PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

OVERLAPPING POINT CLOUD MERGE AND SURFACE RECONSTRUCTION WITH PARALLEL PROCESSING FOR REAL TIME APPLICATION

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR:

Pierre Ramiro Pérez Ramírez

ASESOR:

Cesar Alberto Carranza De La Cruz, PhD.

Lima, Marzo, 2023

Informe de Similitud

Yo, Cesar Alberto Carranza De La Cruz, docente del Departamento de Ingeniería de la Pontificia Universidad Católica del Perú, asesor de la tesis titulada OVERLAPPING POINT CLOUD MERGE AND SURFACE RECONSTRUCTION WITH PARALLEL PROCESSING FOR REAL TIME APPLICATION, del autor Pierre Ramiro Pérez Ramírez dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 8%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 22/04/2023.
- 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, 22 de Abril del 2023

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

Resumen

Compañías mineras están en búsqueda constante de nuevas tecnologías para aumentar su productividad. Una de las tecnologías que les permite realizar la reconstrucción de la superficie sin poner en riesgo la vida de sus trabajadores es el uso de sensores LiDAR junto con plataformas móviles que les permiten rotar el sensor para realizar un escaneo completo de la estructura. Sin embargo, el procesamiento de los datos se realiza a través de ordenadores situados fuera de la mina, debido a su alto coste computacional, lo que se traduce en un alto coste de tiempo.

En esta tesis presento como objetivo principal el diseño de un algoritmo paralelo para la fusión de nubes de puntos capturadas por un LiDAR y la reconstrucción de la superficie en tiempo real, con el fin de reducir el tiempo de procesado, teniendo en cuenta información a priori del patrón de barrido de los puntos. En la literatura se pueden encontrar algoritmos para la reducción de la densidad de puntos, sin embargo, en esta tesis, propongo la idea de suprimir estos puntos basándome en el principio de que la etapa de registro entre cada escaneo puede ser obtenida por un sistema de medición correctamente establecido, por lo tanto, no es necesario utilizar ningún algoritmo ICP. Además, a diferencia de los algoritmos genéricos de reconstrucción de superficies, propongo un nuevo algoritmo que utiliza la información a priori del sistema de escaneo que permite obtener la reconstrucción triangular en un tiempo menor al tiempo de escaneo del LiDAR. Este algoritmo se implementará en un ordenador *desktop* con el uso de GPUs proporcionadas por NVIDIA para evaluar su rendimiento y, también, se implementará en una Jetson Nano con datos de una mina socavón real. Finalmente, proporcionaré algunas recomendaciones y consideraciones a tener en cuenta en las etapas de evaluación del algoritmo secuencial, codificación del algoritmo paralelo e implementación en GPUs.

Abstract

Mining companies are constantly searching for new technologies in order to increase their productivity. One of the technologies that allow them to perform surface reconstruction without risking the lives of their workers is the use of LiDAR sensors in conjunction with mobile platforms that allow them to rotate the sensor to perform a full scan of the structure. However, the data processing is done through computers located outside the mine, due to its high computational cost, resulting in a high cost of time.

This thesis presents as principal objective the design of a parallel algorithm for the fusion of point clouds captured by a LiDAR and the surface reconstruction in real-time, in order to reduce the time processing, taking into account a priori information of the scanning pattern of the points. Algorithms for point density reduction can be found in the literature, however, in this thesis these points are suppressed based on the principle that the registration stage between each scan can be obtained by a measurement system properly stablished, therefore, it is not necessary to use any ICP algorithm. Also, unlike the generic surface reconstruction algorithms, a new algorithm that uses the a priori information of the scanning system is proposed and allows to obtain the triangular mesh in real-time in comparison to the LiDAR scanning time. This algorithm will be implemented in a desktop computer with the use of GPUs provided by NVIDIA to evaluate its performance and, also, will be implemented in a Jetson Nano with real data. Finally, some recommendations and considerations are provided to be taken into account in the stages of evaluation of the sequential algorithm, coding of the parallel algorithm and implementation on GPUs.

Dedicado a mis amistades del pasado,

las del presente, a mi padre, a mi madre,

a mi hermano, a la incondicional MIA

y para esas personas que siempre

me hacen recordar que

"tenés para MÁS"

Index

Figure Index

Glossary of Acronyms and Abbreviation

CREW	:	Concurrent Reading Exclusive Writing
CRCW	:	Concurrent Reading Concurrent Writing
CUDA	:	Compute Unified Device Architecture
CPU	:	Central Processing Unit
CCD	:	Charge Couple Device
EREW	:	Exclusive Reading Exclusive Writing
FOV	:	Field of view
GB	:	Gigabyte
ICP	:	Iterative Closest Point
ORaMG	:	Overlap Removing and Mesh Generation
PRAM	:	Parallel Random Access Machine
RAM	:	Random Access Machine
SM	:	Streaming Multiprocessor
SFU	:	Special function unit

Introduction

Those points obtained from a three-dimensional scan of an object are called a point cloud. This term refers to the absence of connection between the points that give the impression of floating in space. Among the most popular techniques for obtaining a point cloud is laser triangulation, which consists of the emission of a laser towards a mirror that then bounces off the object and is captured by a camera to then calculate the distance. The most popular sensors using this type of technology are called LiDAR. Also, each generated point cloud may present measurement errors, and to perform proper data processing it is necessary that the cloud goes through a filtering stage that eliminates these errors. On the other hand, to achieve a total representation of the object it is necessary to perform scans from different angles of the object and then transform the clouds obtained to the same referential coordinate system, this stage of cloud transformation is called registration. Finally, in most applications, surface reconstruction is performed, which is done by means of algorithms that connect the cloud points forming polygons, which are generally triangles. Currently, in mining works the scanning of structures is performed, for this purpose, by LiDAR sensors. In mining operations is highly important to obtain relevant information of the structures in real time. However, to perform the information processing, such as surface reconstruction or volume calculation, the data is extracted to an external computer of the mine due to its high computational cost, which leads to a high cost of time. In this thesis, a new algorithm is proposed to solved the problems presented, using parallel processing and an embedded system of low power consumption.

1 Chapter 1: Surface reconstruction of a point cloud data

Technology of 3D reconstruction has various applications in the fields of geological exploration, industrial product testing, biomedicine, aerospace, among others. This 3D reconstruction can be summarized in the following steps: First, a scanning process of the scene or object; for this process, exists various methods which affects or not the object or scene [1]. Second, the data obtained need to be processed to be reconstructed; noise filtering, registration, XYZ transformation, etc. Finally, a rendering could be done to represent the data in a mesh structure [2].

1.1 State of the art

1.2.1 Scan by laser triangulation.

Among the most popular techniques for object scanning is laser triangulation scanning [2]. This type of measurement does not have contact with the object and is a type of optical measurement, since it depends on a light source, laser beam. It consists of the emission of a laser towards a mirror which then bounces off the object and is captured by a CCD (charge coupled device) camera [3], see Figure 1.

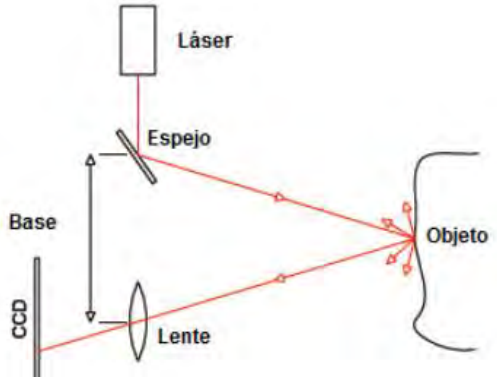


Figure 1. Laser triangulation measurement [2]

To obtain the position of the object, the following data are considered: distance between the mirror and the camera; the camera vertex angle when it receives the bounced beam; and the vertex angle of the mirror. There are other additional parameters such as the time of flight of the laser and its energy; if these data are considered, it is possible to obtain more information about the object such as the type of material and the color [1]. The most popular sensors that use this type of technology are LiDAR (Light Detection and Ranging), which have been used for the reconstruction of cities, cultural heritage, forests, among others [4]. In contrast to photogrammetry, obtaining the point cloud by LiDAR sensors is more accurate and inexpensive. [5]. In Figure 2, the point cloud obtained from a LiDAR-aircraft system is observed and in Figure 3 the result of the its surface reconstruction is observed. In [5], the LiDAR information is obtained and then processed to obtain finer contours and finally rendered to obtain the three-dimensional surface reconstruction..

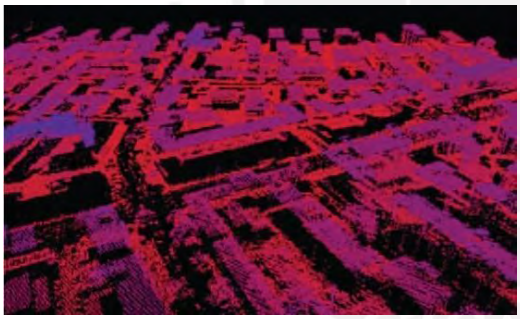


Figure 2. Point cloud of an urban city [5]

Figure 3. Surface reconstruction of an urban city [5]

1.2.2 Data of a LiDAR sensor.

From a single LiDAR scan, a large number of points associated to a local reference system are obtained, this is called point cloud because of the absence of connection between the points that give the impression of floating in the space [2]. See Figure 4, for an example of a point cloud.

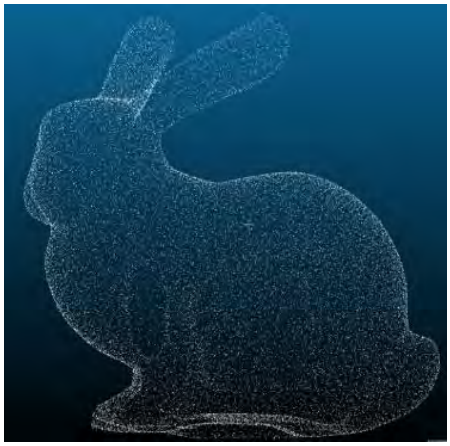


Figure 4. Stanford bunny point cloud [6]

However, to obtain a three-dimensional reconstruction, in some cases it is necessary that the LiDAR sensor does additional scans from different strategic stations (sensor positions) to obtain the necessary information for a virtual model. [7]

1.2.3 Filter stage.

Each generated point cloud can have measurement errors, i.e., there are points in the cloud, considered as noise, which are outside the contour of the real structure. [6]. To eliminate the noise, it is necessary that the point cloud goes through a filtering stage. There are several filtering methods and can be categorized into the following groups: by statistics, by neighborhood points, by projections and by PDEs. [6].

The cause of measurement noise can be due to various factors depending on the scenario. But the most common case for LiDAR sensors is the reflection generated by the type of material. In [1] The reconstruction of a pot lid was obtained, see Figure 5, but a filtering stage to eliminate the errors generated was not considered. Another possible noise factor is not considering the movements that the sensor may have, as it could happen if it is moving with an autonomous terrestrial robot.

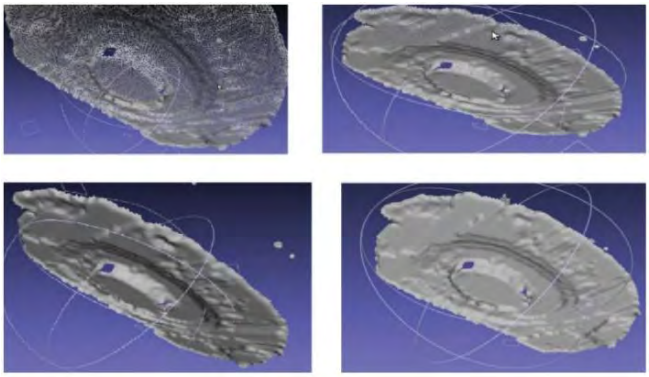


Figure 5. reconstruction of a pot lid [1]

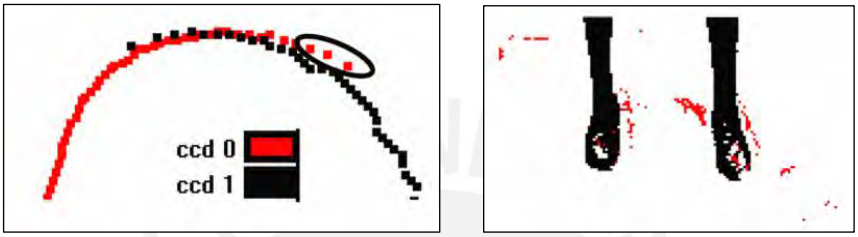


Figure 6. Noisy points detection by contour analysis [8]

On the other hand, in [8] the analysis of which points generated noise was carried out by means of different scans to the contour of the real object, see Figure 6. In the right, it can be seen that the red points are those that generate noise to the point cloud. This type of denoising is performed with a priori knowledge of the real object. Another method is denoising by polygon, which is what was done in [9], see Figure 7, in (a) the point cloud is observed without filtering, and in (b) the unnecessary points are suppressed, limiting the space to be analyzed.

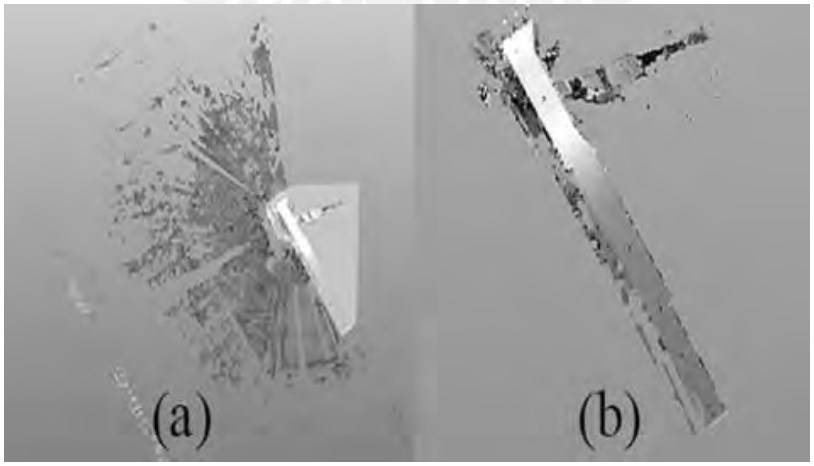


Figure 7. Point Cloud of a building [9]

1.2.4 Registration of a point cloud.

The process of transforming point clouds to a global reference system is called registration [10]. When the parameters of the global reference system are unknown, it is necessary to establish a point cloud as a reference and then concatenate the other point clouds until a matching between them is achieved. The most popular algorithm for this type of scenario is the iterative closest point (ICP) [2][4]. Figure 8 shows two point clouds from different perspectives, and Figure 9, shows the alignment by means of an ICP-based algorithm [11].

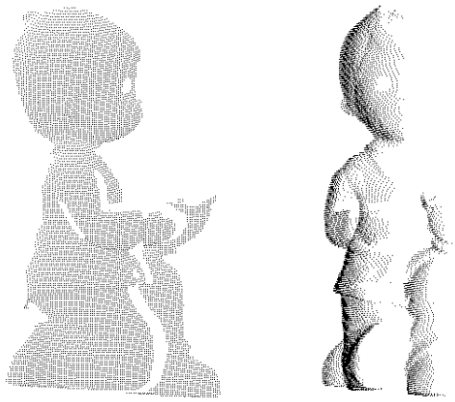


Figure 8. Point clouds of a child [11]

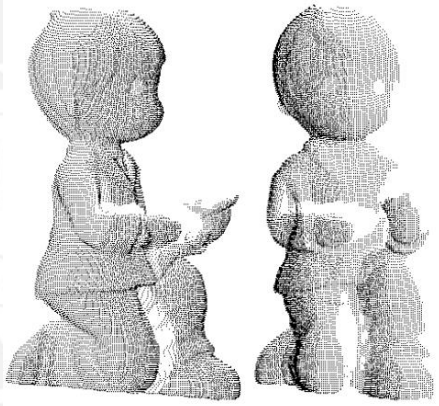


Figure 9. Point cloud registration of Figure 8[11]

For instance, in the case of [9] the map of the structure was available a priori and stations were established for the position of the Trimble LiDAR sensor, model TX5, see Figure 10. In this way a global reference system was defined for all the stations and the result of the concatenation of the point clouds is shown in Figures 11 and 12.

