the nearest values

        Depth map [i][j] = coefficient (1.0 – Depth map[i][j])

Return Depth map

End

3.1.3.4. Backscatter removal

When Backscatter is mentioned it is necessary to know that it refers to the backward scattering component, this component affect the image, generating blur due to the distance between the object and the subject. For that reason, the methodology tries to remove this effect by subtracting the backscatter component. To make this process possible, the algorithm sums the three channel components in order to select the darker sections of the image, which are the lower values and proceed to save them in a vector, then these values are sorted in order to know the lowest values of the list in order to use them to predict the coordinates of this values in the matrix, and finally the lowest pixels estimated by the sort function are selected with its respective distance, these pixels are used as the backscatter values. These values generate a curve which is used to fit the $[Jc, \beta_C^D, Bl, \beta_C^B]$ coefficients from Eq(4), where Jc and Bl ∈ [0 – 1] and $\beta_C^D, \beta_C^B \in$ [0 - 10]. Finally, after the coefficients are predicter for every channel of the matrix the Eq(4) is used to estimate every pixel using the preprocessed depth map as a distance, and then the pixels are joined in order to generate a new three channel de – scattered image. The next algorithm makes a representation of the process [24]. To summarize the algorithm divides the depth map in 10 equal sections according to the depth values, depending of these values the algorithm will select the darkest pixels and will sort them into a vector like function shows.

Input: the three-channel image, the depth map.

Output: Direct signal three channel matrix Drgb

begin

Comment: ►Find the height and width of the image.

(H,W) <- size(img)

Comment: ►Create a vector with the size of the height with small vectors with the size of the width

Sum <- zeros(H,W)

For i in range(H):

For j in range(W):

Sum[i][j] = Img[i][j][0] + Img[i][j][1] + Img[i][j][2]

Comment: ►Make a linspace function to divide the maximun and minimun value of the depth map in 10 parts .

Scope <- (max(Depth) - min(Depth))/10

Start <- min(Depth)

EndValue <- Start + Scope

For i in range 10:

[arrayx, arrayy] <- Where(Depth >= Start , Depth < Scope)

Array_len <- len[arrayx]

Comment: ► This function will utilize the darkest values and will sort them and return their location in Two arrays, which represents the coordinates.

[Sorted_arrayx, Sorted_arrayy] <- Sum(arrayx, arrayy)

Comment: ► This function is made to take a small part of the darkest pixels of the section in order to use it in the estimation of the coefficients

M <- Floor[len * 0.01]

For i in range(M):

B_r[i] <- Img(index_arrayx, index_arrayy)[2]

B_g[i] <- Img(index_arrayx, index_arrayy)[1]

B_b[i] <- Img(index_arrayx, index_arrayy)[0]

D_map[i] <-Depth(index_arrayx, index_arrayy)

Start <- end

EndValue <- Start + Scope

End For

Comment: ► Non-linear equation used in Eq(13)

Params_r(4) = {0.0, 0.0, 0.0, 0.0}

Params_g(4) = {0.0, 0.0, 0.0, 0.0}

Params_b(4) = {0.0, 0.0, 0.0, 0.0}

function <- (b(1) * (1 – exp(1 * b(2)*x)) + (b(3) * exp(-1 * b(4)*x)))

Comment: ► The parameters are calculated fitting the B vectors and the D_map vector as if they were y and x respectively in order to find the parameters of Eq(13)

Params_r <- **lsqcurvefit**(function, B_r, D_map)

Params_g <- **lsqcurvefit**(function, B_g, D_map)

Params_b <- **lsqcurvefit**(function, B_b, D_map)

Comment: ► The backcatter coefficients are used for the parameters for every pixel of the image

Backscatter_r <- function (Params_r, D_map)

Backscatter_g <- function (Params_g, D_map)

Backscatter_b <- function (Params_b, D_map)

Comment: ► The backscatter matrix found for every channel are concatenated in one three channel matrix

Backscatter <- [Backscatter_r, Backscatter_g, Backscatter_b]

Drgb = Img – Backscatter

End

3.1.3.4.1. Curve Fitting Algorithm

After an analysis, it became apparent that the curve fitting function (lscurvefit) requires a set of pixels from the three different channels. Consequently, the algorithm takes a suboptimal efficiency in processing the time during the execution. And that's because of the number of pixels processed in a loop and that the process is repeated three times during the execution, because of the three channels of the image. Given this, it is necessary to convert the serial algorithm into an optimized parallel version in order to perform the processing time.

The next algorithm will show the optimized process utilizing the three channels as an input in order to make the parallelization process for the three channels at the same time:

lsCurvefit:

Inputs:

X: The depth which represents the distance between the object and the camera (**D)**.

Y: The coordinates represented by the backscatter values estimated in the last algorithm, in this case it utilizes the three channels, red, green and blue respectively (**B_r, B_g, B_b**).

Params: Three initialized vectors composed by four values that will represent the coefficient estimated by the formula, these parameters are different for the three channels, for that reason there are three vectors (**Params_r, Params_g, Params_b**).

Output:

Params: The three updated arrays with the estimated values that are utilized in the curve of the equation utilized in the last algorithm.

Begin

Comment: Initialize the parameters of the three new channels.

Params_r = {0.0, 0.0, 0.0, 0.0}

Params_g = {0.0, 0.0, 0.0, 0.0}

Params_b = {0.0, 0.0, 0.0, 0.0}

Comment: Configuration of adjustment.

epsilon = 1e-8 **Convergence tolerance**

stepSize = 0.001 **Step size for parameters updates**

maxIterations = 1000 **Maximum number of iterations of the loop**

Iteration = 0 **The position at which the iteration works**

Comment: Search error using the applied formula for the three channels

Preview_error_r = 0

Preview_error_g = 0

Preview_error_b = 0

Comment: the "error_Function" searches the error using the applied formula: function <- (b(1) * (1 – exp(1 * b(2)*x)) + (b(3) * exp(-1 * b(4)*x)))

Preview_error_r , Preview_error_g, Preview_error_b = **error_Function**(Params_r, Params_g, Params_b, X, B_r, B_g,B_b)

While(Iteration < maxIterations **):**

For i **in range(**Params_r.size()**):**

prevParams_r = Params_r[i]

prevParams_g = Params_g[i]

prevParams_b = Params_b[i]

current_error_r, current_error_g, current_error_b = **error_Function**(Params_r, Params_g, Params_b, X, B_r, B_g,B_b)

Comment: This conditional is repeated for all the three channels, in order to update the parameters with the newest coefficients depending of the error value.

if(current_error > Preview_error**):**

Comment: if the error worsens, revert the update

Params[i] = prevParams

Else:

Comment: if the error improves, continue updating

prevParams=current_error

End if

If (Preview_error < Epsilon**):**

Break

End if

Iteration = iteration + 1;

End While

At the same time the “error_Function” is represented in two different ways, the first one is the next algorithm:

Error_Function:

Input:

X: A vector that contains the coordinates x that represent the depth distance.

