UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

Título: Triangulación de Delaunay en puntos tridimensionales de superficie.

Trabajo de integración curricular presentado como requisito para la obtención del título de Matematico.

Autor:

Tigmasa Pico Hipatia Nahomy

Tutor:

Ph.D. Antón Castro Francesc

Urcuquí, Noviembre 2022

Autoría

Yo, **Tigmasa Pico Hipatia Nahomy**, con cédula de identidad 1804965018, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, noviembre del 2022.

HIPATIA
NAHOMY
TIGMASA PICO

Firmado digitalmente por
HIPATIA NAHOMY TIGMASA
PICO
Fecha: 2023.08.13 15:27:23
-05'00'

Hipatia Nahomy Tigmasa Pico

CI: 1804965018

Autorización de publicación

Yo, **Tigmasa Pico Hipatia Nahomy**, con cédula de identidad 1804965018, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, noviembre del 2022.

HIPATIA
NAHOMY
TIGMASA PICO

Firmado digitalmente por
HIPATIA NAHOMY TIGMASA
PICO
Fecha: 2023.08.13 15:27:47
-05'00'

Hipatia Nahomy Tigmasa Pico
CI: 1804965018

Dedication

To my parents, Hipatia and Victor.
I love you both with all my heart.

A mis padres, Hipatia y Victor.
Los amo con todo mi corazon.

Hipatia Nahomy Tigmasa Pico

Agradecimientos

I am grateful for the unconditional support of my parents, Hipatia and Victor, throughout my life; without you, none of this would be possible. Thank you for love and work you put into raising me, the opportunities you created for me, and for understanding me. To Angie Pico for the help, she gave me without asking for recognition. You have been an inspiration throughout my life. To Danilo Campi for being my safe place and a true friend. I have a site I belong to, thanks to you. Thank you for your honesty and support.

To Francesc Anton for his patience, instructions, and availability at all times. I am truly grateful.

Hipatia Nahomy Tigmasa Pico

Resumen

El presente trabajo de investigación se centra en el estudio y desarrollo de un algoritmo de reconstrucción de una superficie a partir de una nube de puntos obtenida con tecnología LiDAR. El algoritmo busca encontrar una medida cuantitativa de la suavidad y continuidad del terreno analizándolo en pequeñas secciones; en base a ello, se calcula la triangulación de Delaunay mediante la proyección de los puntos de cada sección a un plano o cono tangente.

Palabras Clave: Triangulacion de Delaunay, Mallas, LiDAR, Vecinos cercanos.

Abstract

The present research work focuses on studying and developing an algorithm for reconstructing a surface based on a point cloud obtained with LiDAR technology. The algorithm seeks to find a quantitative measure of the smoothness and continuity of the terrain by analyzing it in small sections; based on this, the Delaunay triangulation is calculated using the projection of the points of each section to a tangent plane or cone.

Keywords: Delaunay triangulation, Meshes, LiDAR, Nearest neighbor.

Contents

Contents **vii**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Background 2
1.2 Problem statement 4
1.3 Objectives 5
1.3.1 General Objective 5
1.3.2 Specific Objectives 5

2 Theoretical Framework **6**

2.1 Triangulation and Delaunay Triangulation 6
2.1.1 Edge flips 8
2.1.2 Delaunay triangulation properties 9
2.1.3 3D Delaunay triangulation 11
2.1.4 Algorithms for computing 3D Delaunay Tetrahedralization 12
2.2 Polygonal and triangular meshes 13
2.2.1 Mesh topology 14
2.2.2 Mesh representation 15
2.3 Surface reconstruction and Delaunay 17
2.4 LiDAR technology 19
2.4.1 LiDAR applications 20
2.4.2 LiDAR and smartphones 21

2.5 KD-tree and spatial indexing 21
    2.5.1 Binary Trees 22
    2.5.2 KD-trees 22
    2.5.3 KD-Trees Nearest Neighbor 23
    2.5.4 Spatial Indexing 24
    2.5.5 Quadtree and Octree 25
2.6 Covariance matrices and representation of surfaces 25
2.7 Tangent plane and tangent cone 27
    2.7.1 Tangent cone 28