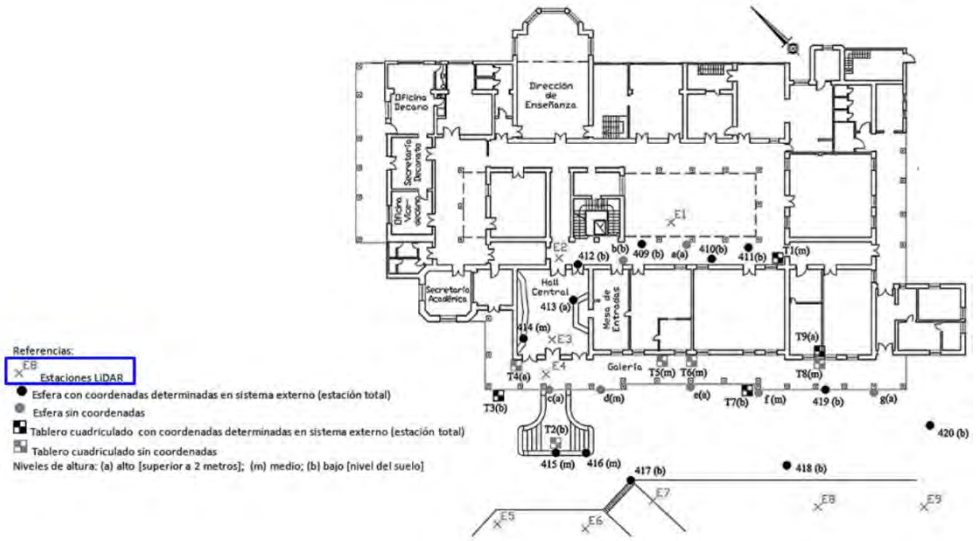


Figure 10. Map of the central building of the School of Engineering of UNLP

Figure 11. Reconstruction of the central building of the Faculty of Engineering of the UNLP [9]

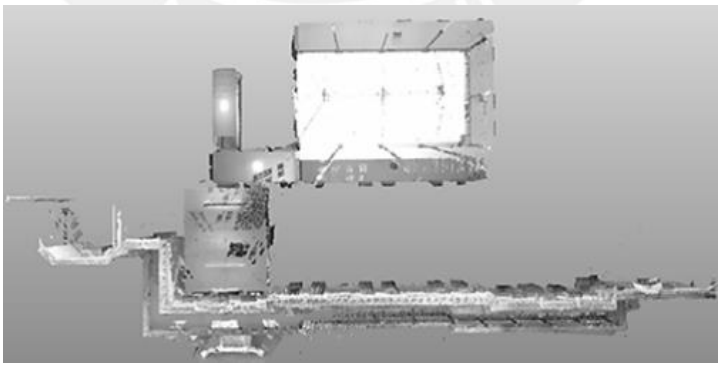
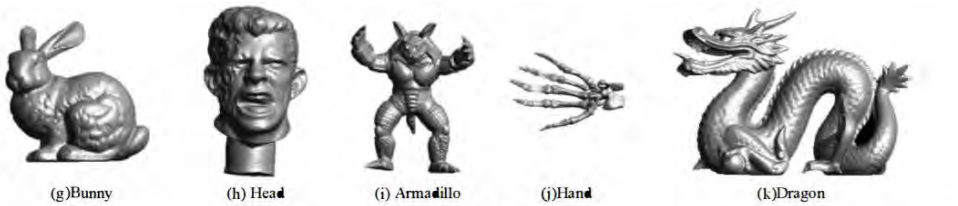


Figure 12. Plan view of the central building of the UNLP School of Engineering [9]

1.2.5 Rendering of the point cloud.

The last process of 3D reconstruction is rendering. Here you can define the color, texture, type of material and geometric shape of the structure. For the geometry of the structure, the points of the cloud must be connected generating polygons. This connection generates a polygonal mesh containing the vertices of each polygon, in general, the mesh is of triangular type. To solve this point connection problem, algorithms based on combinatorial structures (Delaunay triangulation, Voronoi diagrams and alpha forms), or approximations on implicit functions such as Poisson reconstruction, are often used. [2][12]

The most commonly used algorithm for this process is the Delaunay triangulation, some results of this algorithm have already been presented in the Figures 3 and 5. However, Delaunay triangulation process requires a certain time in the order of hundreds of seconds is the proper hardware is not used, which makes it unfeasible for real time scenarios [6]. For this reason, optimal proposals are developed to obtain a rapid reconstruction of the object. In [13] a Delaunay-based algorithm is proposed which, unlike the traditional one, only requires a Delaunay calculation and does not need Voronoi information for the manifold extraction, which makes it robust and efficient. In the following figure some results from a dataset can be observed and a table that records the execution times of both the Delaunay calculation and the manifold extraction.



Result of reconstruction

EXPERIMENT DATA OF SEVERAL SURFACE RECONSTRUCTIONS

Model	Points	Primary Triangles	Output Triangles	Delaunay time(sec.)	Manifold Extraction time(sec.)	Total time(sec.)
Bunny	34,843	481,470	69,635	1.063	1.672	2.735
Head	64,724	901,166	129,333	1.828	3.297	5.125
Armadillo	172,982	2,395,394	345,943	6.469	9.437	15.906
Hand	327,331	4,601,526	654,630	25.281	16.625	41.906
Dragon	435,553	6,296,418	868,235	21.609	26.25	47.859

Figure 13. Results obtained in [13] *about running times for mesh generation*

1.2 Problem Definition

For mining operations, it is necessary to perform the surface reconstruction of the structures, for this, there is a LiDAR-engine system that allows us to obtain the total scanning of the scene. For instance, with the OS1-gen1-16 LiDAR sensor, a 360° scan can be performed with an azimuth elevation angle between +16.9° and -16.6°; the point cloud obtained is called Donut, see Figure 14.

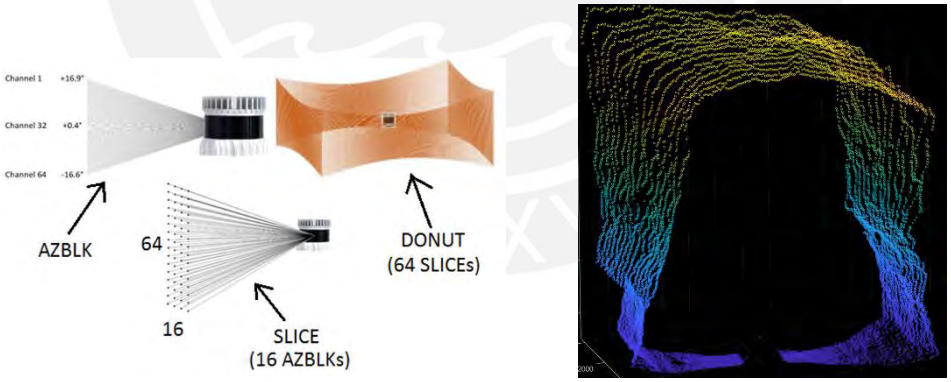


Figure 14. LiDAR OS1-gen1 scan

It was observed that a single scan does not allow to obtain the total information of the space. For this reason, there is a motor that allows the rotation of the LiDAR to scan other zones, this configuration for the motor may vary with different models of LiDAR. For instance,

with the OS1-gen-16 LiDAR, the z-axis is positioned horizontally and the motor will rotate the sensor on the new vertical axis, the x-axis of the LiDAR, see Figure 15. For this case, in order to obtain a total scan of the scene at least 6 Donuts must be generated.

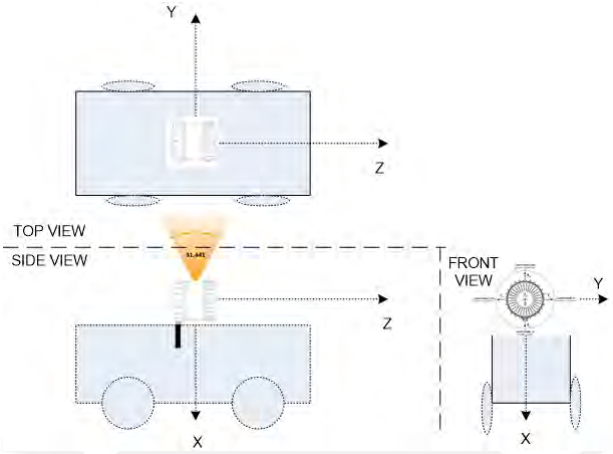


Figure 15. Measurement systema LiDAR-motor

In the processing stage of the rotation, it was observed that there is a large amount of overlapping point, i.e., there is redundant information from the scan, see Figure 16. In addition, the reconstruction process involves a high computational cost that cannot be performed by the autonomous LiDAR-motor system and the data must be extracted to an external computer to process the information.

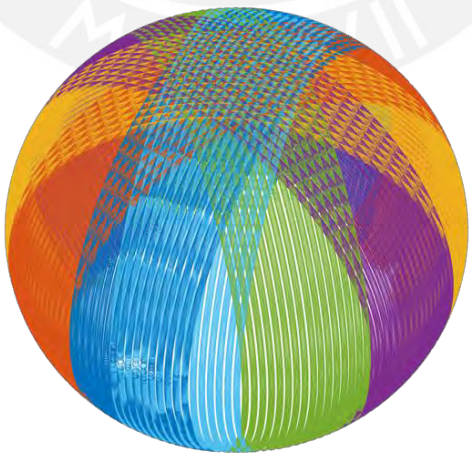


Figure 16. Points cloud with overlapping points

Therefore, it is expected that the three-dimensional reconstruction time of this system can be reduced if redundant information is eliminated. In addition, it is possible to perform previous calculations considering a priori the parameters of the measurement system to perform real-time processing. For this reason, this thesis proposes an algorithm that performs the calculation of the triangular mesh for the three-dimensional reconstruction, by means of a mathematical model that represents the Donuts generated by the LiDAR without overlapping, considering the measurement parameters of the system. In this way, this will reduce the computational cost compared to the generic algorithms that generate the triangular mesh. Finally, the triangular mesh will be presented and the point cloud fusion algorithm will be parallelized in order to obtain real-time results

1.3 Objectives

1.4.1 Main objective.

Design a parallel algorithm for the fusion of point clouds captured by a LiDAR taking into account a priori information of the scanning pattern of the points.

1.4.2 Specific objectives.

1. Mathematically model the point cloud overlap of a standard Lidar scan.
2. Design the sequential algorithm for merging overlapping sections of a point cloud.
3. Parallelize the algorithm to achieve real-time processing.
4. Implement the parallel algorithm on a GPU and evaluate its performance in terms of execution time and memory required.

2 Chapter 2: Overview of LiDAR-motor system measurement, surface reconstructions and basics of parallel algorithms

In the previous chapter, the problem of the computational cost of redundant point cloud processing was presented, and in order to obtain a real-time processing it is necessary to know some theoretical fundamentals about point cloud fusion, surface rendering and parallel processing. Also, to explain the operation of a 360° LiDAR sensor on which the solution of this thesis project will be focused and to define the time range to be considered a real-time processing.

2.1 Principles of point cloud merging

In general, the merging of two point clouds can be called point cloud registration [10]. As explained in the previous chapter, the registration process is the transformation of point cloud coordinates to the same coordinate system. However, in this case the point cloud registration is already done but the reduction or remotion of redundant points needs to be solved.

Redundant points will only be present in certain areas of the cloud; therefore, these areas have a higher point density. To reduce this density there are several techniques such as: Simplification based on normal eigenvalues [14], or based on growing neural gas [15]; and downsampling based on the weighted average of the farthest point [16], or based on Tensor Voting [17]. However, the solution proposed is to remove these redundant points, to accomplish this the measurement parameters of the LiDAR-engine system must be considered.

2.2 Definitions of the scanning method of the LiDAR-motor system

The solution of the thesis will be focused on 360° LiDAR devices, which will be coupled to a mobile automaton that will rotate by a motor to get a full scan of the scene. Next, I will define some terms to understand the scanning principle of this system, also, the parameters and configuration of a 360° LiDAR, for instance the OS1-16, will be mentioned.

Azimuth: is the block that will contain the rays, this block will have a corresponding angle that will belong to the horizontal plane of the LiDAR. For this case, the LiDAR has 16 beams in each azimuth and is configured for a scan of 1024 azimuths.

Ticks: For a scan the LiDAR performs a 360° rotation. And, an encoder counts the necessary pulses to completed the rotation. In this case, the device has a total of 90112 pulses and between each azimuth the encoder counts a total of 88 pulses.

Altitude beam angles: This is the angle corresponding to each beam within the azimuth. The values of the angles corresponding to the 16 beams are shown in Figure 18.

Azimuth offset angles: The azimuth block has a corresponding angle, however, the beams belonging to the same azimuth do not maintain this same angle, resulting an angle offset. This offset is due to calibration issues in manufacturing. Because of this, LiDAR devices have different offset angles, despite being of the same model. Therefore, these sensors contain a command to obtain these angles, which will be used to obtain the coordinates of the point cloud. The command is shown in the Figure 17. And for this device, the values of the angles corresponding to the 16 beams are shown in the Figure 18.

Command	Description	Response Example

get_beam_intrinsics	Returns JSON-formatted beam altitude and azimuth offsets, in degrees.	<pre>{
		"beam_altitude_angles": [
		16.926,
		16.313,
		"...",
		-16.078,
		-16.689
		],
		"beam_azimuth_angles": [
		3.052,
		0.857,
		"...",
		-0.868,
		-3.051
		]
		}</pre>

Figure 17. Command to obtain the LiDAR angles[18]

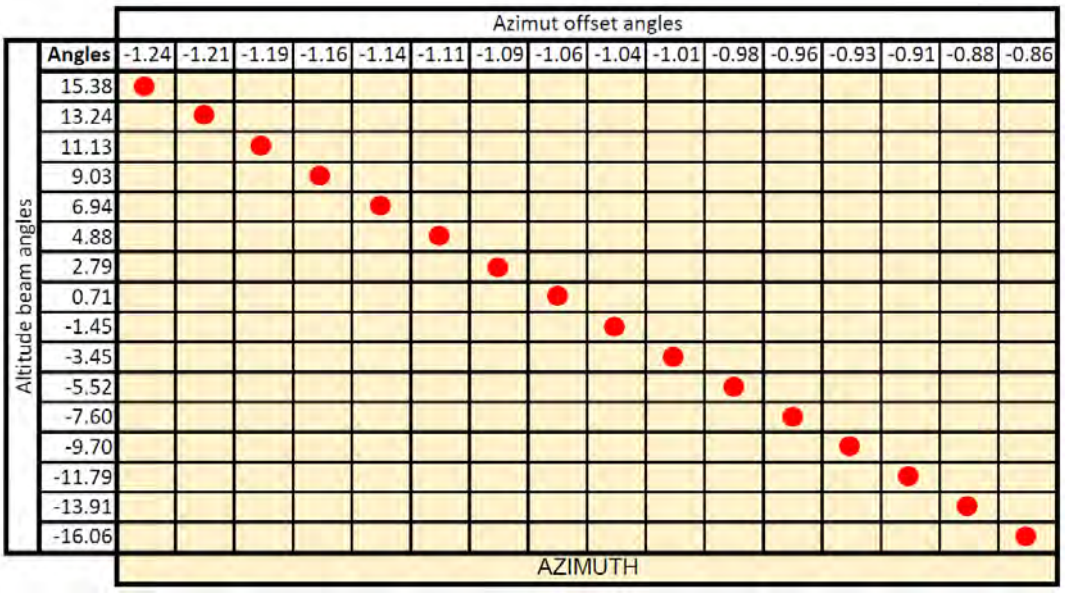


Figure 18. Representation of the angle values for the beams within azimuth block

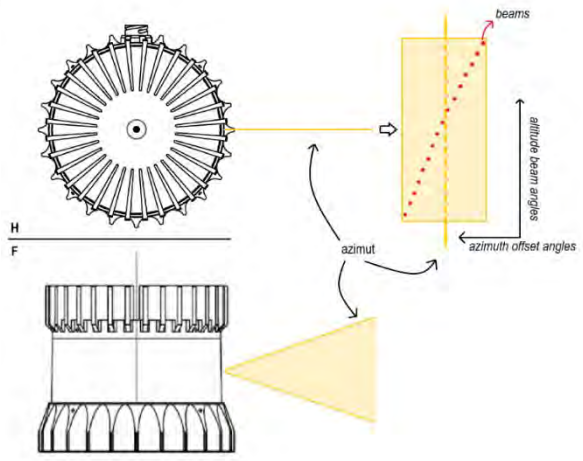


Figure 19. Views of a LiDAR OS1-16 [18]

2.2.1 Package data of a 360° LiDAR.

For the lidar used in this thesis, it was mentioned that the scanning involves 1024 azimuths, and each azimuth contains 16 beams, i.e., one scan of the sensor is equivalent to obtaining a cloud of 16384 points. As mentioned in chapter 1, each scan performed by this particular sensor is called a Donut. Also, the points obtained are not in cartesian coordinates and must be processed. Next, I will present the information obtained from each scanned point.

For instance, the information is received in packets containing the data of 16 azimuth blocks. Each azimuth block contains information such as: measurement time, measurement identifier, frame identifier, encoder counts, range, photon signal, reflectivity, environment photons, azimuth block status. But, for most applications, including this thesis work, only the following parameters will be used: encoder count and range. The following figure shows a table with the data of the package.

Word	Azimuth Block 0	Azimuth Block 1	...	Azimuth Block 15
(Word 0,1)	Timestamp	Timestamp	...	Timestamp
(Word 2[0:15])	Measurement ID	Measurement ID	...	Measurement ID
(Word 2[16:31])	Frame ID	Frame ID	...	Frame ID
(Word 3)	Encoder Count	Encoder Count	...	Encoder Count
(Word 4,5,6)	Channel 0 Data Block	Channel 0 Data Block	...	Channel 0 Data Block
(Word 7,8,9)	Channel 1 Data Block	Channel 1 Data Block	...	Channel 1 Data Block
	.	.		.
(Word 193, 194, 195)	Channel 63 Data Block	Channel 63 Data Block	...	Channel 63 Data Block
(Word 196)	Azimuth Data Block Status	Azimuth Data Block Status	...	Azimuth Data Block Status

Figure 20. Data of LiDAR packet information [18]

The channels used by this model are the i channels, $\forall\ i = 2 + 4 * n\ ;\ n \in [0,15]$, the other channels have irrelevant information. Inside these "word bits" is the range information (length of 20 bits) and other parameters. The following figure shows the data contained in the "Data Block".