Y: A vector that contains the Y coordinates corresponding to the points in the dataset, representing the backscatter.

Output:

Error: Error value based on the parameters provided by the formula Eq(4).

Begin

Error = 0.0

Comment: Initializes the vector size using the depth array.

dataSize = size(x)

For i **in range(**dataSize**):**

    Comment: Adjustment function Eq(4): f(x)=b1*(1–exp(b2*x))+b3*exp(-b4*x)

    b1 = param[0]

    b2 = params[1]

    b3 = params[2]

    b4 = params[3]

    result =b1 * (1 – exp(b2*x[i])) + (b3*exp(-b4 * x[i]))

    error += result * result

return error

End for

However, the algorithm is repeated three times for every channel and the for is repeated until the array is over, for that reason it is possible to make a CUDA parallelization process utilizing all the X array at the same time, and after all that process make a summatory of the error to have the coefficient in order to use it with the curve fitting. Using this knowledge, the next algorithm shows the parallelized version of the algorithm:

Error_Function_optimized:

Input:

X: A vector that contains the coordinates x that represent the depth distance.

Y: Three vectors that contain the Y coordinates corresponding to the points in the dataset for the three channel values, representing the backscatter (y_r, y_g, y_b).

Datasize: The size of the vector that represent the backscatter.

Output:

Error: Error value based on the parameters provided by the formula Eq(4) for the three channel (error_r, error_g, error_b).

Locs = threadIdx.X + blockIdx.X * blockDim.X

If (Locs < Datasize**):**

Comment: Initializes the vector size using the depth array for the three channel.

b_1r, b_2r, b_3r, b_4r = params_r[0],params_g[1],params_r[2], params_r[3]

b_1g, b_2g, b_3g, b_4g=params_g[0],params_g[1],params_g[2], params_g[3]

b_1b, b_2b, b_3b, b_4b=params_b[0],params_b[1],params_b[2], params_b[3]

Comment: The equation is applied for the three functions.

result_r = b_1_r * (1 – exp(b_2_r *x[Locs])) + (b_3_r *exp(-b_4_r * x[Locs]))

error_r[Locs] = result_r * result_r

result_g = b_1_g * (1 – exp(b_2_g *x[Locs])) + (b_3_g *exp(-b_4_g * x[Locs]))

error_g[Locs] = result_g * result_g

result_b = b_1_b * (1 – exp(b_2_b *x[Locs])) + (b_3_b *exp(-b_4_b * x[Locs]))

error_b[Locs] = result_b * result_b

End if

End

To summarize the algorithm, the main objective of this process is to choose the parameters that are in proximity to the curve of the Eq(4), based on its four function values. These values are updated based on the estimated error depending of the parameter, and the aim is to find the four parameters nearest to the equation utilizing the backscatter arrays estimated. These vectors use the values of the three channels calculated in the previous algorithm, “backscatter estimation points”.

In addition to this, the parallelization process was identified through the curve fitting function. Instead of estimating the values for the three channels repeating the process for every channel separately, it computes the three channels in the same function. So the error is calculated simultaneously and in parallel. This process ensures that the three channels parameters updates are synchronized with the original values estimated by Eq(4).

3.1.3.5. Color compensation

After the backscatter value is properly removed from the three-channel photograph, it is necessary to estimate the illumination map to make a color compensation and a color cast removal so the reflectivity in the image is decreased from the last result. For this methodology, the algorithm adopts a variant of the local space average color and it utilizes a known range map [5], also known as neighborhood map. The methodology works the following way: for every utilized pixel in the image matrix, its local space average color is estimating utilizing the neighborhood map estimated before.

The algorithm works the following way:

Input: one channel of the RGB image, the backscatter values utilized, the neighborhood map, the neighborhood map numbers, "p" constant which control the locality of the illuminant map, and the constant "f" which is used to control the brightness of the illuminant map, the direct signal D estimated in the backscatter removal (D).

Output: The illumination map matrix for every channel

Comment: ►create an empty matrix with the size of the one channel part of the image and an array with the number of neighborhood constants found.

avg_cs = zeros_like(img)

avg_cs_prime = copy(avg_cs)

sizes = zeros_like(neighborhood_numbers)

Comment: ►create an empty list with the number of neighborhood numbers

Locs_list = {None} * neighborhood_numbers

Comment: ►The algorithm enters to a loop of a range of 1 to neighborhood_map values

for labelin range(1, neighborhood_numbers + 1):

    Locs_list{label - 1} = where{neighborhood map == label}

    Sizes{label - 1} = size(locs_list{label-1}{0})

Comment: ►The algorithm enters to a loop of the maximun number of iterations used in the input

For a in range(max_iters):

    For labelin range(1, neighborhood_numbers + 1):

        locs = locs_list{label - 1}

```
            size = sizes{label - 1} – 1
            avg_cs_prime{locs} = (1 / size) * (summatory(avg_cs{locs} – avg_cs{locs}))
        new_avg_cs = (D * p)+(avg_cs_prime * (1 - p))
        if maximum(abs(avg_cs – new_avg_cs) < total):
            break
        avg_cs = new_avg_cs
result = f * bilateral_filter(maximun(0, avg_cs))
return result
```

End

3.2. Time performance

Time represents an important topic in software development, because it has direct impact in optimization of the code in terms of the running time. When writing code, is normal to find situation where the execution takes much more time than the necessary, for that reason is important to compare different types of languages which take an important part of the optimization process.