3 Methodology **29**
3.1 Phases of Problem Solving 29
    3.1.1 Description of the Problem 29
    3.1.2 Analysis of the Problem 30
    3.1.3 Algorithm Design 32
    3.1.4 Implementation 33
    3.1.5 Testing 35

4 Results and Discussion **39**

5 Conclusions **47**

Bibliography **48**

List of Tables

2.1 Indexed triangle mesh vertices and triangles 17

3.1 Density of points in the poin cloud set 33

4.1 Nearest neighbor histogram for the whole point cloud 40
4.2 α histogram for the whole point cloud 42
4.3 Values α in Section 1 43
4.4 Values of α in Section 2 43
4.5 Values of α in Section 3 43

List of Figures

1.1 Divide and conquer algorithm. 3
1.2 Randomized incremental algorithm. 3
1.3 3D Delaunay triangulation 4

2.1 Delaunay triangulation on a small set of points. 7
2.2 Voroinoi diagram in green lines. Convex hull is represented by the orange dotted lines. 8
2.3 Delaunay edge and triangle with their circum circle 9
2.4 Edge flips and cocircular points 9
2.5 Projection of the Delaunay triangulation of 4 points on the paraboloid of revolution 10
2.6 Tetrahedra: Vertices, edges and faces (only f_1 is labeled) 11
2.7 Delaunay triangulation of 5 points and circumsphere defining a Delaunay tetrahedron 12
2.8 Polygonal and triangular mesh 14
2.9 Geometry vs Topology on meshes 14
2.10 Manifold and not manifold objects 15
2.11 Orientation of triangles in a mesh 15
2.12 Möbious band, not orientable 16
2.13 Triangular mesh made of 3 triangles in 3D space 17
2.14 LiDAR light energy used to measure objects in the ground 20
2.15 KD-tree example 23
2.16 The labeling of a Quadtree and an Octree [1, 2]. 25
2.17 The vector Field implicit representation of a surface, taken from [3]. 26

2.18 Cones of a parametric surface. (a) A regular parametric surface. (b) Two tangent cones. (c) Normal cone. (d) Visibility Cone [4] 28

3.1 Examples of irregular terrains with singularities. 30
3.2 Graphic representation of α 31
3.3 Region of Ibiza from where the LiDAR data set was taken. Image comes from the National Geographic Institute of the government of Spain's web page. 34
3.4 Data set view using Google Maps: approximate location of the point set. . 34
3.5 LiDAR data set viewed using LAStools 35
3.6 Section 1: $(x, y, z) \in [0, 400) \times [500, 100) \times \mathbb{R}$ 36
3.7 Section 2: $(x, y, z) \in [400, 800) \times [0, 500) \times \mathbb{R}$ 37
3.8 Section 3: $(x, y, z) \in [1200, 1600) \times [1000, 1500) \times \mathbb{R}$ 38

4.1 Delaunay triangulation of the projection of point i = 1 44
4.2 Delaunay triangulation of the projection of point i = 2 45
4.3 Delaunay triangulation of the projection of point i = 3 45
4.4 Delaunay triangulation of the first three points of the set and their nearest neigbors 46

Chapter 1

Introduction

In computer science, there are several important methods for triangulation, and Delaunay triangulation is one of them. It has important properties such as maximizing the minimum angle on the plane, it is dual to the Voronoi diagram, and perhaps one of the most important, the empty circumcircle criteria [5]. In the two-dimensional case, there exist several algorithms to compute the Delaunay triangulation that can be extended to the three-dimensional space or even the n-dimensional space [6]. In the three-dimensional case, the triangulation becomes a tetrahedralization that reconstructs volume.

We want to address the reconstruction of the surface of objects or terrain defined from point clouds by creating a triangular mesh with Delaunay properties. This represents quite a challenge because it is not guaranteed that the surface to be triangulated will be smooth or continuous; moreover, when the terrain has holes or cracks, it can lead to incorrect triangulation or holes in the mesh.