Word	Byte 3	Byte 2	Byte 1	Byte 0
(Word 0)	unused[31:24]	unused[23:20] range_mm[19:16]	range_mm[15:8]	range_mm[7:0]
(Word 1)	signal_photons[31:24]	signal_photons[23:16]	reflectivity[15:8]	reflectivity[7:0]
(Word 2)	unused[31:24]	unused[23:16]	noise_photons[15:8]	noise_photons[7:0]

Figure 21. Data of the "Data Block" [18]

2.2.2 Conversion to a XYZ coordinate system.

With the range information, the encoder counts and the elevation and offset angles, it is possible to obtain the point cloud in the XYZ coordinates with the following formulas [18]

$$r = range\ mm$$

$$\theta = 2\,\pi\left(\frac{encoder_{count}}{90112} + \frac{beam_{azimuth_{angles[i]}}}{360}\right)$$

$$\emptyset = 2\pi\,\frac{beam_{altitude_{angles[i]}}}{360}$$

$$x = rcos(\theta)\ cos\ cos\ (\phi)$$

$$y = -rsin(\theta)\ cos\ cos\ (\phi)$$

$$x = rsin(\phi)$$

2.2.3 Register with other Donuts.

After generating the first Donut ($i = 0$) the motor rotates the sensor, stops, and performs another scan, and so on until obtain a full scan. The coordinate system for the other Donuts must be rotated i times, where i is the value of the Donut being scanned.

To perform the rotation, I use a rotation transformation matrix for each of the points, but for this I must define a coordinate system. The sensor has a coordinate system defined, but the

axes will be changed when coupled to the motor. So, the following LiDAR axis changes will be made:

$$X_L \rightarrow -Z_{ref}, \qquad Y_L \rightarrow Y_{ref}, \qquad Z_L \rightarrow X_{ref}$$

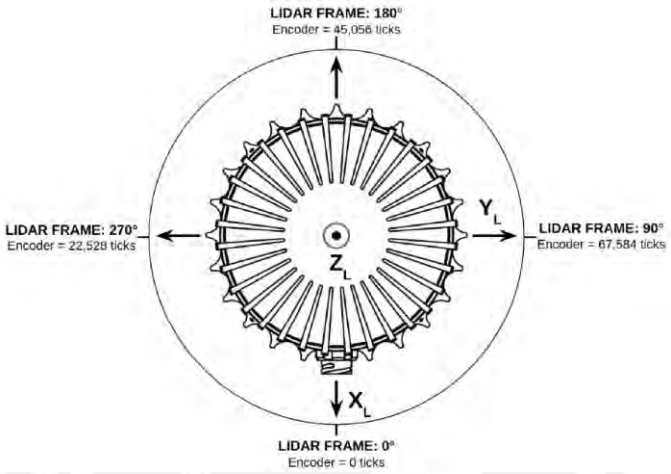


Figure 22. System coordinates of LiDAR OS1[18]

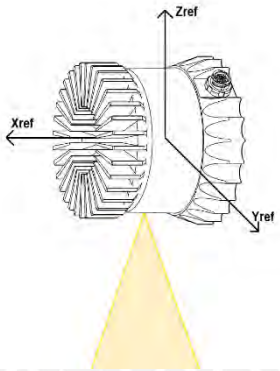


Figure 23. System coordinates of LiDAR-motor [18]

With the defined reference system, I can perform the rotation transformation for each data obtained from the sensor. The following rotation matrices about one axis are defined as follows [19]:

$$R_x(\alpha) = (1\ 0\ 0\ 0\ \ cos\ cos\ (\alpha)\ \ -(\alpha)\ \ 0\ (\alpha)\ \ \ cos\ cos\ (\alpha)\ )$$

$$R_y(\alpha) = (cos\,cos\,(\alpha)\ \ 0\,(\alpha)\ \ 0\,1\,0\ \ -(\alpha)\ \ 0\ \ cos\,cos\,(\alpha)\ \ )$$

$$R_z(\alpha) = (cos\,cos\,(\alpha)\ \ \ -(\alpha)\ \ 0\,(\alpha)\ \ \ cos\,cos\,(\alpha)\ \ 0\,0\,0\,1\,)$$

For the case of the Donuts only the matrix $R_z(\alpha)$ will be used, where alpha will be the angle that has been rotated in total from the referential Donut. The other matrices will be applied in the algorithms designed in the next chapter

2.3 Principles of Surface Reconstruction Techniques

Among the most popular surface reconstruction techniques are Delaunay triangulation and Voronoi diagrams.

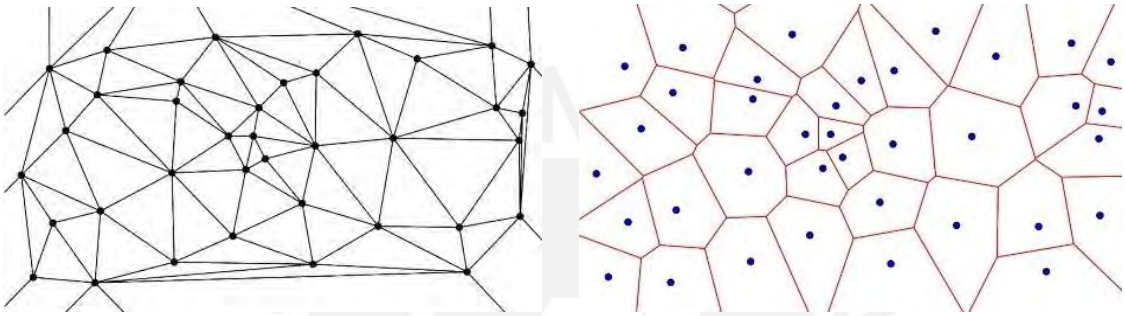
2.3.1 Delaunay triangulation.

On the one hand, in the Delaunay case, the triangulation principle is based on obtaining triangles where the vertices are the points of the point cloud, and as a condition each circumference containing the triangle cannot have a point inside it, in addition, it has a property called max-min which implies that the smallest angle of the triangles is maximized. In Figure 24.a. an instance of this triangulation is observed.

2.3.2 Voronoi diagrams.

On the other hand, Voronoi diagrams are made up of cells that enclose a single point of the point cloud. The generation of these cells is done with straight lines equidistant from the points formed, unlike Delaunay, the cells can be polygons where the number of sides depends on the number of adjacent points. [20]. In Figure 24.b an instance of Voronoi Diagram is observed.

Also, unlike Delaunay, Voronoi diagrams do not generate a triangular mesh, but generate cells that store each of the points. However, there is a duality between them where a transformation from cells to triangles is possible. Both represent the same thing but from a different point of view. The transformation consists of considering the vertices as centers of gravity of the polygon and the centers of gravity of the polygon as vertices. This rule applies to go from Delaunay to Voronoi and vice versa. In Figure 25 the duality can be observed.



(a) *(b)*

Figure 24. (a) Delaunay triangulation. (b) Voronoi diagram [20]

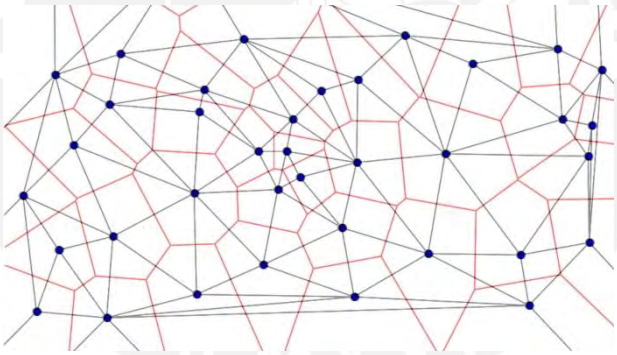


Figure 25. Delaunay and Voronoi duality [20]

In addition, is defined for both Voronoi and Delaunay the "convex hull", which is the smallest convex polygon containing all points. That is, the boundary polygon of the polygon generation. [20]

2.4 Basis of parallel algorithms

The goal of parallelization is to perform faster operations in order to reduce execution time. This need arises because there are several problems that require the processing of a large amount of data and lead to high execution time. Such as point cloud processing for Delaunay triangulation [21]. In recent years, parallel processing by GPUs has gained prominence and has advantages over CPUs in terms of cost and execution time [22]. In the following, some basic definitions of parallelization are presented, then some generalities of parallel algorithms, also parallel models, parallelization techniques, performance metrics and finally a generalization of the NVIDIA architecture.

2.4.1 Basic definitions.

- **Task:** It is the basic unit of operation that the operating system controls.
- **Granularity:** It is the size of the tasks.
- **Process:** It is an instance of the program, focused more on the global state of execution. It contains one or more threads.
- **Thread:** A subset of a process that shares resources with other threads.
- **Planner:** Allocates the processor and/or execution time for each thread to be executed
- **Map:** It is the assignment of a thread to a physical unit for execution.
- **Time complexity:** Describes the execution time it takes to run an algorithm as a function of the number of inputs.
- **Computational model:** Simplified representation of the computer architecture where the algorithm is planned to be executed.

2.4.2 Asymptotic notation.

In computer science, time complexity is expressed asymptotically, since when generalizing the algorithm for different input values it is irrelevant to know the time complexity constants of the algorithm. Therefore, an algorithm with complexity $O_1(n) = 3n$ and another with complexity $O_2(n) = 7n$ in computer science will have the same time complexity.

However, in most engineering applications, the algorithm that should be chosen is the one with complexity $O_1(n) = 3n$. [23]

2.4.2.1 O Notation.

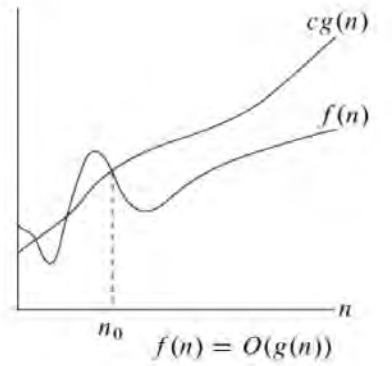


Figure 26. O notation [23]

It is satisfied that $f(n) = O\big(g(n)\big)$ if there exist two positive constants c and n_0 such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$

2.4.2.2 Ω notation.

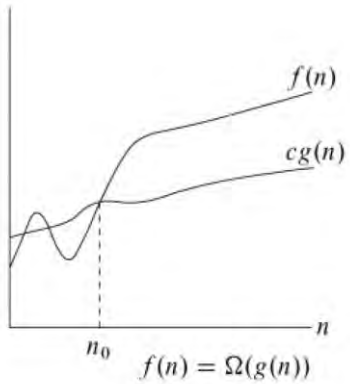


Figure 27. Ω notation [23]

It is satisfied that $f(n) = \ \Omega\big(g(n)\big)$ if there exist two positives constant c y n_0 such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$

2.4.2.3 Θ notation.

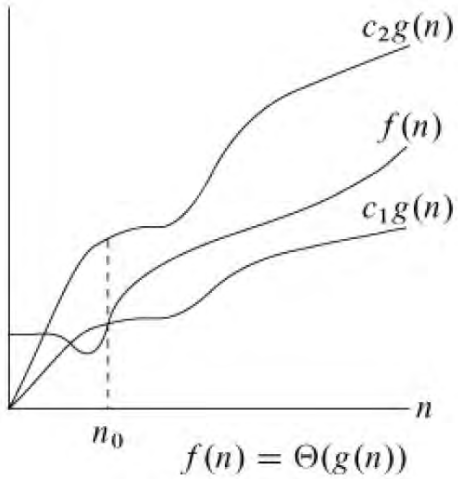


Figure 28. Θ notation [23]

It is satisfied that $f(n) = \Theta(g(n))$ if there exist two positive constants c_1, c_2 y n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$. In other words, the notation Θ exists if the other notations also exist, $f(n) = \Theta(g(n))$ si $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$.

2.4.3 Overview of parallel algorithms.

- **P:** Problem.
- **n:** number of inputs.
- **A:** parallel algorithm.
- **p:** number of processors.
- **T*(n):** Asymptotic time taken by the best-known sequential algorithm to solve problem P with an input of size n.
- **T_p(n):** Time in which A solves P using p processors.
- **T_1(n):** Time in which A solves P using 1 processor, it should be noted that this time is not necessarily similar to T*(n).
- **T_∞(n):** Time in which A solves P using infinite processors.
- **T(n):** Time in which A solves P using P(n) processors.

2.4.4 Parallel models.

The parallel computer is a set of interconnected processors that allow the coordination of their activities and information exchange. In the sequential case there is a RAM model which is widely accepted; however, parallel models are more complex and are not widely accepted. This is because the candidate models for parallelization must meet: *simplicity*, the representation of the architecture cannot be complex; and *implementability*, the model must be able to be put into operation i.e., the performance of the model must reflect the actual performance. Some of the candidate models that meet the above are:

- **DAGs (*Directed Acyclic Graph*):** models the activities to be executed in the form of a directed network.
- **PRAM (*Parallel Random-Access Machine*):** It is a collection of processors that communicate through a shared memory and work under a common clock. Because any processor can access any memory location it is necessary to group algorithms according to memory accesses. EREW, can only be applied to memories that do not allow concurrent writes/reads. CREW, allows concurrent reads, but not concurrent writes. CRCW. Allows concurrent reads and writes.
- **Distributed memory:** A collection of processors communicating through an interconnection network, each processor has a local memory and a unique identifier.

2.4.5 Parallelization Techniques.

Unlike the design of sequential algorithms, the design of parallel algorithms presents more difficult challenges and there is no guide, but rather a collection of techniques that have been found to be effective in solving various problems. Even so, in some scenarios the design is not feasible and the time complexity cannot be reduced. Some parallelism techniques are presented below.

- Balanced trees
- Jumping with pointers
- Divide and conquer

- Partitioning
- Segmentation
- Cascade acceleration
- Symmetry breaking

2.4.6 Performance metrics.

- **Acceleration:** let A which solves P at $T_p(n)$. The acceleration achieved by A is expressed by the following.

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

- **Efficiency:** Let A which solves P in $T_p(n)$. The efficiency achieved by A evaluates how efficient A is in using the p processors and is expressed by the following.

$$E_p(n) = \frac{T_1(n)}{pT_p(n)}$$

- **Cost:** I define the cost of a parallel algorithm as the asymptotic execution time for P(n) processors times the number of processors.

$$C(n) = T\,(n)P(n)$$

- **Work**: It is the number of operations that the parallel algorithm executes without taking into account the number of processors.

$$W(n)$$

- **Weak Optimality:** When the number of operations performed in sequential is the same as the number in parallel.

$$W(n) = \Theta(T^*(n))$$

- **Strict Optimality:** When the asymptotic execution time in parallel is p times faster than the best asymptotic time in sequential.

$$p = O(\frac{T^*(n)}{T(n)})$$

2.4.7 NVIDIA architecture.

In 2006 NVDIA released CUDA (Compute Unified Device Architecture) which is a GPU-based computing platform. Developers were able to access the GPU's instruction list and memory through CUDA and execute parallelism [22]. Over time NVDIA has evolved its GPU architectures, there have been noticeable changes between the early architectures, but as of Fermi the essence is pretty much the same. The following is a list of NVIDIA's architecture names [24].

- Tesla – 2006
- Fermi – 2010
- Kepler – 2012
- Maxwell – 2014
- Pascal – 2016
- Volta – 2017
- Turing – 2018
- Ampere – 2020

Figure 29. Fermi architecture [24]

The figure above shows the Fermi architecture. Where the orange block is called "GigaThread Engine" this block will be in charge of distributing the tasks to the GPU GPCs. On the other hand, the CPU has access to define the number of threads, grouped in blocks, and blocks that will be executed in the GPU, both can be defined in one dimension, two dimensions or three dimensions.

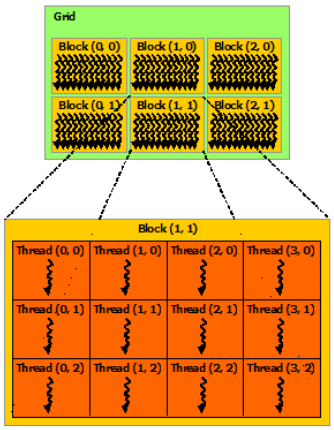


Figure 30. Grid composition[24]

Within the GPCs of the Fermi architecture, there are some blocks called streaming multiprocessor (SM), the architecture of these blocks can be observed in Figure 31. Inside the SM there is the Warp Scheduler that is in charge of scheduling the warps to the available resources of the SM. The warps are a group of 32 threads. Meanwhile, the Dispatch Unit is the one that sends the warps to the resources.