3.2.1. Theoretical comparison of the execution speed

Considering what was mentioned in the last paragraph, it's important to make a comparation of the python and the C++ languages, and how working with one or another would improve the execution of the code. For that reason, this section will make a comparation between C++ and python languages.

a- Compilation vs Interpretation:

Nowadays, the programming languages are selected based on their computational model. These categorizes are utilized to simplify the communication between the program and the computer, this is because there are multiple ways to order a computer to execute an action. Therefore, some languages are capable to release faster than other languages. C++ and Python are included in this category and they are divided based on how they execute the program and how they have a different impact in their execution speed. These categories are the compilation and the interpretation respectively.

- **Compilation:** Michael Lee Scott in his book "Programming Languages Pragmatics" describes this section as "the translation of Low-level codification languages into an equivalent target program into machine language after the program is compiled in the software" [6]. This mean that the program is converted into machine language in the compilation process before executing the code. This

means that low level languages like C++ will be compiled before running the code that will translate codification into machine language before executing the program. Fig 3.4 makes a representation of the process followed by the compilation languages.

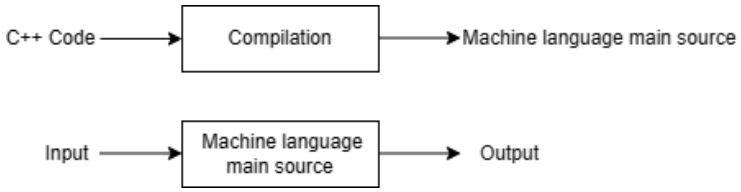


Figure 3.4: Compilation and execution representation [6]

Fig. 3.4 show the process followed by the compilation focused on C++ language. the C++ code will be compiled creating a machine language program called “main source”. Then the process will use machine language as an input code to execute the program giving as a result the output, which in this case will be the enhanced image.

- **Interpretation:** On the other hand, this process focuses on the execution of the code utilizing the source code in the main source instead of transforming it into machine language. This means that the code will be executed in a sequence until a certain condition in the program is met. The codification languages utilized in this category always reads the statement more than one time, for that reason the code is read line by line [6]. In the case of python, the function is executed until the last line is read making it slower that other languages. The next figure shows a representation of the execution of this process.

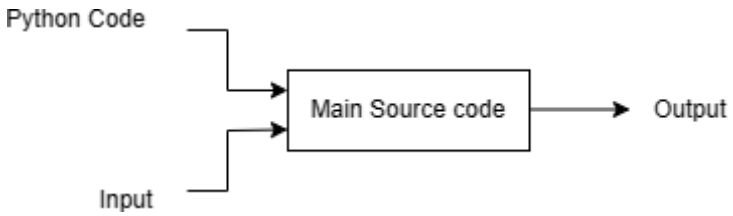


Figure 3.5: Interpretation interaction with the main source [6]

As Fig. 3.5 shows the main source written in python language will be processed in the same language by the computer instead of creating a machine language source. This means that the program will make the compilation and read the code line by line at the same time, this yields to slower execution time compared with low level languages.

b- Static typing vs Dynamic typing:

- **Static typing:** Low-level languages like C++ utilizes static overloading that helps to execute a program into the compile-time [26]. This means that the code will determine the data types of the coefficients at the compilation time. Ones this data is declared the information can't be changed during the running time.
- **Dynamic typing:** This category allows the data type to be determined while the program is being executed [26]. This category provides more flexibility and makes an easier way to write and debug the code than the static typing. In this case Pythons and other high-level languages utilize the Dynamic typing methodology, considering that this language determines the datatype while the code is running.

  However, when we talk about preformation of the running time static typing can offer a better performance, because the data is known at the compiling time. On the other hand, dynamic typing would be determined during the runtime to make

sure that the performed operations are valid for the information of the variables utilized.

c - Memory management C++:

This process refers to allocating, deallocating or managing the memory information and then free it when is no longer required [27]. This subsection has an important influence when defining which language is faster than others, because some of these allows more or less control over the memory, which helps to accelerate the process depending of the memory usage.

For low-level languages like C++, it provides full control over the memory, which means that all the information utilized could be allocated or deallocated when is not longer necessary. This process helps accelerating the execution time by defining when, where and how the information will be allocated or deallocated in the memory.

On the other hand, high level languages like python doesn't have control over the memory allocation. As a result, the memory would be freed in the end of a section [27]. The deallocation of the memory will take much more time comparing with some of the complicative languages like C++.

In other words, considering all the reasons mentioned before, it is necessary to utilize a low-level language instead a high-level language when programming the water removal code to accelerate the execution process.

3.3. Tools used for the implementation:

3.3.1. CUDA Toolkit:

CUDA toolkit is the parallel computation program or the application programming interface that was designed by NVIDIA to allow the developers to have a better control over the physical recourses of the computer [18]. This includes the GPU programing, that allows codifying in C or C++ and have a better control over the communication process over the CPU and the GPU to apply the parallelization process using kernels. This computation process yields to reducing the processing time by parallelizing the serial algorithm, using different types of techniques. In this case the parallelization method that is going to be applied is called partitioning and works with different subproblems of the images in parallel. So, the image processing time is faster than applying it in a serial algorithm. The CUDA toolkit utilized to make this project it the 12.0 version, this version provides a C++ support that is compatible with the Visual Studio version used for this project.

3.3.2. Visual Studio:

It's an integrated development environment which provides a lot of characteristics to the software development, such as the edition, the compilation, the debug and evaluate the performance of the GPU while the program is running, so it's used as a diagnosis control tool that provides the GPU performance and helps to show the space memory where the image vector is allocated int the computer. For that reason, Visual Studio is more effective to apply CUDA toolkit because it can evaluate the performance of the GPU process while the program is compiling.

3.3.3. Function Evaluation:

One of the main objectives says that the water removal process has to be done in real time. for that reason, some functions have been changed to work in the GPU instead of the CPU. These functions work with CUDA Toolkit where the kernels are used in the application of the partitioning technique.

4. Chapter 4: Implementation and results:

The last section explains about the characteristics of the hardware utilize in the implementation. The results obtained for the application of the functions explained in the methodology process and the time results utilizing a CUDA version of the code.