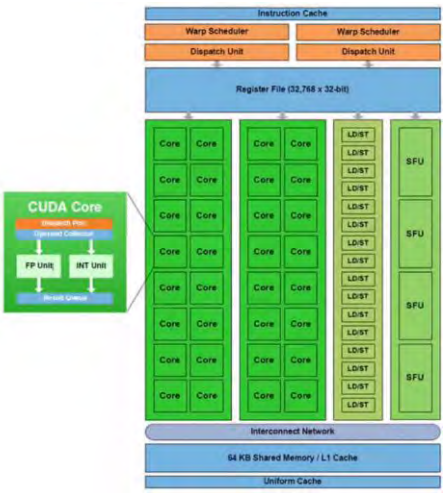


Figure 31. Architecture of the streaming multiprocessor [24]

In the case of the Fermi architecture, the resources available to the SM are integer/float, load/store, SFU (transcendental operations such as power and sinusoidal). Also. Each SM contains a shared memory only for the blocks belonging to the SM, which also has space to act as L1 (level 1) cache memory. However, when the information is not found in the cache or in the shared memory, it accesses the L2 cache memory (level 2) where it will take an average of 50 clocks, and if it does not get the required information in this memory either, it accesses the global memory with an average of hundreds of clocks, but the access to the global memory is through a bandwidth of 32 bytes, which could compensate the access to the global memory. The following figure shows the explained above.

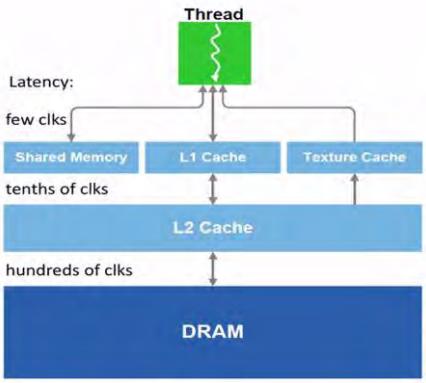


Figure 32. Memory hierarchy. [24]

3 Chapter 3. Overlap Removing and Mesh Generation

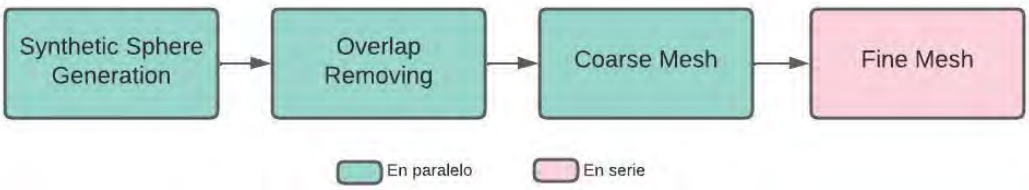


Figure 33. Architecture of the software

In this chapter, the method for mesh reconstruction is described. In Figure 33, the core idea is represented in four main stages, stages in green color run low complexity parallel algorithms and the stage in red color runs a sequential algorithm. Parallel techniques are applied in order to achieve real time process and low latency framework for 3D spheric scene reconstruction.

3.1 Synthetic Sphere Generation

In general, there are three key requirements involved in mesh reconstruction from point clouds: local region detection, geometric feature keeping, and resampling [25]. This work focuses on two of those three keys required. Local region detection is related with neighbor relationship of points, in other words, if the location of a point is known, the other points can be located with its neighbor relationship. For mesh reconstruction, geometric features should be kept for geometric analysis. Also, resampling converts the initial mesh into an isotropic mesh, the distances between points are approximately equals.

The solution proposed focuses in maintaining the neighbor relationship of points from the raw data by using a priori information from the measurement system for a fast mesh reconstruction. In order to achieve this, its important to realized that all LiDAR follow a

stablish scan pattern that allows us to know, a priory, the mesh between neighboring points. In Figure 34, it is solved the case where the LiDAR has a vertical resolution of nBeams channels (with a vertical field of view VFoV) and a horizontal resolution of nAzBlk azimuths (with a horizontal field of view HFoV). One scan (HFoV = 360◦) rotates along its x axis and generates a donut (set of nBeams×nAzBlk points). Each point follows the standard (x, y, z) cartesian coordinates. However, to have a full scan of the scene, is necessary to perform nDonuts scans (rotating the LiDAR angleRot[1 . . . nDonuts] along its z axis, not exceeding VFoV per scan).

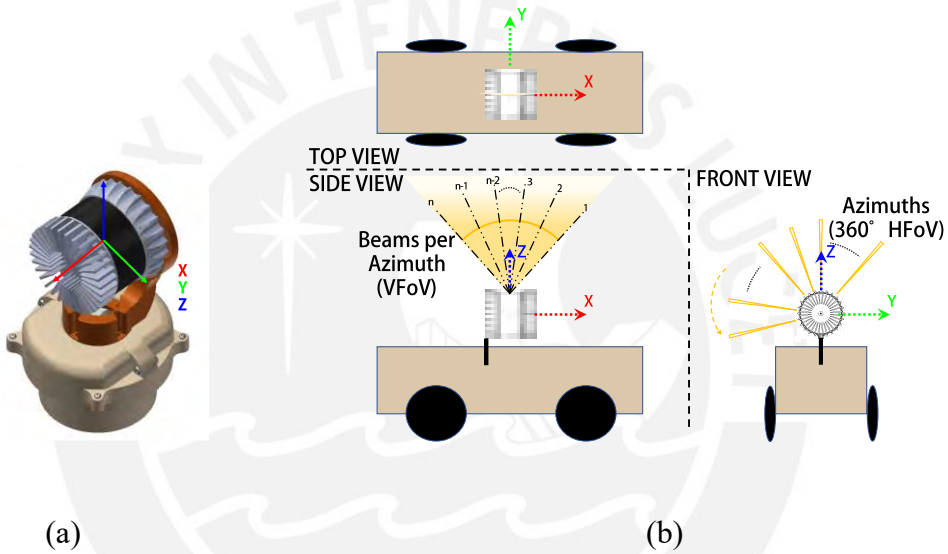


Figure 34. Measurement system

From the raw data, the points of one scan are sorted as the matrix show in Figure 35. The rows represent the amount of beams in one azimuth block, meanwhile the columns represent the amount of azimuth blocks. In general, the data obtained goes with the first nBeams in one azimuth block then continue with the next block of beams until sweep all the number of azimuths. To identify the id. of a point, the next formula proposed in Figure 35, using the information of the id azimuth (idAz) and the id Beam (idBeam) of a point.

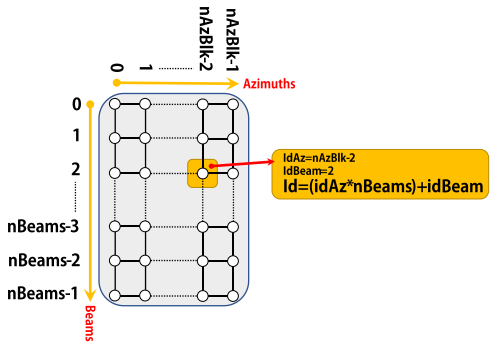
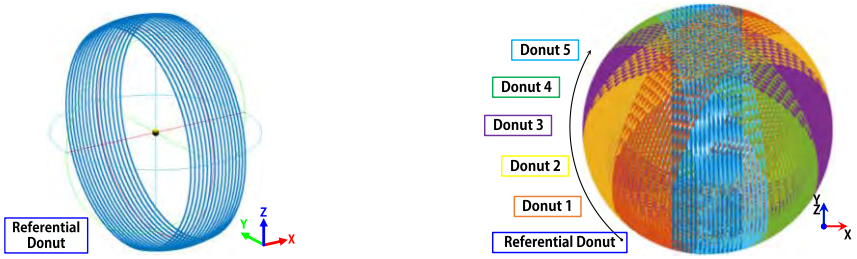


Figure 35. Sorted matrix of raw data

After a full scan (nDonuts) overlapping sections can be observed, especially in the upper and lower area of the XY plane, see Figure 36.b. In order to remove the overlapping points, is necessary to generate a synthetic sphere that represents an ideal scan of a sphere structure, this solution is proposed in Algorithm 1. This algorithm needs some parameters of the measurement system such as: the amount of azimuths per Donut scan (nAzBlk), the amount of beams in an azimuth block (nBeams), the amount of Donuts required to get a full scan (nDonuts), the rotation angle for each donut (angleRot[nDonuts]), and the elevation angle for each beam within an azimuth block (angleElev[nBeams]), also, the radius of the sphere is set to 1. First, it generated an initial azimuth block with Algorithm 1.1 (line 3), the elevation angle of each beam is used to generate the points of the azimuth block. Then, these points are rotated along X axis (line 5) to get one Donut, the referential Donut (see Figure 36.a). Finally, the Referential Donut generated is rotated to get the next Donuts (line 9).



(a)Referential Donut *(b) Full Donut scan*

Figure 36. Donut scan

Algorithm 1: Synthetic Sphere Generation

inputs: nAzBlk, nBeams, nDonuts, angleRot[nDonuts], R (Sphere radius, set to 1), angleElev[nBeams] (elevation angle of each beam within an azimuth).

output: *Sphere:* Point cloud array of synthetic sphere

1: **procedure** *Synthetic_Sphere_Generation* (inputs, outputs)

2: $angle \leftarrow \frac{360°}{nAzBlk}$ ▷angle between azimuths

▷Generate points for one synthetic azimuth

3: $AzPoints_{xyz}$[1...nBeams] ← *buildAzPoints*(R,nBeams, angleElev) ▷Azimuth Block Generation

4: **for** i ←0 to nAzBlk -1 **in parallel do**

5: $RefDonut_{xyz}[i \times$ nbeams] ←rotXaxis($AzPoints_{xyz}$, $angle \times i$) ▷Azimuth Block Rotation

6: **end for**

▷Generate the synthetic sphere

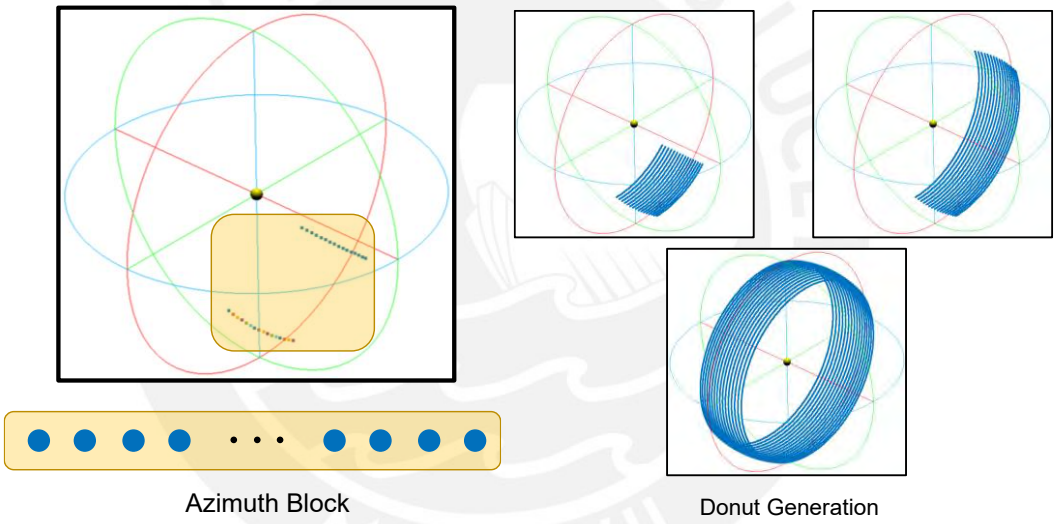
7: $nPPD \leftarrow nAzBlk x nBeams$ ▷*points per donuts*

8: **for** i ←0 to nDonuts -1 **in parallel do**

9: $Sphere_{xyz}[i \times nPPD]$ ← rotZaxis($RefDonut_{xyz}$, angleRot[i]) ▷Referential Donut Rotation

10: **end for**

11: **end procedure**



Note: In this work, it is stablished that initial nBeams target to negative Z-axis (lines 3-5).

Algorithm 1.1: Azimuth Block Generation

inputs: nBeams, R (Sphere radius, set to 1), angleElev[nBeams]

output: $AzPoints_{xyz}$: Point cloud array with nBeams points

1: **procedure** *buildAzPoints*(inputs, outputs)

2: **for** i ←0 to nBeams-1 **in parallel do**

3: $x \leftarrow sin$(angleElev[i])

4: $y \leftarrow 0$

5: $z \leftarrow -cos$(angleElev[i])

6: **end for**

7: **end procedure**

3.2 Overlap Removing

In Figure 36.b, the overlap area can be observed. To remove redundant points, Algorithm 2 is proposed, the general idea is to set (0,0,0) all the overlapping points as shown in Figure 37. this algorithm does not need the actual points, just the scan pattern that is known a priory. As inputs, the same as Algorithm 1 are used with addition of the synthetic Sphere obtained to be modify, and the output will be the same Sphere point cloud but with redundant points set to (0,0,0). For simplicity, the top view of the synthetic sphere is taken as a reference (XY plane), due to the scan patten stablished by the measurement system, see Figure 38. Each donut defines two planes (A and B) perpendicular to the XY axis, where all the points are contained between those planes (e.g., Plane A and B for the yellow donut in Figure 38), in order to obtain these planes, two points are selected which correspond with the first horizontal Beam and the last horizontal nBeam, respectively (lines 2-3). Then, those points are rotated with the respectively angle for each donut and the equation line is defined to get the planes A and B (lines 5-6). Also, it is defined that all the points of the reference donut (RefDonut), blue Donut in Figure 38, are non-overlapping and are contained between bold blue planes. For the remaining donuts (lines 7 - 16), each one is compared with the previous donut. Tthe points that fall below Plane A of the previous donut (for the left side of the sphere) and above Plane B for the right side (e.g., see purple donut in Figure 38) are removed. Then, the remaining points that fall inside the planes of the RefDonut are also removed.

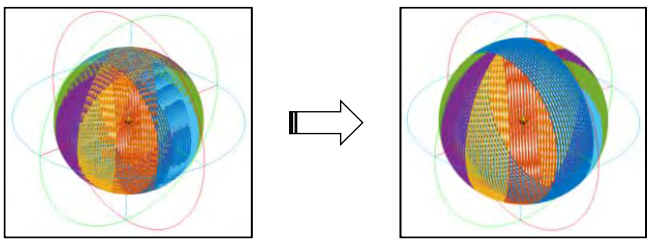


Figure 37. Overlap Removing result

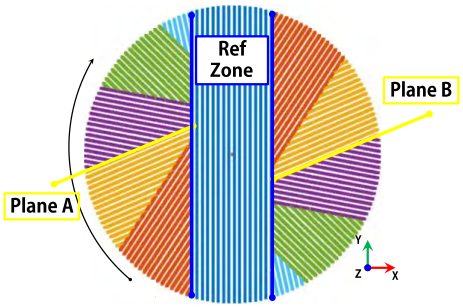


Figure 38. Synthetic Sphere with no overlap points

Algorithm 2: Overlap Removing

inputs: nAzBlk, nBeams, nDonuts, angleRot[nDonuts], R (Sphere radius, set to 1), angleElev[nBeams] (elevation angle of each beam within an azimuth), *Sphere* (Point cloud array of the synthetic sphere from Alg.1)

output: *Sphere:* Point cloud array with redundant points set to (0,0,0)

procedure *OvRe* (inputs, outputs)
    $xA \leftarrow R \times cos$(angleElev[nBeams-1])
    $xB \leftarrow R \times cos$(angleElev[nBeams-1])
    for $i \leftarrow 1$ to nDonuts -1 **do**
        $PlaneA \leftarrow rotPlaneZaxis(xA,$ angleRot$[i])$
        $PlaneB \leftarrow rotPlaneZaxis(xB,$ angleRot$[i])$
        for $j \leftarrow 0$ *to* nAzBlk $\times$ nBeams $-$ 1 **in parallel do**
            $index \leftarrow i \times$ nAzBlk $\times$ nBeams $+j$
            $x, y, z \leftarrow Sphere[index]$
            cond1: (x, y) inside $RefZone$
            cond2: $x < 0$ AND $y < PlaneA$
            cond3: $x < 0$ AND $y > PlaneB$
            if cond1 OR cond2 OR cond3 **then**
                setPointToZero($Sphere[index]$)
            end if
        end for
    end for
end procedure

3.3 Coarse Mesh

With the redundant points set to (0,0,0), the surface reconstruction is next. An initial mesh can be generated using the scan pattern within each Donut, this is done in Algorithm 3. For this section, the order of the vertices of the triangles is set as counter-clockwise when the sphere is viewed from an external perspective. This algorithm generates two triangles per point

in the sphere, excluding the points scanned for the last beam, i.e., there will be 2 × (nbeams − 1) triangles between azimuths in each donut. The main loop (lines 6-20) generates in parallel all the vertices. Inside the loop, for each point, its location is computed (e.g., see Figure 39, yellow dot) and three more neighbors are selected to generate two triangles (e.g., see Figure 39, right side). Also, since in parallel processing race condition must be avoided, a Flag array is created which give us the information of which triangle should be removed or considered, those triangles with a vertex equal to point-zero are removed. Finally, in a sequential process the flag array is verified to count the amount of triangle removed and generated the final Mesh (line 21). This last step will remove all the overlapping triangles; however, it will create some gaps in the surface (see Figure 40).

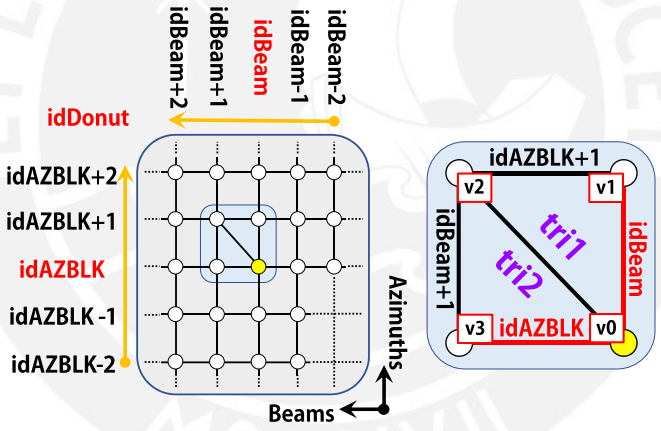


Figure 39. Triangle connection for Coarse Mesh

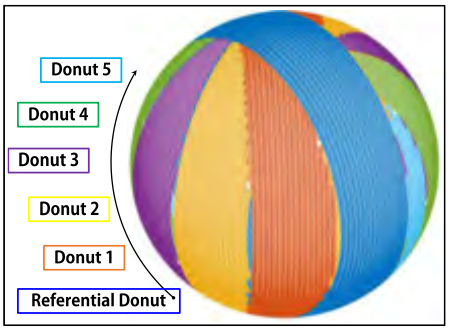


Figure 40. Coarse mesh reconstruction

Algorithm 3: Coarse Mesh

inputs: nAzBlk, nBeams, nDonuts, *Sphere* (Point cloud array with redundant points set to (0,0,0))

output: *Mesh* (array with triangles vertices), *nTriangles* (number of triangles generated)

procedure *CoarseMesh*(inputs, outputs)

▷ *nTh:* number of threads

$nTh \leftarrow nDonuts \times nAzBlk \times (nBeams - 1)$

$Mesh_{v0,v1,v2} \leftarrow$ emptyList()

$triangleFlag \leftarrow$ emptyList()

$countTriangles \leftarrow 0$

for $i \leftarrow 0$ *to* nTh **in parallel do**

$idDonut \leftarrow \lfloor i/((nBeams - 1) \times nAzBlk) \rfloor$

$idAzBlk \leftarrow \lfloor (i - idDonut \times nAzBlk)/nAzBlk \rfloor$

$idBeam \leftarrow i - idDonut \times$ nAzBlk $- idAzBlk \times (nBeams - 1)$

$v0 \leftarrow i$

$v1 \leftarrow ((idAzBlk + 1)\%$nAzBlk$) \times nBeams + idBeam + idDonut \times nAzBlk \times nBeams$

$v2 \leftarrow v1 + 1$

$v3 \leftarrow v0 + 1$

$tri1_{v0,v1,v2} \leftarrow v0, v1, v2$

$triangleFlag[i \times 2] \leftarrow$ checkVexZero(tri1) ▷0: relevant triangle, 1: redundant triangle, must be removed

$tri2_{v0,v1,v2} \leftarrow v0, v2, v3$

$triangleFlag[i \times 2 + 1] \leftarrow$ checkVexZero(tri1) ▷0: relevant triangle, 1: redundant triangle

$Mesh_{v0,v1,v2}[i \times 2], Mesh_{v0,v1,v2}[i \times 2 + 1] \leftarrow tri1, tri2$

$countTriangles \leftarrow countTriangles + 2$

end for

▷Remove triangles with a Vertex equals to point-zero

$countTriangles, Mesh \leftarrow removeTrianglesVexZero(Mesh_{v0,v1,v2},$ triangleFlag)

end procedure

3.4 Fine Mesh

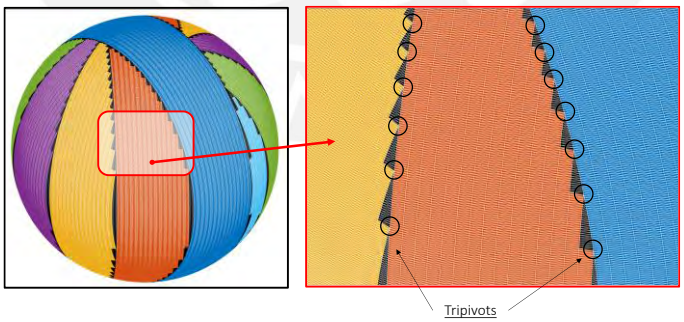


Figure 41. Pivot triangles

In the previous algorithm, the sphere obtained presents small gaps between donuts. To fill those gaps, it is necessary to connect the nearest points between different donuts. In most

gap areas only two donuts are involved but in some top and bottom areas there will be gaps which involve three donuts. To fill those gaps the creation of a pivot triangle is proposed.

3.4.1 Pivot triangles generation

Pivot triangles (tripivot) are defined as those triangles that link one point from a donut with two points from another adjacent donut (e.g., see Figure 41, black circles). To compute the tripivots, is required to divide each donut in four sectors (see Figure 42). Since Referential Donut has no point removed, only the next Donuts need to be analyzed. For the other Donuts points removed generate a particular shape in the edge, in the Figure 42.ii is shown the two possible shape that can be obtained, also, depending of the sector the shape can be defined as class A or B (see Figure 42.iii). Note that, if one side has a "stair shape" the other side of the Donut, in the same sector, has a "ramp shape".

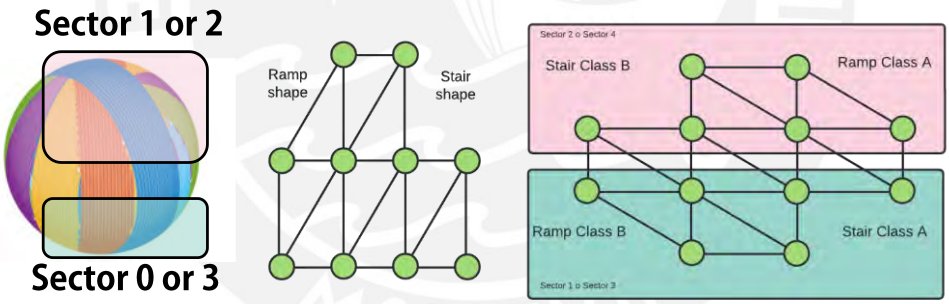


Figure 42. Sectors and type of shape of a cropped Donut

With these new properties defined, is simple to generate the pivots triangles. First, the algorithm finds the border points of a sector for each Donut. In the case of sector 0 or 3, a loop is done starting from the bottom point and the algorithm sweeps with nBeams steps until reach a point different to (0,0,0); for the case of the sector 1 or 2, it starts from the top and sweeps until reach a point different to zero. The direction of the sweep will be in a positive step if the

sector is 0 or 2, but it will be negative for sectors 1 and 3. After reaching the border point, the shape can be stablished, this property will help in the fill the gap stage. To decide the type of shape is necessary to check if a particular neighbor point is a zero-point if this condition meets the shape is a “ramp shape”, for sectors 0 and 3, or a “stair shape”, for sectors 1 and 2. The position of this neighbor point also depends on the sector. In Figure 43 can be observed in red color the neighbor point and in yellow color the border point, in this figure for both cases the neighbor point is different to zero-point, as a result is a “ramp shape” for sectors 1 and 3; and a “stair shape” for sectors 0 and 2.

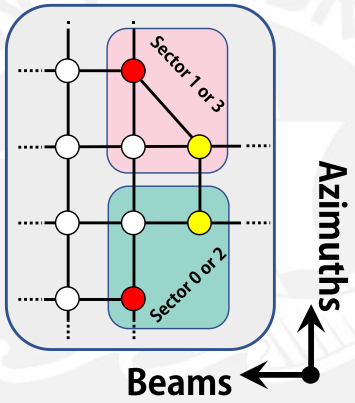


Figure 43. Position of neighbor point for the decision of the shape

In general, for each border point two connections can be possible: RefDonut connection or PrevDonut connection, in some cases can be both. To know the type of connection is required to compare the alfa and beta angles of the point. the alfa angle is defined as the angle former between the projection of the point in the XY plane and the negative Y axis; and the beta angle is defined as the angle former between the projection of the border point in the XY plane and the rotation angle of the previous Donut of the Donut which the point belongs (see Figure 44). If the beta angle is minor than alfa, the connection of the pivot triangle if with the previous Donut, in other cases the connection is with the Referential Donut.

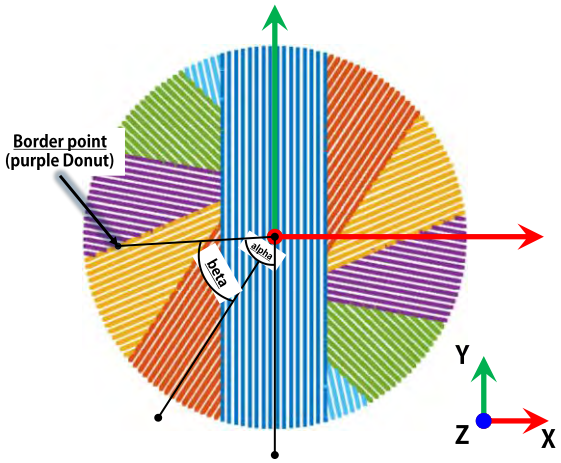
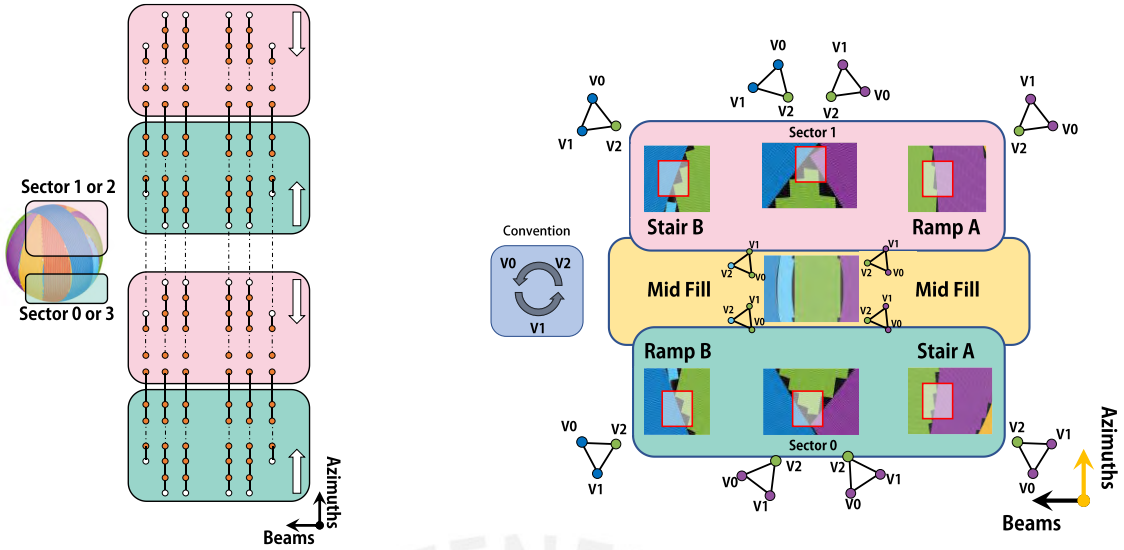


Figure 44. Alpha and Beta angles

Then, with the Y-coordinate and Z-coordinate of the point the angle elevation with Z-axis can be obtained with the donut stablished for the connection, two points are found and surround this elevation angle and those points are set as vertices V0 and V1. This process is repeated for all the border points of my sphere.

In Figure 45 an example of this process is shown, on each sector it sweeps (in the white arrow direction, Figure 45.a) each column beam until it gets an azimuth border point (a non-zero point). Then, that point and the two closest points of the adjacent donut defines the tripivot. For instance, in Figure 45.b, different tripivots triangles are shown, also, it is stablished that vertex V2 is always the border vertex of a Donut, and the other vertex are connected with a different Donut. Still, must be considered, there are some particular cases were two tripivots connect with three different donuts (see in Figure 45b, the middle top and bottom instances of tripivots triangles connected with three different donuts). With the tripivots established, the zone that they surround can be filled by connecting the points. There are several ways how the fill can be performed, in the next section all the cases presented in this work will be explained.



(a) Sectors of a remaining Donut *(b) Instances of tripivots triangles*

Figure 45. Remaining Donut features

Algorithm 4: Fine Mesh

inputs: nAzBlk, nBeams, nDonuts, angleRot[nDonuts] (rotation angle of the motor for each donut), *Sphere* (Point cloud array with redundant points set to (0,0,0))

output: *Mesh* (array with triangles vertices), *nTriangles* (number of triangles generated)

procedure *FineMesh*(inputs, outputs)

$params_{DAB} \leftarrow [nDonuts, nAzBlk, nBeams]$

▷ The first Donut does not need this mesh

for $i \leftarrow 0$ *to* $nDonuts - 1$ **do**

▷ Divide the Donut in 4 zones

for $sector \leftarrow 0$ *to* 3 **do**

$counter \leftarrow 0$

$Pivots \leftarrow emptyList()$

for $Beam \leftarrow 0$ *to* $nBeams - 1$ **do**

$azimuth \leftarrow getAzimuthBorderPoint(sector, Beam, nAzBlk, Sphere)$

if $azimuth\ found$ **then**

$idPoint \leftarrow [i, azimuth, Beam]$

$tripivot_{v0,v1,v2} \leftarrow getTripivot(idPoint, rotAngle[i], params, Sphere)$

$Pivots_{v0,v1,v2}[counter] \leftarrow tripivot$

$counter++$

end if

end for

if $counter \neq 0$ **then**

$Fill_{v0,v1,v2} \leftarrow FillZone(i, sector, counter, Pivots)$

$concatenate(Mesh, Pivots, Fill)$

end if

end for

end for

end procedure

3.4.2 Filling the zone surrounded by pivot triangles (FillZone)

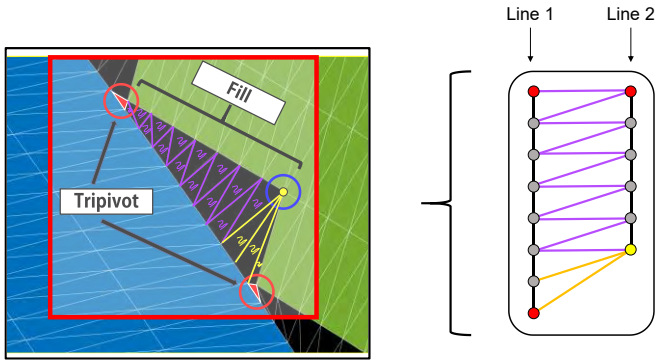


Figure 46. Two-Donut Gap Fill

Case 1: This is the most general case, an instance of this is shown in Figure 46. First, the two tripivots are computed (upper red circle Tp1 and lower red circle Tp2), then the blue circle is computed (Tp3, next beam from the lower tripivot). Starting from Tp1 I build the triangles sweeping the azimuths on both sides (see purple triangles). However, it ran out of azimuths on the green side, therefore, it keeps that point fixed (Tp3) and keeps sweeping on the other donut until it hit Tp2 (see yellow triangles). Depending on the sector and the type of shape the direction of the steps for the fill would be positive or negative, it uses the information of the table below.

Sector	Shape	Class	Donut	Step-L2
1 o 2	X	A	Prev	-
1 o 2	X	B	Ref	+
3 o 4	X	A	Ref	+
3 o 4	X	B	Prev	-

Table 1. Information for Two-Donut Gap Fill.

Case 2: Despite most of the gaps are filled, there are some particular gaps in the top and the bottom of the sphere where three Donuts are involved, see Figure 47.

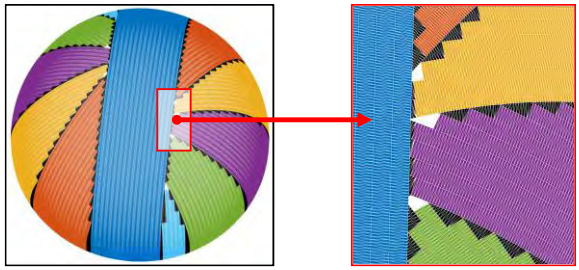


Figure 47. Tri-Donut Gaps

For this particular fill, a rigorous analysis was done. There was up to 128 possibilities but with the new properties defined (sectors, type of shapes) it can be summarized all of them in Figure 48, also, in Table 2 is the information for the Fill. The information on this table only works if the local region was established as mentioned in previous section. The order of the neighboring point is important to obtain the correct fill.