4.1. Hardware structure:

Hardware takes an important place when it comes to the explanation of the efficiency and the responsiveness of the program, this is important on determining its processing time. Factors such as CPU speed, memory capacity have a direct influence in the performance of the program. This section will focus on the explanation of the hardware used and the characteristic of the system used for the implementation of the process.

4.1.1. Central processing unit (CPU):

The CPU utilized for the development of the water removal project is the processor AMD Ryzen 5 5600H, produced by AMD. This model belongs to the products line of AMD Ryzen 5 Mobile Processors, used for mobile products like laptops.

- **Relevant technical specifications:**
    - **Architecture:** the CPU is composed by an AMD Ryzen Zen 3, which is characterized by its parallel processing capacity.
    - **Clock frequency:** This part explains the maximum frequency achievable by a single core on the central processing unit executing a single-threated workload on the project. In this case the CPU operates in a base frequency of 3.3GHz and can reach a maximum of 4.2 GHz in maximum boost clock of 4.2 GHz.
    - **Number of cores:** The AMD Ryzen 5 is composed by 6 CPU cores, which is capable to make multiple tasks simultaneously.

- **Support for simultaneous threads:** The processor is capable to handle 12 threads simultaneously, accelerating the processing capacity.
- **Cache:** The computer is composed by a L2 cache memory of 3MB, and a L3 memory of 16MB, which allow a faster access to data and optimizes the performance in task that require a high processing level.

Fig 4.1 makes a better representation of the characteristics utilized by the computer to make the implementation, and the optimization to perform the processing time.

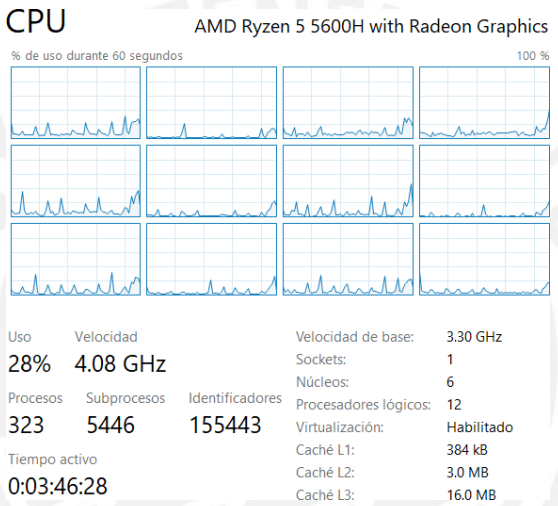


Figure 4.1: CPU AMD Ryzen 5 5600H provided information

However, this process is limited by the number of cores in the CPU, this is because the serial algorithm repeats the process in a loop for every pixel in the matrix of the image and the depth map, for that reason a serial computation is not enough to make a faster execution, and in that case its necessary to use the GPU to make a parallel computation.

4.1.2. Graphic processing unit (GPU)

The GPU used for this project is the Nvidia GeForce RTX 3050, produced by Nvidia corporation, and is designed to provide breakthrough levels of performance and efficiency.

- **Relevant technical specifications:**
    - **Architecture:** The graphic card is based on ampere architecture which is designed to accelerate many different types of computationally intensive applications and workloads. This architecture is described as the second generation of the Turing architecture but it adds different features and deliver significantly performance than Turing architecture [7].
    - **Number of CUDA cores:** The graphic card is composed by 2048 CUDA cores. These cores allow simultaneous calculations and accelerate the executions of some functions that require parallelism.
    - **Bandwidth memory:** It's composed by a bandwidth memory capacity of 192.03GB/s. this parameter represents speed at which information can be transferred from the graphic card memory.
    - **GPU global memory:** The GPU is composed by a 4GB of shared memory. This memory is utilized by the graphic card cores to store the data and facilitate the communication between the information in the parallel function.

The Fig 4.2 makes a representation of the Ampere architecture utilized in the project. It is composed by 7 graphic processing cluster, and 12 streaming multiprocessors (SM) for every cluster. Every SM will use 128 CUDA cores [7] that will be utilize to make a parallelization process in the code.

Figure 4.2: RTX3050 Ampere architecture [7]

However not every function of the code requires a parallelization, that's because is faster enough or it's because it can be implemented in the function. But in the case of the water removal process, the functions can implement a parallelization technique are the "Fill depth map", "Construction of the neighborhood map", and "Curve fitting" functions. That will be implemented in the next section.

4.2. Implementation:

4.2.1. Fill depth map:

The estimation of the erroneous information represents an important topic for the underwater restoration this part makes a research of the empty pixels of the depth map and store the number of empty pixels and the last pixel values into a different array.

- **Description of the utilized inputs:**
    - **Depth map:** For the implementation the images have one channel that work with a value of [0 - 255] pixel values. The height utilized for every image are 600 pixels and the width is 800 pixels. All images present inconsistent information represented as holes with a value of 0 in the pixel coordinates, see Fig 4.4 (a) and (d).

- **Methodology evaluation:**

The first part of the process consists in dividing the holes from the reliable values using a mask, to make this process the image enters to a loop and gives a "1" value for every pixel that is different to the hole and assign "0" to every empty pixel in the depth map. For this part I made a test with two different one channel depth map with erroneous information in this part the depth map enters to a loop where a conditional classifies the matrix depending if the matrix has pixels or its empty, Fig 4.3 shows the obtained mask of two different depth maps with erroneous information.

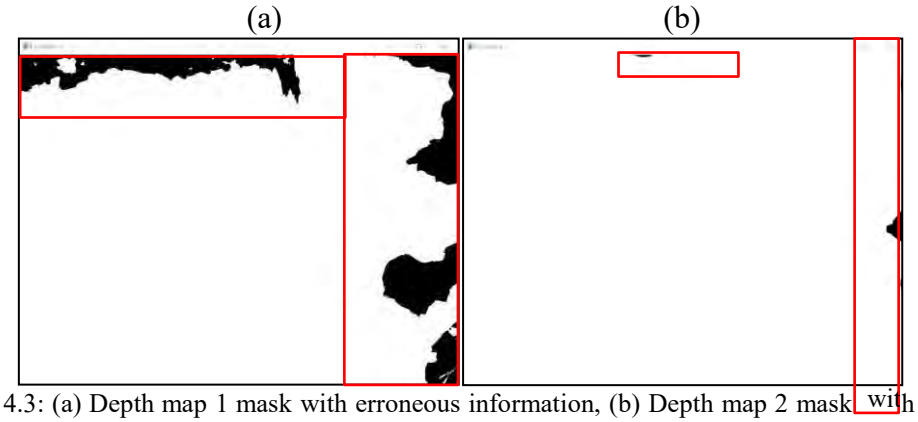


Figure 4.3: (a) Depth map 1 mask with erroneous information, (b) Depth map 2 mask with erroneous information

After obtaining this section the mask is used to detect the zero and if it's detected the code starts counting the empty pixels. But in the case of the right corners, it is possible that in some cases the depth map doesn't detect pixels, so in that cases the code will replace the 0 with the corner pixel with last value detected by the loop.