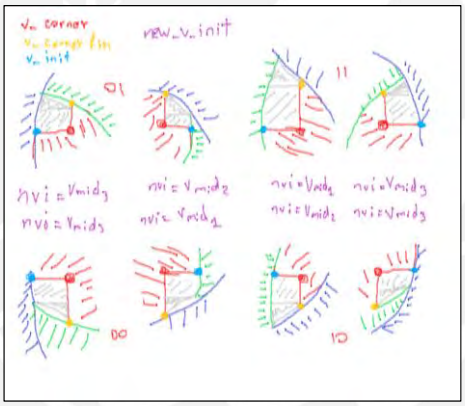


Figure 48. Possible scenarios of Tri-Donut Gap Fill

Sector	Shape	Class	V_lim1	index	V_lim2	index
1	Stair	B	V_tripivot	3	V_trimid	1
1	Ramp	A	V_tripivot	3	V_trimid	3
2	Stair	A	V_tripivot	3	V_trimid	3
2	Ramp	B	V_tripivot	3	V_trimid	2
3	Stair	B	V_tripivot	3	V_trimid	3
3	Ramp	A	V_tripivot	3	V_trimid	2
4	Escalera	A	V_tripivot	3	V_trimid	1
4	Rampa	B	V_tripivot	3	V_trimid	3

For index “2”:

```
Xor( sector(2),nor( sector(1),escalera ) )
```

For index “1”:

```
Not( xor( sector(2),sector(1) ) || c )
```

Table 2. Information for Tri-Donut Gap Fill

In Figure 49 is shown the stages of this fill. The idea is to divided this complex fill into Two-Donut Gap Fill. First, the initial, middle and last tripivots are defined. Then, with the initial and the middle tripivots a parallel fill can be done, sweeping the azimuths on both sides. Finally, with the middle and last tripivots a Two-Donut Gap Fill can be done.

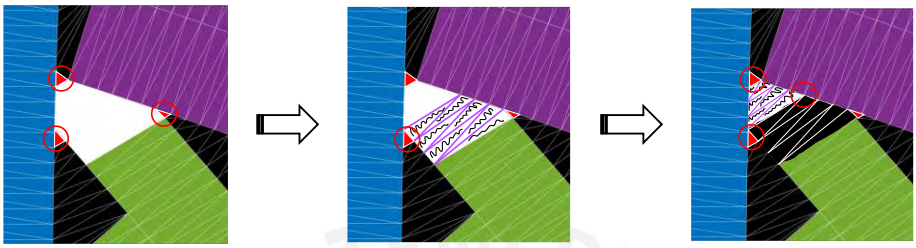


Figure 49. Tri-Donut Gap Fill process

In summary (see Figure 50), the process of the Overlap Removing and Mesh generation algorithm is focus on maintaining the scan pattern (local region property) and use this information reduce time processing. Also, it removes those redundant points with no additional information for its purpose. Finally, the mesh generation is done using the information of the local region. In addition, this mesh result can be used in a Voxel-based reduction structure in order to obtain an isotropic mesh and achieve an ideal mesh reconstruction.

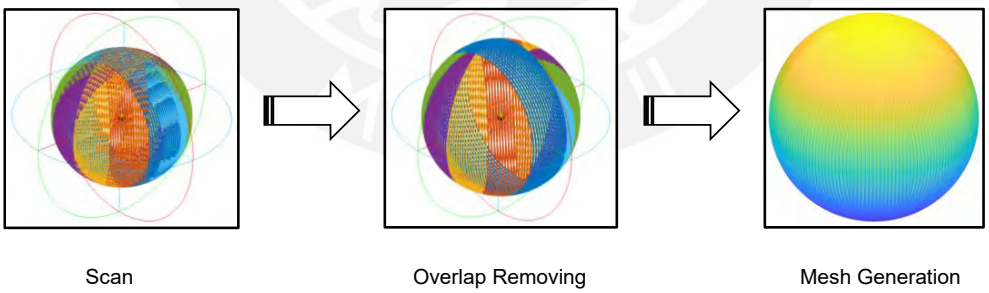


Figure 50. Summary of the Overlap Removing and Merge Generation Algorithm

4 Chapter 4. Mesh Generation, Results and Analysis

In this chapter algorithm is tested in two different experiments to obtain results of time and performance, also the technical specifications of the hardware used for the experiments is detailed.

4.1 Technical details

For the experiments, a 360° LiDAR sensor is used to obtain the data and an embedded system with a GPU integrated to test the performance of the ORaMG algorithm (Overlap Removing and Mesh Generation). In this thesis, two different GPUs alternatives are used for each experiment.

- OS1 OUSTER LiDAR.
  - Horizontal FOV of 360° degrees
  - Vertical FOV of 33.2°
  - Amount of azimuth blocks in a 360° rotation: 1024 (nAzBlk)
  - 16 beams in each azimuth block (nBeams)
  - Scan rotation rate of 10 Hz
  - 163840 points per second
- Desktop computer.
  - CPU Intel® Core™ i9-9900KF @3.60GHz with 8 physical cores
  - 16GB of RAM memory
  - GPU NVIDIA® GeForce RTX 2070
    - 2304 CUDA cores @1.41 GHz
    - 8 GB RAM with a bandwidth of 448 GB/s
    - Turing Architecture
- Embedded system.
  - Low power consumption Jetson Nano 2GB Developer Kit
    - 128 CUDA cores GPU @640 MHz
    - 2 GB RAM with a bandwidth of 25.6 GB/s
    - Maxwell architecture

Two experiments setups are designed for testing ORaMG:

- Synthetic Test: In a desktop computer, running times of Algorithm 1, 2, 3 and 4, using a synthetic sphere of radius 1. Is expected to obtain an execution time less than the rotation time established in order to achieved real time.
- Field test: In a Jetson Nano, running times and mesh reconstruction analysis using real point cloud data from a mine captured with a 360° LiDAR and the measurement system.

4.2 Synthetic test

In this section, the partial result and the details of each algorithm are analyzed. Also, all the information obtained is gathered in a table.

4.2.1 Synthetic Sphere Generation

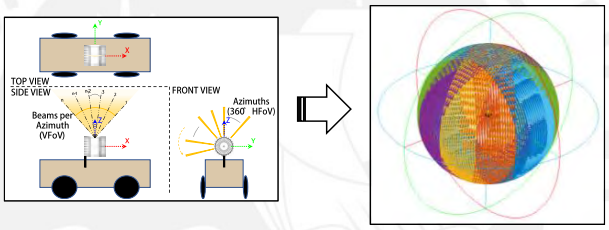


Figure 51. Synthetic sphere generated

As explained in chapter 3, knowing and mathematically modeling the scan patter of the LiDAR device, a synthetic sphere can be generated. In Figure 51, six different color is observed, where each color represents a Donut scan, in my case six scan was enough to do a total scan of the scene using a 33.5370° rotation of the motor.

Also, is considered that the LiDAR devices has the 9 possible combinations of scan, I can set this combination changing the amount of beams or the amount of azimuths. Since, some combinations lead to point cloud with the same initial amount of points I gathered them into groups, which I called PCxxKp, were xx denotes the total number of points divided by 1024.

The information of points for the different combinations is in the next table. In addition, in Figure 52 the spheres obtained for the five different PCxxKp group are ilustrated.

PCxxKp	PC98Kp	PC196Kp		PC393Kp			PC786Kp		PC1572Kp
nBeams	16	16	32	16	32	64	32	64	64
nAzimuths	1024	2048	1024	4096	2048	1024	4096	2048	4096
Points per Donut	16384	32768	32768	65536	65536	65536	131072	131072	262144
Total points of the sphere	98304	196608	196608	393216	393216	393216	786432	786432	1572864
Size of memory (MB)	9.437	18.874		37.748			75.497		150.994

Table 3. Synthetic sphere details for different scan combination

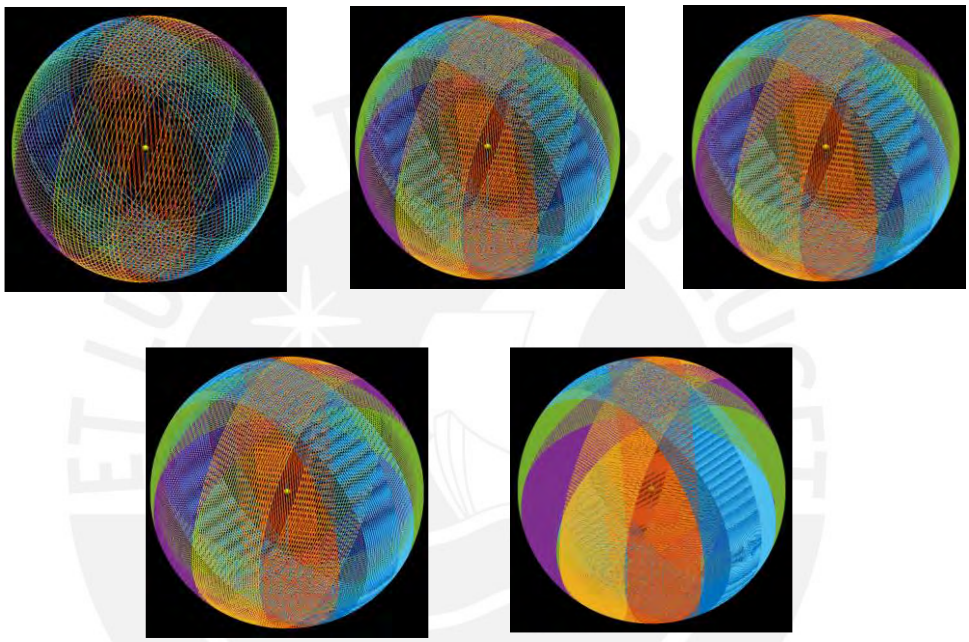


Figure 52. Spheres generated for each PCxxKp group

4.2.2 Overlap Removing

With Algorithm 2 (Overlap Removing) the redundant points are removed for each combination. In the next table I can observed that the amount of points removed is almost constant, in general 39% of the initial points are removed. This means that the time processing is considerable reduced since is only necessary to process the points remaining. In addition, in Figure 53 can be observed how the points remotion is done, in the top there are the synthetic sphere with all the points, and in the bottom the spheres with no redundant points.

PCxxKp	PC98Kp	PC196Kp		PC393Kp			PC786Kp		PC1572Kp
nBeams	16	16	32	16	32	64	32	64	64
nAzimuths	1024	2048	1024	4096	2048	1024	4096	2048	4096
Points Removed	39171	78346	77895	156706	155789	155233	311585	310498	620963
Percent Removed (%)	3985	39.85	39.62	39.85	39.62	39.48	39.62	39.48	39.48
Points Remaining	59133	118263	118713	236510	237427	237983	474847	475934	951901

Table 4. Details of points removed

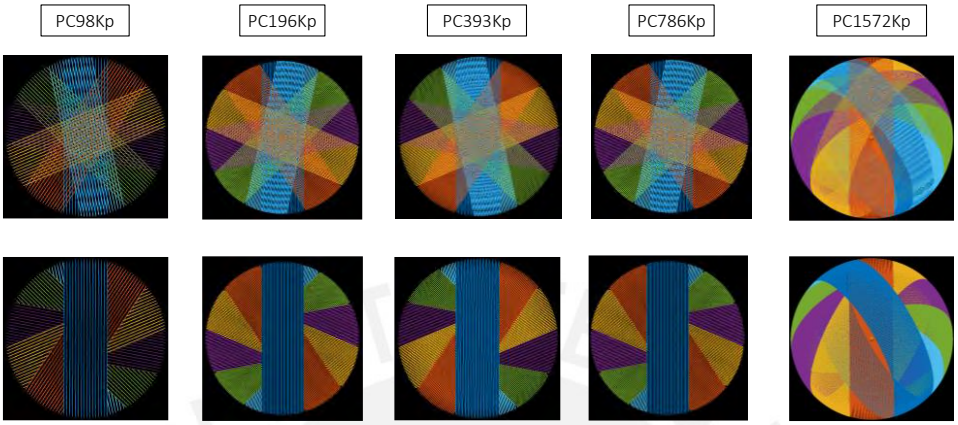


Figure 53. Spheres with redundant points removed

4.2.3 Coarse Mesh

With Algorithm 3 the first triangles are created, this initial mesh mostly completes the sphere, still some gaps between donuts can be observed, see Figure 54. In general, it can be created the triangles block for the initial Donut, however for the rest of the donuts, Is necessary to remove those triangles which have a redundant point. the number of triangles created for each combination is detailed in Table 5.

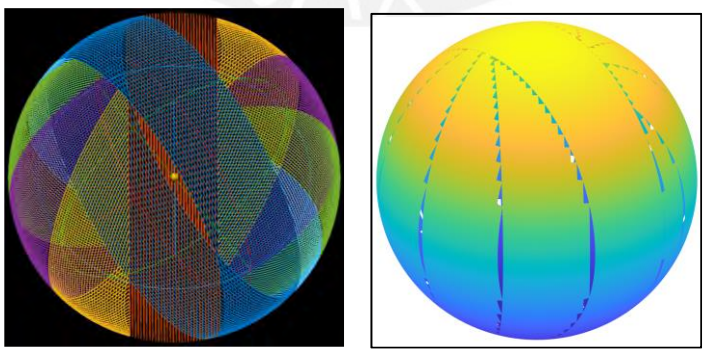


Figure 54. Coarse Mesh

PCxxKp	PC98Kp	PC196Kp		PC393Kp			PC786Kp		PC1572Kp
nBeams	16	16	32	16	32	64	32	64	64
nAzimuths	1024	2048	1024	4096	2048	1024	4096	2048	4096
Triangles generated	108226	216579	433259	227227	454736	909716	465430	931400	1863432

Table 5. Numbers of triangles generated in Coarse Mesh algorithm

4.2.4 Fine Mesh

With the last algorithm the remaining gaps are filled, and the last numbers of triangles is obtained. For this stage, the triangles generated are just a few in comparison with the triangles generated in Coarse Mesh, see Table 6 to see this percent of triangle generated. In the next figure the two mesh created are illustrated, Coarse Mesh at the left and Fine Mesh at the right, with the combination of both the final Mesh Sphere is generated.

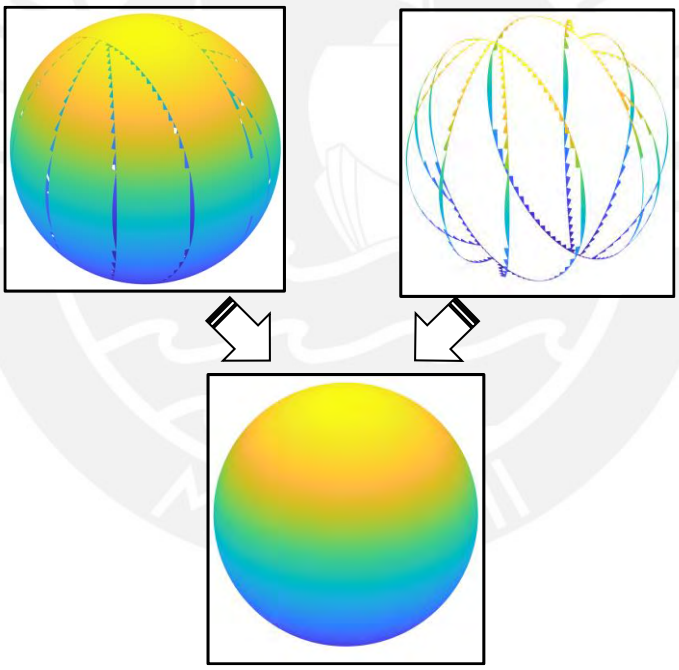


Figure 55. Combination of both mesh, Coarse and Fine mesh

PCxxKp	PC98Kp	PC196Kp		PC393Kp			PC786Kp		PC1572Kp
nBeams	16	16	32	16	32	64	32	64	64
nAzimuths	1024	2048	1024	4096	2048	1024	4096	2048	4096
Triangles generated	10036	19943	39757	10195	20114	39970	10530	20456	40348
percent (FMT/(CMT+FMT)) (%)	8.49	8.43	8.41	4.29	4.24	4.21	2.21	2.15	2.12

Table 6. Details of triangles generated in Fine Mesh

In the table above, it can be deduced that Fine Mesh could not require parallel processing, because the number of triangles generated for the total mesh in less than 8.5%. Also, design a parallel algorithm for Fine Mesh algorithm would be not necessary, since Coarse Mesh requires all the time processing of the mesh generation and the divergence in Fine Mesh implies that a parallel algorithm can has negative outcome.

4.2.5 ORaMG test running times

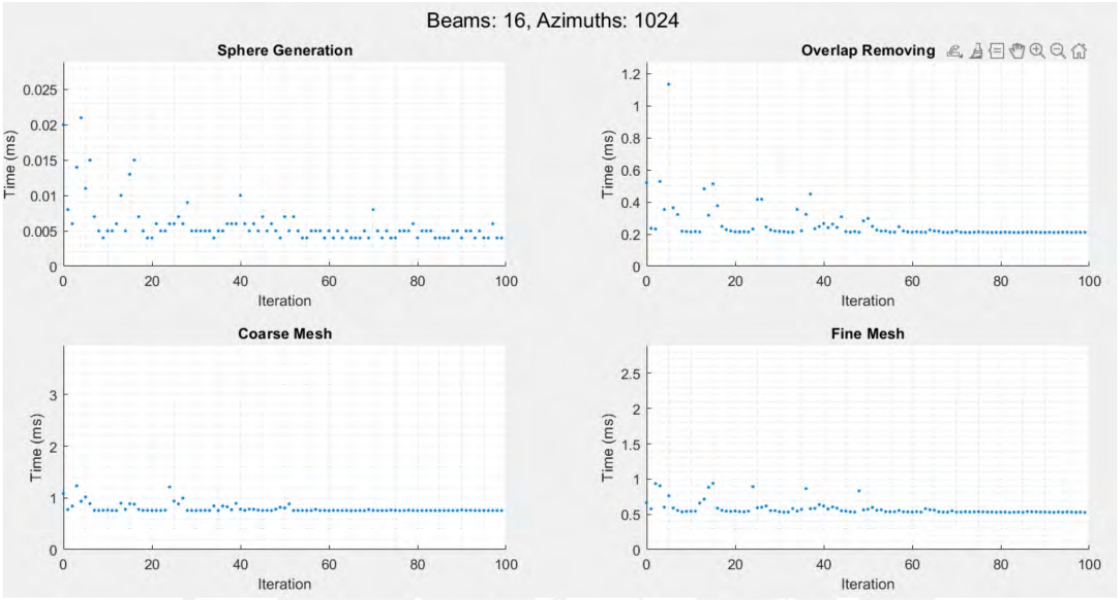
Finally, the running time test for each stage and the mesh information of triangles for Overlap Removing and Mesh Generation (ORaMG) is gathered in Table 7. Below each PCxxKp the number of beams and azimuths per donut is noted. In all cases, a full sphere (6 donuts) is scanned.

Detail	PC98Kp	PC196Kp		PC393Kp			PC786Kp		PC1572Kp
	16 1024	16 2048	32 1024	16 4096	32 2048	64 1024	32 4096	64 2048	64 4096
Sphere Gen. (ms)	0.005778	0.013404	0.014879	0.008131	0.016606	0.016626	0.010657	0.018091	0.01701
Overlap Remov. (ms)	0.254889	0.548717	0.878091	0.425758	0.925899	1.606555	0.805687	1.591171	2.707091
Coarse Mesh (ms)	0.790646	1.641637	3.145495	1.605859	3.410101	6.649455	3.252383	6.763242	12.868264
Fine Mesh (ms)	0.579232	1.275111	2.216808	0.973687	2.032879	3.828625	1.58101	3.174	6.181291
Total time (ms)	1.630545	3.478869	6.255273	3.013435	6.385485	12.101261	5.649737	11.546504	21.773657
Initial points	98304	196608	393216	196608	393216	786432	393216	786432	1572864
Points removed	39171	78345	156706	77895	155789	311587	155234	310502	620972
Percent removed (%)	39.85	39.85	39.85	39.62	39.62	39.62	39.48	39.48	39.48
Points remaining	59133	118263	236510	118713	237427	474845	237982	475930	951892
CM triangles gen.	108226	216579	433259	227227	454736	909716	465430	931400	1863432
FM triangles gen.	10036	19943	39757	10195	20114	39970	10530	20456	40348
Total Triangles gen.	118262	236522	473016	237422	474850	949686	475960	951856	1903780

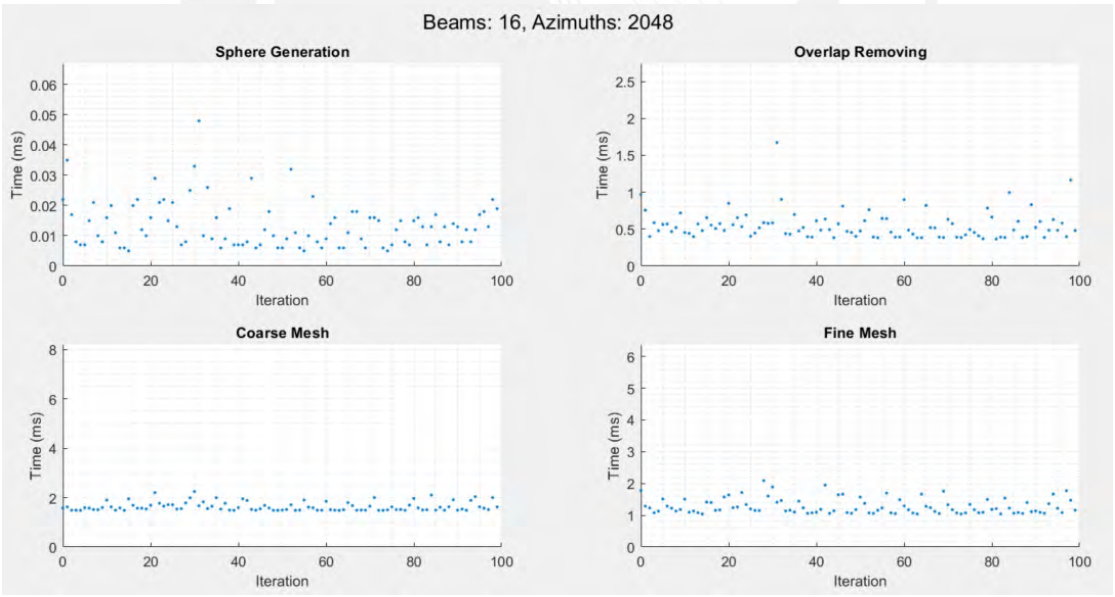
Table 7. ORaMG results

Something relevant from Table 7 is the execution time for all combinations. The LiDAR device takes 100ms just to do one Donut scan, that means that a sphere scan takes 600ms. These results show that real time can be achieved and are notably below the threshold time, even in the group with the maximum amount of points the running test just takes 21.7 milliseconds which is still below than 600 milliseconds.

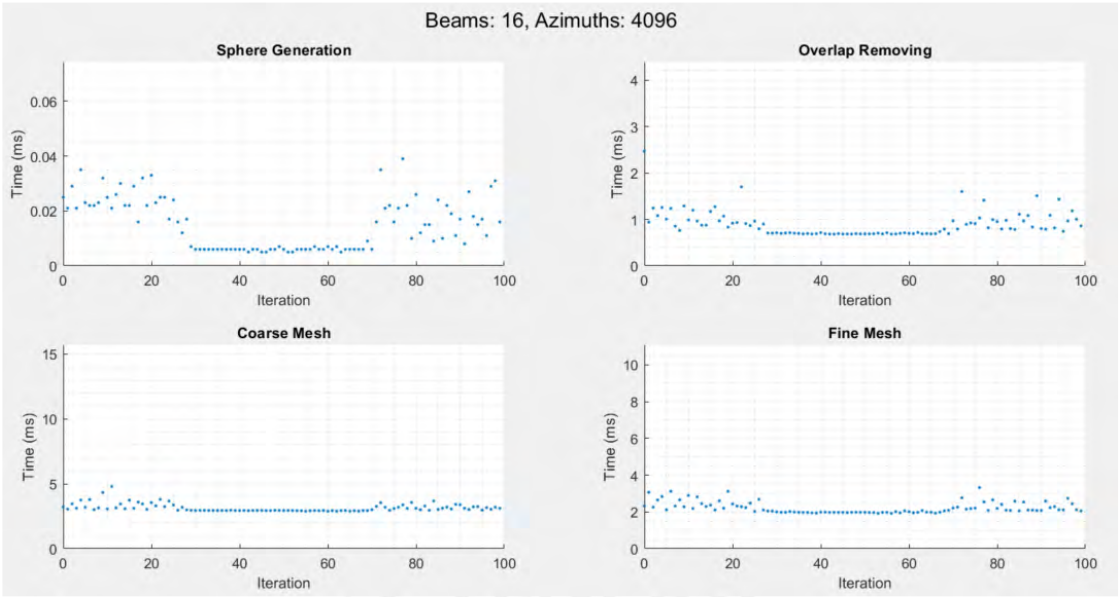
To obtain the results on Table 7, 100 iterations are done, the first iteration is discarded and mean for the rest is done. In the next figure, the value for each iteration for the 4 algorithms of ORaMG is illustrated.



(a) iterations of 16 beams and 1k azimuths

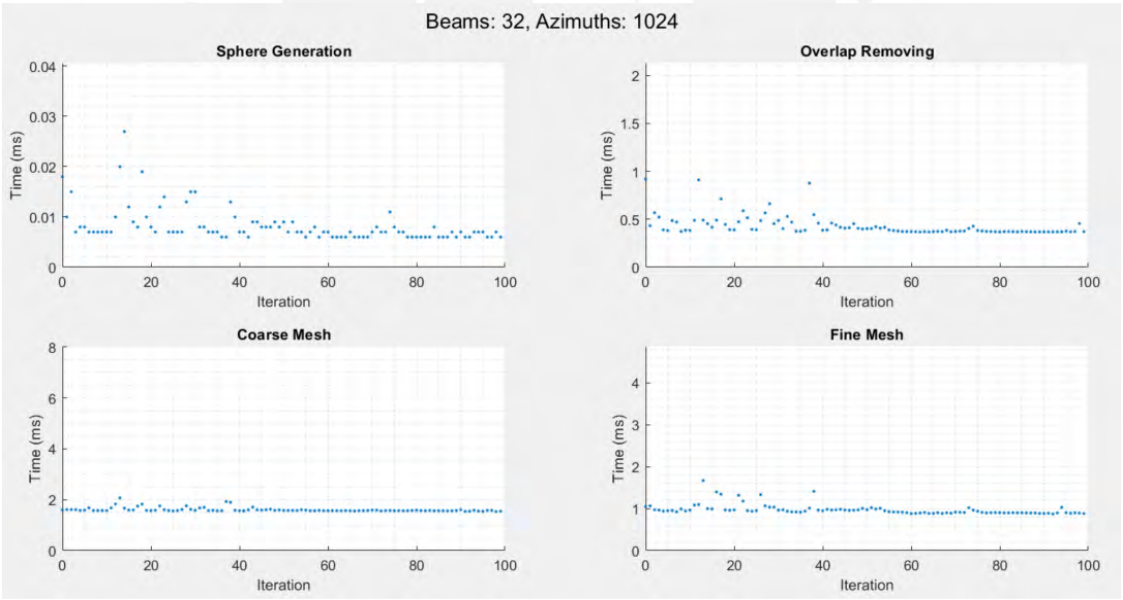


(b) iterations of 16 beams and 2k azimuths

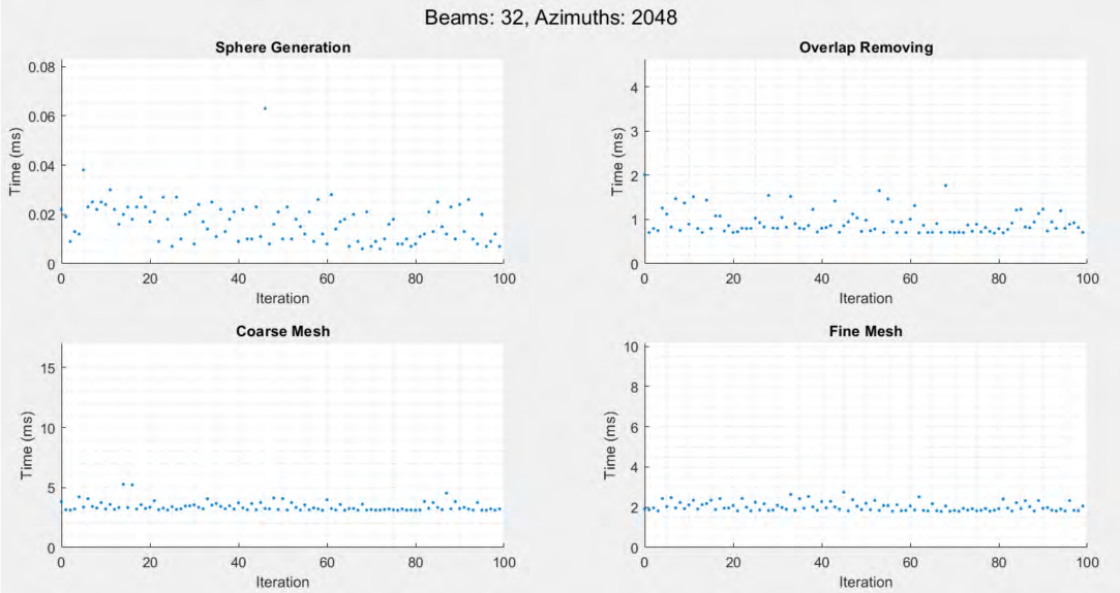


(c) iterations of 16 beams and 4k azimuths

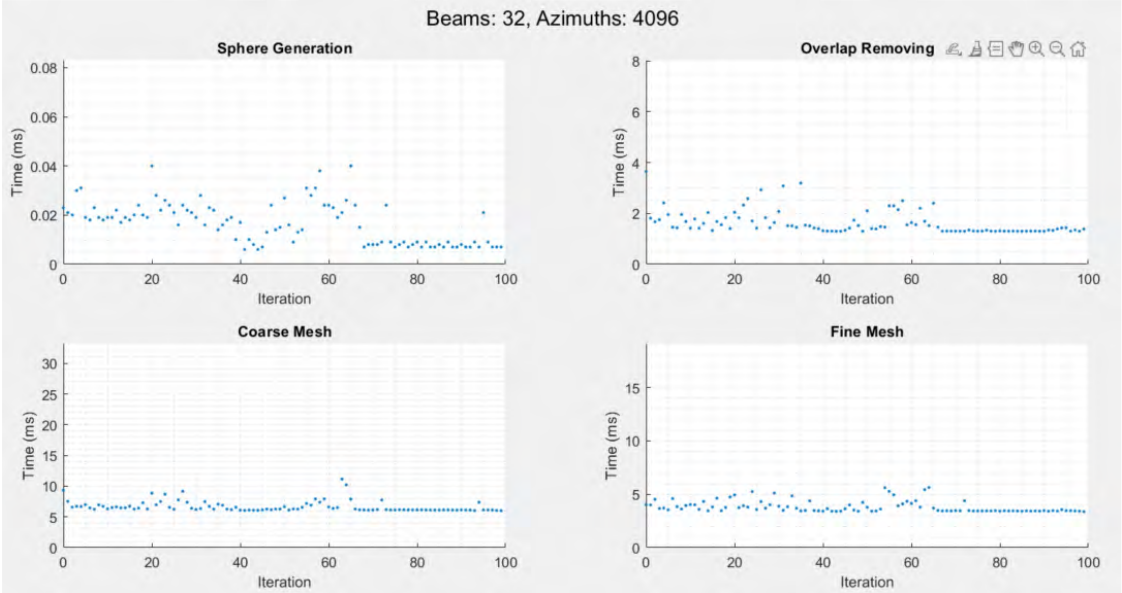
Figure 56. Iterations for 16 beams and 1k,2k,4k azimuths block