After that program will run a loop from right to left so it can detect the holes and replace the erroneous pixels with data utilizing the number of empty pixels and the last detected pixels saved in the other matrix to calculate a slope and then using it in a line function to estimate every empty of the hole.

After this part the new depth map will be used to generate a new mask, but in this case the mask will be go up to down to estimate the remaining values of the hole. The process would be the same, which means that if it detects a hole the code will start counting the values until another pixel is found, where the new values will be saved in a matrix. And finally, the code will do from down to up to fill the empty holes utilizing a new scope that will used to generate the new values of a line function which will replace the last erroneous pixels.

- **Results:**

The next figure Fig 4.4 makes the comparative result of the original depth map, the anisotropic diffuse filter and the new methodology developed, where it can be seen that the anisotropic diffuse filter filles the erroneous data but it smooths the original map. On the other hand, the new algorithm estimates the erroneous values preserving the values from the original depth map, obtaining a similar result to the original input but with the erroneous information corrected. Fig 4.4 (c) show the results obtained in this section.

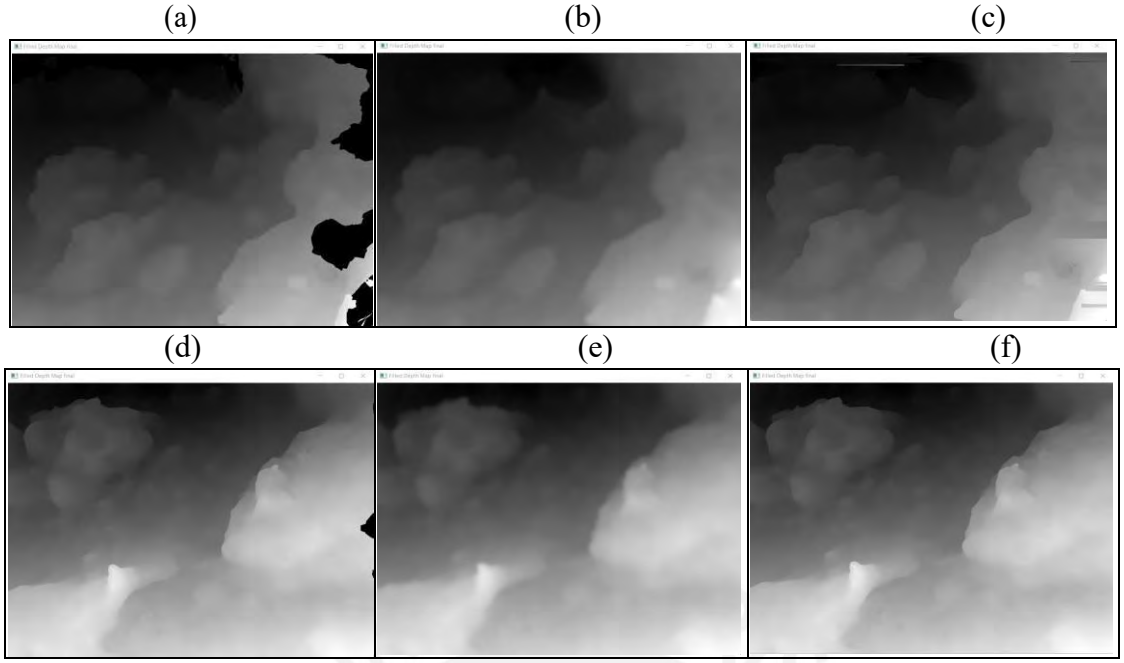


Figure 4.4: (a) Input Depth map 1 with erroneous information (b) Anisotropic diffuse filter depth map 1 methodology (c) Depth map 2 with in-painting utilizing own algorithm. (d) Input Depth map 2 with erroneous information (e) Anisotropic diffuse depth map 2 methodology (f) Depth map 2 with in-painting utilizing own algorithm.

4.2.2. Construction of the neighborhood map:

The function “Construction of the neighborhood map” was implemented with the purpose to estimate the contrast in the image so it can be used in the color compensation after the backscatter removal process.

- **Description of the utilized inputs:**
    - **Depth map:** The implementation of this process utilizes a new depth map with the erroneous pixels corrected after the execution of the “filled depth map” function. The image is composed by one channel with 600 pixels height and 800 pixels width.
    - **Epsilon:** Constant utilized to estimate the delimitations of every section of the matrix. This input is composed by a float of 0.05.
- **Methodology evaluation:**

The implementation follows the same methodology mentioned in the last chapter, in this case the output will have the same characteristics as the depth map but with empty values, and the coefficient will be initialized depending the maximum and minimum value of the depth map which was normalized and preprocessed to obtain the distance values. After that process the program locates every empty pixel in the matrix and will initialize the process sending one of these empty values into a list. Then, the function evaluates the coordinates where the pixel fell and if the depth value for that coordinate is less than the initialized epsilon coordinates the depth map will enter to a conditional where the list will be filled with the neighborhood pixels that satisfy the condition, and this process will continue until the loop doesn’t find a pixel that work for that conditions, and the process starts again until every pixel is filled. The coordinates won’t repeat because after one section is completed, the algorithm will be initialized again and will evaluate the

neighborhood matrix, but in this case, it sees that the section is already completed, so the filled section won't be taken again.

- **Results:**

Upon executing the "Construct neighborhood map" in C++. The code provides Fig 4.5 (b) where it shows that the algorithm divides the contrast into a different number of sections which varies depending of the image. In this specific case the program gives 14 neighborhood values that represents the number of sections estimated. The Fig 4.5 also makes a comparative result of the neighborhood map output, and there can be seen that the colors are different in (b) than (c), however the neighborhood values remain as the same in both cases.

(a) (b) (c)

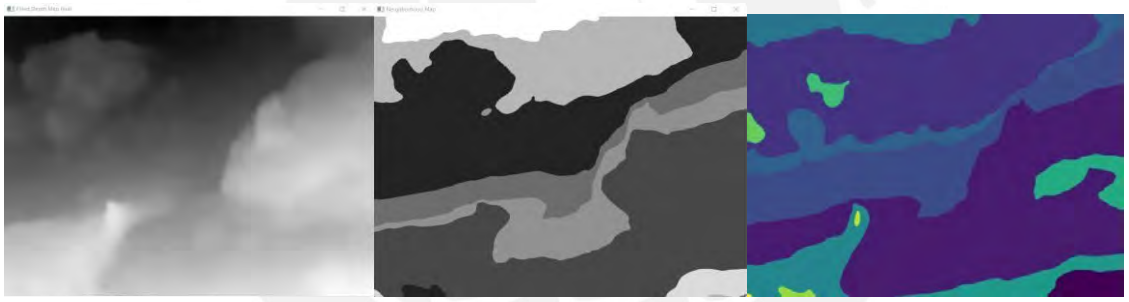


Figure 4.5: (a) depth map input utilized for this process. (b) Neighborhood map implemented in C++ (c) Neighborhood map implemented in python

4.2.3. Backscatter removal:

The function utilized “Backscatter removal” was implemented to subtract the effect produced by the reflectivity of the particles in the scene. To make this process possible the program will estimate the backscatter curve utilizing the values provided by the image.

- **Description of the utilized inputs:**
    - **Depth map:** Preprocessed depth map, in this case the values between [0 - 255] pixels were normalized into [0 - 1]and then inverted and multiplied for 10 to convert the values into a range that is used as a distance. Composed by one channel and 600 pixels of height and 800 pixels of width.
    - **Image:** The three-channel image normalized to the range [0 - 1] with 600 pixels of height and 800 pixels of width.
- **Methodology evaluation:**

The methodology follows the algorithm mentioned in the in the last chapter, in this case the code initializes an empty matrix that will save the addition of all the pixels from the three channels in the image, this process helps to identify the darkest pixels in the photograph. The algorithm sorts the darkest values of the 10 different sections of the image which depends of the distance estimated by the depth map. Then the program sorts this value and saves it into three different arrays which represent the backscatter estimation points, also the depth map for that coordinates were saved into a different array, so it will be possible to estimate the backscatter values from the image and the saved depth map helps to identify the position of the pixel in the space Fig 4.6 shows a representation of the backscatter values estimated versus the distance between the object

and the camera of the three channel of the image, in this case the three arrays are composed by 2771 pixels, giving as a result the next image.

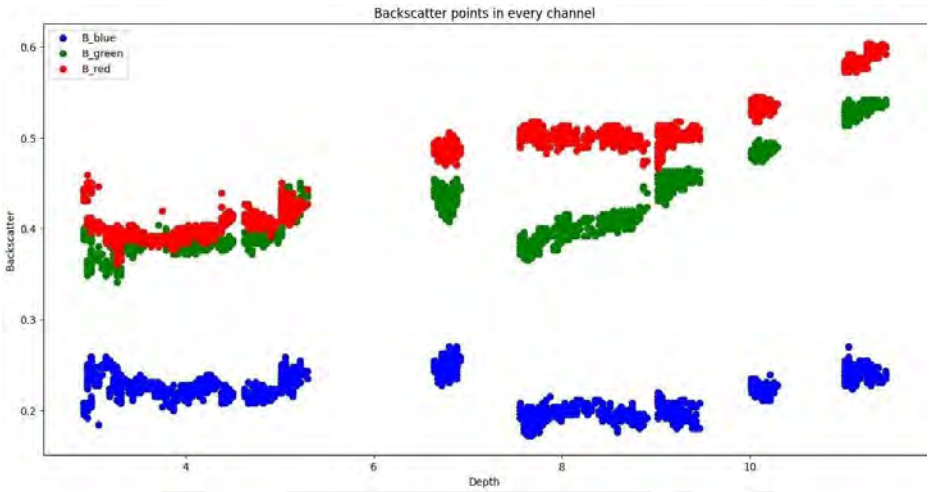


Figure 4.6: Backscatter points estimated vs the depth distance

However, after this process it was necessary to estimate the $[Jc, \beta_C^D, Bl, \beta_C^B]$ from Eq.4, so it was necessary to use a curve fitting function based on minimum quadratic values, that tries to make a linear approximation of the parameters utilizing the backscatter and depth points, but in this case, it's applied for a nonlinear equation where the functions tries to adjust the curve to the provided information, returning all the parameters for that points. Fig 4.7 shows the curve fitting result for the Eq.4 with the estimated points using Table II parameters to make the curve.

Table II: $[Jc, \beta_C^D, Bl, \beta_C^B]$ values estimated for the image utilized.

	$\boldsymbol{Jc}$	Q_C^D	$\boldsymbol{Bl}$	Q_C^B
Channel R	0.01	0.08	0.229	0.005
Channel G	0.001	0.001	0.432	0.001
Channel B	0.001	0.001	0.483	0.001