(a) iterations of 32 beams and 1k azimuths

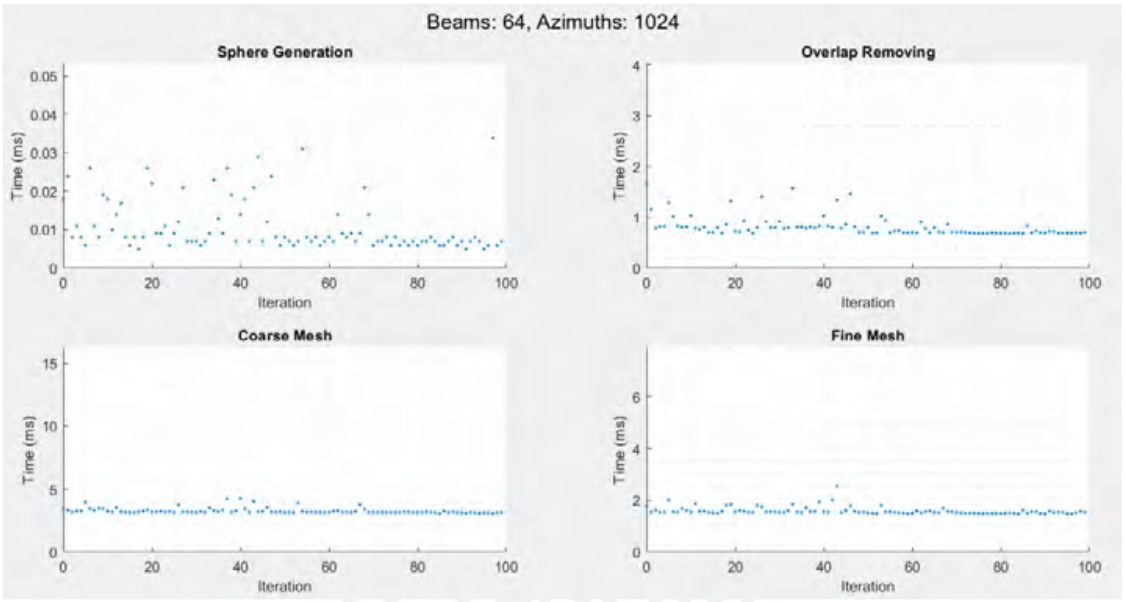


(b) iterations of 32 beams and 2k azimuths

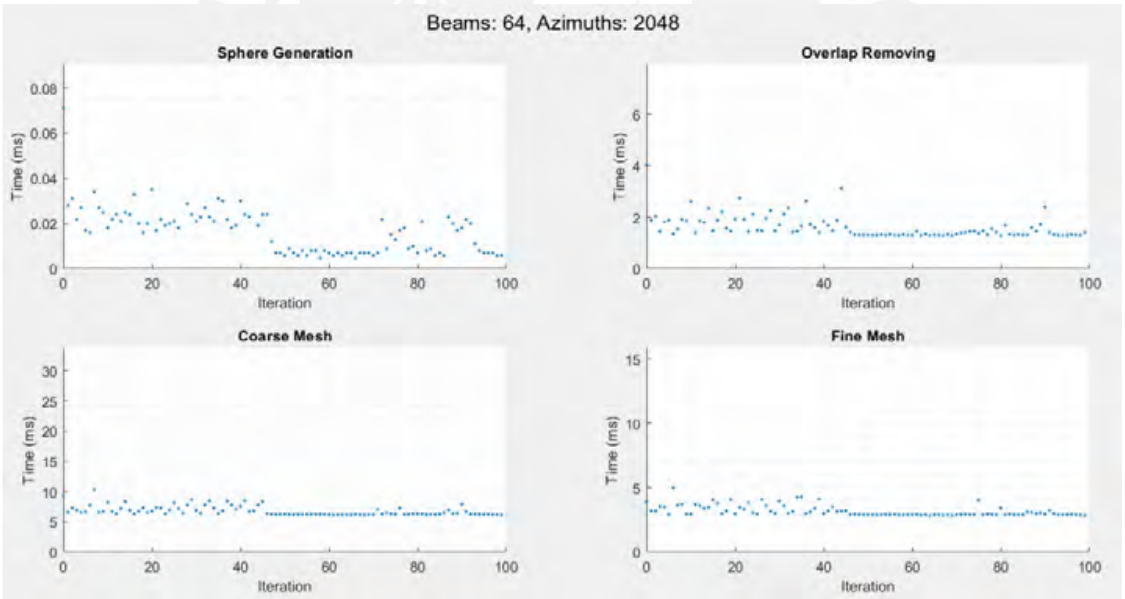


(c) iterations of 32 beams and 4k azimuths

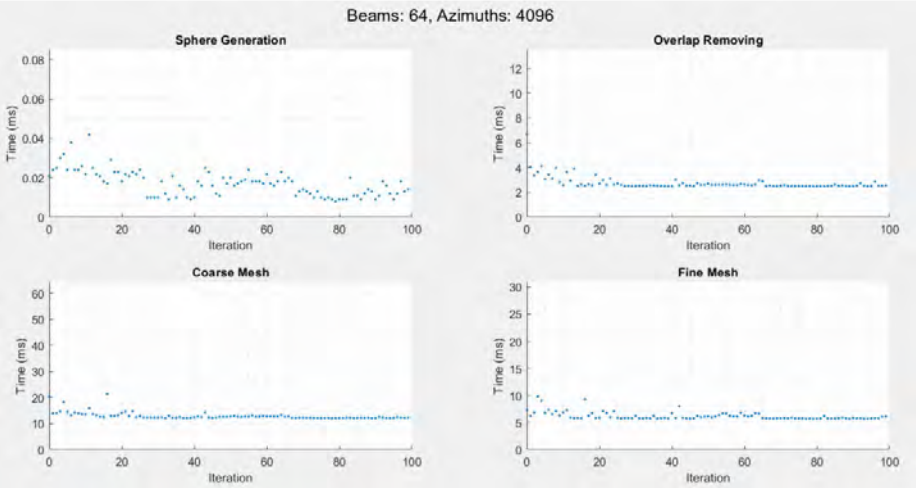
Figure 57. Iterations for 32 beams and 1k,2k,4k azimuths block



(a) iterations of 64 beams and 1k azimuths



(c) iterations of 64 beams and 2k azimuths

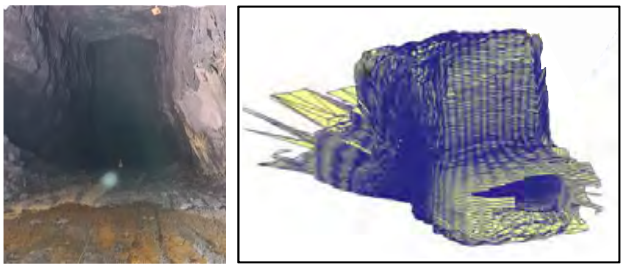


(c) iterations of 64 beams and 4k azimuths

Figure 58. Iterations for 64 beams and 1k,2k,4k azimuths block

4.3 Field test

For the second experiment, the algorithm is tested with real data. Inside an underground mining gallery, the system device is inserted to obtain the data, see Figure 59a. In this field test, the LiDAR device only has the combination of 16-1024 and to represent a low power consumption scenario the Jetson Nano device is used. The result of the running test time and mesh information is shown in Table 8. It can be observed that the amount of points reaming is lower than the synthetic sphere. This is because the sensor cannot scan its bottom and some wet surfaces in the mining does not allow to scan more points. However, the result gives us approximately an identical virtual model.



(a) *(b)*

Figure 59. Mining gallery and its virtual model

Detail	PC98Kp
	16 1024
Sphere Gen. and Overlap Rem. (ms)	1.387051
Coarse Mesh (ms)	3.598828
Fine Mesh (ms)	3.864254
Total time (ms)	8.850133
Initial points	98304
Points removed	50181
Percent Removed (%)	51.05
Points remaining	48123
Total triangle gen.	95143

Table 8. Field test results

In the technical details the desktop computer has a high performance for my algorithm, because the amount of CUDA core is 2304. In the case of the Jetson Nano, only 128 CUDA cores are available (low power consumption). However, the running time in my field test demonstrate that with a low power embedded system it can still obtain real time results. In the table above the total time for my mesh generation is still below 600 ms. In addition, both execution times are compared in the next table, the results are promising.

Detail	Running time	Power comsuption
Desktop (High perfomance)	1.630545 ms	215 Watts
Jetson Nano (low power consumption)	8.850133 ms	5 Watts

Table 9. Comparison jetson nano with desktop computer

4.4 Design cost

Finally, in the next table is exposed the cost of the hardware applied and the hours dedicated to this work, some devices were bought in dollars in an exchange rate of S/3.8 per each dollar but the values in the table are in the local currency of Peru (/S).

Detail	Quantity	Cost by quantity	Cost
Hours	640	60	38400
Lidar	1	1	13300
Desktop	1	1	5600
Jetson Nano	1	1	224.2
Motor structure	1	1	3500
Others (Movibility, safety cloths, ...)			400
Total cost			**61424.2**

Table 10. Design cost

5 Conclusions

In summary, a method to design a triangle mesh generation parallel algorithm from a particular point cloud in real time using low power consumption embedded system is developed in this work. The point cloud is obtained using the rotatory system where the angle of the motor is variable and works for any 360° LiDAR 3D. This algorithm can generate mesh for different coherent values of the angle of the motor. Also, with the considerable time remaining obtained gives the possibility to do other point cloud processing such odometry.

5.1 Objectives achieved

- The mathematical model for the scan pattern for 360° LiDAR sensors is achieved. In particular, the model applied for the experiments considered that the points obtained were previous sorted, but after this sort process the mathematical model can be applied for any 360° LiDAR.

- A sequential algorithm is also done which was the basis for the design of the parallel algorithm. This algorithm was developed in MATLAB to analyzed the concept of the idea in order to design the parallel algorithm.

- The design of a parallel algorithm (ORaMG) to merge point clouds and create an initial mesh using a priori information of the scan pattern of the points. Only the important stages were parallelized. Also, the method of the design is detailed in order to create similar algorithm but for other 3D LiDAR with different pattern of scanning and different rotation system.

- This algorithm was implemented in a GPU, where the most repetitive and most time complexity task were distributed to the CUDA cores. This algorithm was tested in a low power consumption embedded system using real data obtained from a mine in real time processing time. In particular, for a LiDAR scan that takes 600ms, the algorithm takes approximately 8ms for processing and obtain the final result, it means least that the 2% time of obtaining the data.

The results obtained in section 4 validate that the proposed thesis is able to achieve real time processing. Also, this proves that keeping the neighbor relationship of points in a point cloud reduce significantly the computational complexity of the mesh generation

5.2 Recommendations

In order to replicate this algorithm for other LiDAR devices, it is highly recommend: First, get the mathematical model of the scan pattern, all the LiDAR follows a scan pattern that can be obtained by Fourier analysis or predictive methods, the precision does not have to be so exact, a minimal error can be considered. Second, generate a synthetic data, this could be a sphere in a 3D plane or a circle for 2D scans. Third, create an initial, the Coarse Mesh, for this it is recommend to classify the points (stablish an array with their spatial information such as azimuth angle, quadrant position, etc.). And finally, a Fine Mesh generation to fill the gaps remaining on the Coarse Mesh.

In addition, try to parallelize only the algorithms that can give a considerable change on the time processing. For this, divide the code in different stages and analyze which one takes more processing time and parallelize it.

Also, in some cases is necessary to use the module operator but if the number of the second parameter is a power of two, the logical operator “mask” can be used instead of using the module operator. For instance, in this work only the range 0 to 16383 was used, all numbers above like 16384 were masking to keep staying in the range.

5.3 Possible applications

The ORaMG algorithm has significatively reduce the processing time for a mesh generation. At least, 90% of time is free to used. The remaining time can be used for other applications. This gives the possibility to do other point cloud processing such mapping, odometry, improve the mesh, etc. For instance, ORaMG does not give an isotropic mesh, but gives an initial mesh for post processing. In collaboration with other author (Huapaya C.) an algorithm for the improvement of the initial mesh was created, also in a real time processing. This method used a voxel structure downsampling which in combination with ORaMG, a mesh system is designed in order to obtain an isotropic mesh generation in real time, see Figure 60.

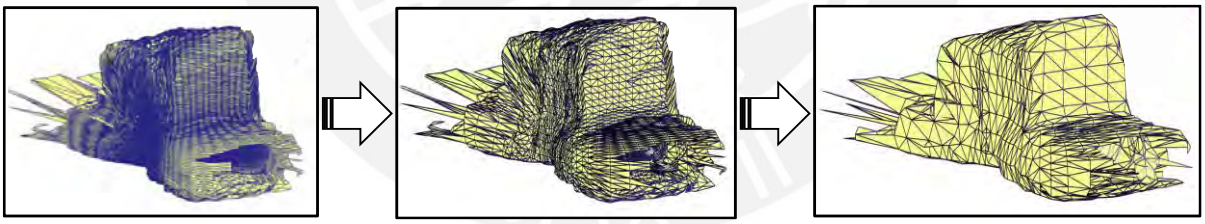


Figure 60. Real time voxel downsamplig mesh system

Finally, a future work application is generalizing the method for any type of LiDAR scan. The basic idea is to find the scan pattern to reduce the local region processing, remove the redundant points, generate an initial CoarseMesh and finally fill the gaps with a FineMesh.

6. References

[1] H. V. Leydy Muñoz Edilson Quiñónez, "Reconstrucción 3D de objetos sumergidos en aguas limpias," *Ingeniería*, vol. 18, no. 2, pp. 36, 53, 2013.

[2] M. A. Quintana Rosales, "Registro de una secuencia temporal de nubes de puntos utilizando tecnologia Kinect para la reconstrucción tridimensional de material arqueologico," Pontificia Universidad Católica del Perú, 2014.

[3] Y. Zhang and J. Liu, "Application of 3D laser scanning technology in structural design of key parts of marine port machinery," *J. Intell. Fuzzy Syst.*, vol. 38, no. 2, pp. 1273–1279, 2019, doi: 10.3233/jifs-179490.

[4] W. Qingshan and Z. Jun, "Point Cloud Registration Algorithm Based on Combination of NDT and PLICP," *Proc. - 2019 15th Int. Conf. Comput. Intell. Secur. CIS 2019*, pp. 132–136, 2019, doi: 10.1109/CIS.2019.00036.

[5] L. Hui-ying, Y. Chao, W. Guan-liang, L. Wen-hui, and L. Cai, "A HIERARCHICAL CONTOUR METHOD FOR AUTOMATIC 3D CITY RECONSTRUCTION FROM LIDAR DATA 1 . College of Computer Science and Technology , Jilin University , Changchun , P . R . China , 130012 ; 2 . College of Resources and Civil Engineering , Northeastern Unive," pp. 463–466, 2012.

[6] X. F. Han, J. S. Jin, M. J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3D point cloud," *Signal Process. Image Commun.*, vol. 57, no. October, pp. 103–112, 2017, doi: 10.1016/j.image.2017.05.009.

[7] X. Chen, Q. Wu, and S. Wang, "Research on 3D reconstruction based on multiple views," *13th Int. Conf. Comput. Sci. Educ. ICCSE 2018*, no. Iccse, pp. 269–273, 2018, doi: 10.1109/ICCSE.2018.8468705.

[8] Q. G. Tian and J. T. Li, "Pre-processing of 3D scanning line point cloud data," *ICCASM 2010 - 2010 Int. Conf. Comput. Appl. Syst. Model. Proc.*, vol. 10, no. Iccasm, 2010, doi: 10.1109/ICCASM.2010.5622877.

[9] S. Falip and D. Del Cogliano, "Generation of 3D Point Clouds with Terrestrial Laser Scanner. Georeferencing and Quality Evaluation Generación de Nubes de Puntos 3D con Escáner Laser Terrestre. Georreferenciación y Evaluación de la Calidad," *Ingeniería*, vol. 24, no. 2, pp. 171–197, 2019, [Online]. Available: https://doi.org/10.14483/23448393.14542.

[10] M. T. Ahmed, J. A. Marshall, and M. Greenspan, "Point cloud registration with virtual interest points from implicit quadric surface intersections," *Proc. - 2017 Int. Conf. 3D Vision, 3DV 2017*, pp. 649–657, 2018, doi: 10.1109/3DV.2017.00079.

[11] J. Liu and X. Q. Gong, "Study on the alignment method of multi-view point-clouds in reverse engineering," *Proc. - 2010 3rd Int. Congr. Image Signal Process. CISP 2010*,

vol. 5, pp. 2360–2362, 2010, doi: 10.1109/CISP.2010.5648243.

[12] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson Surface Reconstruction,” *Eurographics Symp. Geom. Process.*, 2006, doi: 10.2312/SGP/SGP06/061-070.

[13] S. Dahu and Z. Li, “A fast surface reconstruction algorithm based on Delaunay,” *Proc. - 2012 Int. Conf. Comput. Sci. Inf. Process. CSIP 2012*, pp. 981–984, 2012, doi: 10.1109/CSIP.2012.6309020.

[14] H. Li, P. Xu, and Y. Shen, “A self-adaption fast point cloud simplification algorithm based on normal eigenvalues,” *Proc. - 2014 7th Int. Congr. Image Signal Process. CISP 2014*, pp. 852–856, 2014, doi: 10.1109/CISP.2014.7003896.

[15] S. Orts-Escolano, V. Morell, J. Garcia-Rodriguez, and M. Cazorla, “Point cloud data filtering and downsampling using growing neural gas,” *Proc. Int. Jt. Conf. Neural Networks*, pp. 1–8, 2013, doi: 10.1109/IJCNN.2013.6706719.

[16] B. Zou, H. Qiu, and Y. Lu, “Point Cloud Reduction and Denoising Based on Optimized Downsampling and Bilateral Filtering,” *IEEE Access*, vol. 8, pp. 136316–136326, 2020, doi: 10.1109/ACCESS.2020.3011989.

[17] O. Ervan and H. Temeltas, “Downsampling of a 3D LiDAR Point Cloud by a Tensor Voting Based Method,” *ELECO 2019 - 11th Int. Conf. Electr. Electron. Eng.*, pp. 880–884, 2019, doi: 10.23919/ELECO47770.2019.8990544.

[18] Ouster, “Software User Guide Release v1.13.0,” pp. 1–60, 2019.

[19] Stanley I. Grossman, *Algebra Lineal*. 2014.

[20] M. Erwig, “The graph Voronoi diagram with applications,” *Networks*, vol. 36, no. 3, 2000, doi: 10.1002/1097-0037(200010)36:3<156::AID-NET2>3.0.CO;2-L.

[21] M. Bin Chen, “A divide-and-conquer algorithm of Delaunay triangulation with GPGPU,” *Proc. - Int. Symp. Parallel Archit. Algorithms Program. PAAP*, pp. 175–177, 2012, doi: 10.1109/PAAP.2012.50.

[22] Y. Chao, T. Wu, X. Wang, and G. Zheng, “The computation of delaunay triangulation of LiDAR point cloud based on GPU,” *Int. Conf. Geoinformatics*, vol. 2016-Janua, pp. 1–4, 2016, doi: 10.1109/GEOINFORMATICS.2015.7378671.

[23] J. JáJá, *An introduction to parallel algorthims*. 1992.

[24] NVIDIA, “CUDA Toolkit 7.5 Documentation,” *Developer Zone*, 2015. .

[25] C. Lv, W. Lin, and B. Zhao, “Voxel Structure-based Mesh Reconstruction from a 3D Point Cloud,” *IEEE Trans. Multimed.*, 2021, doi: 10.1109/TMM.2021.3073265.