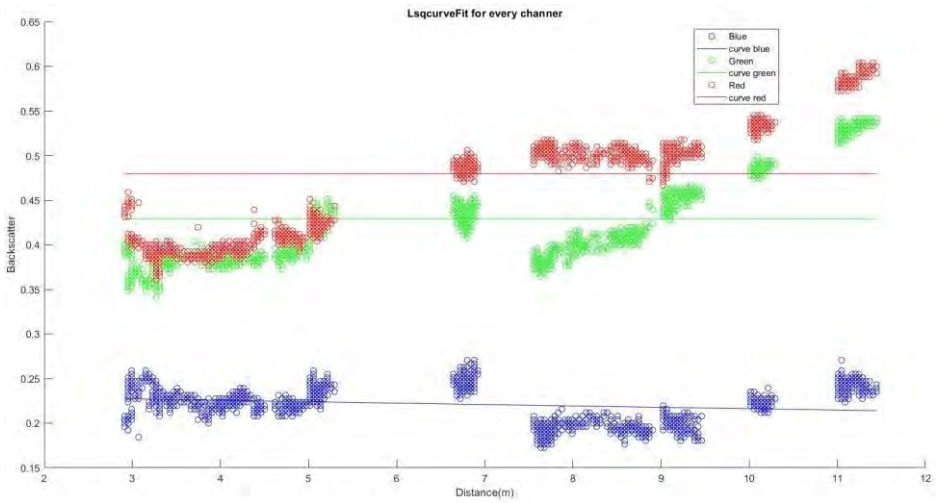


Figure 4.7: Estimated points vs the distance and the channel curves using the parameters from table II

Finally, after the parameters were estimated the program use them with the depth pixels to estimate the backscatter matrix. After this process, the three channels of the backscatter are subtracter from the original image giving as a result Fig 4.8.

Figure 4.8: De-scattered image

Fig 4.8 shows to be darker than the original image because pixels values were subtracted from the original photograph, to solve this problem the brightness was adjusted. Finally, Fig 4.9 shows a comparation of original image with the de-scattered version, and it shows that the smooth effect was removed from the photograph and some of the colors were recovered.

(a) (b)

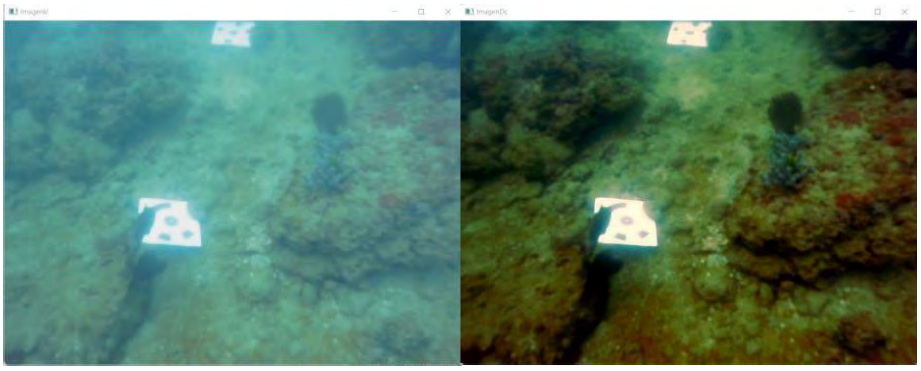


Figure 4.9: (a) original image (b) de-scattered image with a readjustment of the brightness.

4.3. Time:

In order to optimize the performance in terms of the running time, an evaluation was made to evaluate execution comparing C++ with python and based on this comparison the slowest function was parallelized. Table III shows the comparative results of the execution process for an image composed by a height and width of 600 x 800 respectively, the table also shows the slowest section of the code that in this case is the backscatter values specially in a function called “curve fitting”, the functions take 627 ms to run for each channel, which means that the function runs three times for every channel of the image.

Table III: Comparative execution time of all the functions implemented

	Python	C++	CUDA (Naive version)	CUDA (improved)	Threads
In paint depth map	17.57 s	4 ms	-	-	-
Construction of the neighborhood map	2.69 s	75 ms	-	-	-
Backscatter estimation points	0.43 s	66 ms	-	-	-
Backscatter values	1.145 s	2.841 s	0.751 s	0.591s	1.122s

In this case the slowest function is “curve fitting”, due to its computational complexity, it was considered that the function enters to a loop where it evaluates the backscatter values and the distance between these parameters which are saved as arrays, for that reason the

computational complexity estimated for this function is O(n). In the case that the computer is only using one processor p = 1 the obtained value remains as O(l x n) which in the worst of the cases the functions remain as O(n). In this specific case that the algorithm works with CUDA it is necessary to mention that the number of processors influence significantly in the time performance. But in this case the function repeats until it reaches the convergence or it reaches the maximum number of iterations which means that it varies depending the information provided by the arrays.

The CUDA functions are a Naive Version and a Parallel Version of the serial code, this means that the two different methodologies were evaluated for the same process to demonstrate the execution time with the provided information. The acceleration was calculated dividing the execution time of the serial sequence with the execution time of the not optimized and the optimized parallel sequence.

Based on the execution time estimated before it is possible to find:

- Serial time = 2.841 s (Time for executing the Serial Version for the three channels in the image)
- Naive Parallel Time = 0.751 s (Time for executing the Naive Version in parallel for the three channels)
- Optimized Parallel Time = 591ms (Time for executing the Optimized Version in parallel for the three channels)
- Acceleration = 2.841s / 0.591s= 4.807

Which means that the non-optimized version of the code is 4.807 times faster than the serial version.

4.4. Conclusions:

- For water removal, the original interpreted code was parallelized, enhanced, and ported into a compile code allowing to significantly reduce the running time.
- It is necessary to write the code in a compiled language instead of an interpreted language to improve the running time of the code. This process shows to take less time while executing the program, and provides more control over the memory management than python, which helps in the communication between the C++ and the CUDA files while using headers.
- It's important to correct the erroneous information provided by the depth map because the distance can abruptly change from one value to "0" due to the empty pixels of the matrix. These errors affect the preprocessing of the depth map, which means that the distance won't be accurate to the depth data and yields to an erroneous estimation of the backscatter parameters. To solve this error a new algorithm was proposed to predict the erroneous values utilizing different arrays to store the hole size and the pixels values, which are then used to estimate the empty values.
- To solve the degradation, effect a small group of the backscatter pixels were selected to estimate the backscatter curve. This curve is used in the estimation of the parameters which are then used in the rest of the matrix.
- All the experimental results shows that the water enhancement function can work properly for underwater with a strong degradation, especially for those with a robust reduction of the red channel.
- The slowest section of the code shows to be the "curve fitting" function utilized in the estimation of the backscatter parameters, for that reason a Naïve Version of the parallelized process was made in CUDA to reduce the processing time which

shows to accelerate 4.807 times the process which now takes less time to be executed due to the implementation in a GPU.

5. References:

[1] P. Zhuang, J. Wu, F. Porikli, and C. Li, “Underwater Image Enhancement With Hyper-Laplacian Reflectance Priors,” *IEEE Transactions on Image Processing*, vol. 31, pp. 5442–5455, 2022, doi: 10.1109/TIP.2022.3196546.

[2] J. Y. Chiang and Y. C. Chen, “Underwater image enhancement by wavelength compensation and dehazing,” *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1756–1769, Apr. 2012, doi: 10.1109/TIP.2011.2179666.

[3] H. Lu, Y. Li, L. Zhang, and S. Serikawa, “Contrast enhancement for images in turbid water,” *Journal of the Optical Society of America A*, vol. 32, no. 5, p. 886, May 2015, doi: 10.1364/josaa.32.000886.

[4] C. O. Ancuti, C. Ancuti, C. de Vleeschouwer, and P. Bekaert, “Color Balance and Fusion for Underwater Image Enhancement,” *IEEE Transactions on Image Processing*, vol. 27, no. 1, pp. 379–393, Jan. 2018, doi: 10.1109/TIP.2017.2759252.

[5] D. Akkaynak and T. Treibitz, “Sea-thru: A Method For Removing Water From Underwater Images,” Jun. 2019. doi: 10.1109/CVPR.2019.00178.

[6] M.L. Scott, “Programming Language Pragmatics”, Morgan Kauffman, 2000.

[7] Nvidia, “Updated with NVIDIA RTX A6000 and NVIDIA A40 Information V2.0 NVIDIA AMPERE GA102 GPU ARCHITECTURE Second-Generation RTX NVIDIA Ampere GA102 GPU Architecture.”

[8] E. Park and J. Y. Sim, “Underwater image restoration using geodesic color distance and complete image formation model,” *IEEE Access*, vol. 8, pp. 157918–157930, 2020, doi: 10.1109/ACCESS.2020.3019767.

[9] Q. Yang, R. Yang, J. Davis, and D. Nistér, “Spatial-Depth Super Resolution for Range Images.” [Online]. Available: http://vis.uky.edu/˜liiton/

[10] D. Min, J. Lu, and M. N. Do, “Depth video enhancement based on weighted mode filtering,” *IEEE Transactions on Image Processing*, vol. 21, no. 3, pp. 1176–1190, Mar. 2012, doi: 10.1109/TIP.2011.2163164.

[11] A. A. Khoddami, P. Moallem, and M. Kazemi, “Depth Map Super Resolution Using Structure-Preserving Guided Filtering,” *IEEE Sens J*, Jul. 2022, doi: 10.1109/JSEN.2022.3176669.

[12] D. Li, B. Lin, X. Wang, and Z. Guo, “High-Performance Polarization Remote Sensing With the Modified U-Net Based Deep-Learning Network,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, 2022, doi: 10.1109/TGRS.2022.3164917.

[13] J. Qian, J. Li, Y. Wang, J. Liu, J. Wang, and D. Zheng, “Underwater image recovery method based on hyperspectral polarization imaging,” *Opt Commun*, vol. 484, Apr. 2021, doi: 10.1016/j.optcom.2020.126691.

[14] F. Cozman and E. Krotkov, “Depth from Scattering,” Jun. 1997. doi: 10.1109/CVPR.1997.609419.

[15] C. Tsiotsios, M. E. Angelopoulou, T. K. Kim, and A. J. Davison, “Backscatter compensated photometric stereo with 3 sources,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Sep. 2014, pp. 2259–2266. doi: 10.1109/CVPR.2014.289.

[16] M. Martin, Nishchol Mishra, S. Sharma, and Gunjan Pandey, “‘UD-ETR Based Restoration & CNN Approach for Underwater Object Detection from Multimedia Data,’” Feb. 2020. doi: 10.1109/IDEA49133.2020.9170740.

[17] W. Zhang, P. Zhuang, H. H. Sun, G. Li, S. Kwong, and C. Li, “Underwater Image Enhancement via Minimal Color Loss and Locally Adaptive Contrast Enhancement,” *IEEE Transactions on Image Processing*, vol. 31, pp. 3997–4010, 2022, doi: 10.1109/TIP.2022.3177129.

[18] NVIDIA Corporation, “CUDA C++ Programming Guide Design Guide,” Nov. 2022.

[19] J. JaJa, *An introduction to parallel algorithms*. Addison-Wesley Publishing Company, Inc, 1992.

[20] A. Díaz, E. Caicedo, L. Paz, and P. Piniés, “Depth map denoising and inpainting using object shape priors,” *Computacion y Sistemas*, vol. 24, no. 1, pp. 221–239, 2020, doi: 10.13053/CyS-24-1-3004.

[21] D. Ferstl, C. Reinbacher, R. Ranftl, M. Ruether, and H. Bischof, “Image guided depth upsampling using anisotropic total generalized variation,” in *Proceedings of the IEEE International Conference on Computer Vision*, Institute of Electrical and Electronics Engineers Inc., 2013, pp. 993–1000. doi: 10.1109/ICCV.2013.127.

[22] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images,” 2012.

[23] D. Akkaynak, T. Treibitz, T. Shlesinger, R. Tamir, Y. Loya, and D. Iluz, “What Is the Space of Attenuation Coefficients in Underwater Computer Vision?”

[24] J. Zhou, T. Yang, W. Chu, and W. Zhang, “Underwater image restoration via backscatter pixel prior and color compensation,” *Eng Appl Artif Intell*, vol. 111, May 2022, doi: 10.1016/j.engappai.2022.104785.

[25] D. Slater and G. Healey, “What is the spectral dimensionality of illumination functions in outdoor scenes?,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Comp Soc, 1998, pp. 105–110. doi: 10.1109/cvpr.1998.698595.

[26] R. T. Saunders and C. Jeffery, “Fostering Usability of Dynamic, Recursive, Heterogeneous Dictionaries in Statically-Typed Languages.” [Online]. Available: http://www.picklingtools.com.

[27] F. Zehra, D. 2, M. J. 3, and M. Pasha, “Comparative Analysis of C++ and Python in Terms of Memory and Time,” 2020, doi: 10.20944/preprints202012.0516.v1.

6. Annexes:

- Source code: https://github.com/sebastiantorres1/Water-enhancement-project.git