On the other hand, with the technological advances in phones, tablets, and digital cameras, range scanners, nowadays LiDAR technology is available for almost everyone, and so it is the software needed to use it. There are several libraries with point cloud data obtained from LiDAR scans in phones. Apple, for instance, is using it for their Apple Maps app [7] to obtain a digital 3d model of cities. Instead, Samsung uses LiDAR scanner for their JetBot 90 AI+; it uses the scanner to navigate the rooms and to fulfill its purpose of vacuuming [8].

It is important to consider this availability of new technologies and use the opportunity to develop an algorithm to reconstruct a surface from three-dimensional point clouds and obtain the correct surface topology. Once achieved an optimal an efficient implementation, this algorithm could be used for reconstructing terrain in real time. Moreover, by using LiDAR technology, we can create a system with the intention of using the notion of range of vision. As a result, we want a mesh that represents the surface topography accurately, i.e., adapting itself to the surface irregularity.

The structure of this work is as follows: later, on this same Chapter, the divide and conquer and the randomized incremental algorithm are described on the Background Subsection 1.1. Then, subsection 1.2 describes the statement of the problem this investigation work is focused. On Chapter 2, important definitions, as well as the the theoretical basis used to construct the Delaunay triangulation, are presented. Later on Chapter 3 the process for developing the algorithm using the tools described on the previous chapter is presented. Then, on chapter 4 the results obtained from the algorithm implementation are shown as well as the analysis on the algorithm performance. Finally, on chapter 5 the conclusions of this project are presented.

1.1 Background

There are several techniques with variable run time and implementation complexity for producing the Delaunay triangulation. For the two dimensional case, [9] present two algorithms: the first one based on a divide and conquer approach, takes $O(nlogn)$ time; and the second one adds points to an existing triangulation one by one. In the worst case it will take $O(n^2)$ time. A better description on the *Divide and Conquer Algorithm*, is described on [10]. It is a recursive algorithm such that given a set of points P, if $|P| > 3$, it splits the set P into two sets, P_L and P_R and they become the new input for the next iteration. If $|P| = 3$ it forms a triangle, if $|P| = 2$ it creates a line; then it creates new triangles between the sets and checks if the triangulation is correct as shown bellow.

It is an efficient algorithm for the two dimensional Delaunay triangulation, and even though it can be generalized to higher dimensions, it is not any more the only optimal algorithm.

On [5], they describe the *Randomized Incremental Algorithm* which consists on adding two points, p_{-1} and p_{-2} at random. Then together with the highest point $p_0 \in P$, maximum

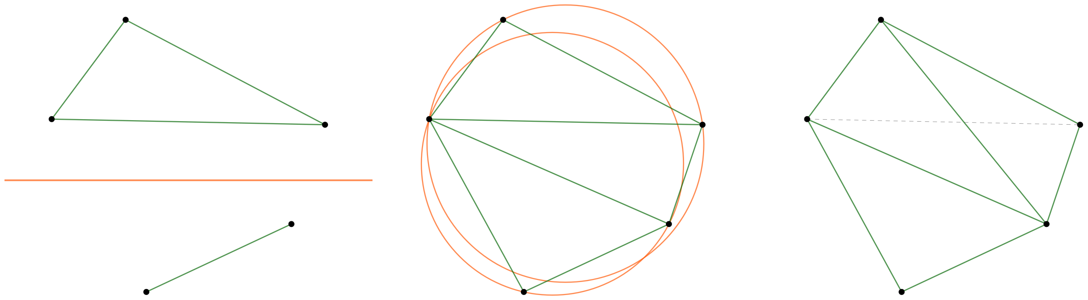


Figure 1.1: Divide and conquer algorithm.

value on either x or y coordinate; this points form a triangle containing $P \backslash \{p_0\}$. For this, p_{-1} and p_{-2} need to be far enough so that once computed the Delaunay triangulation of $P \cup \{p_{-1}, p_{-2}\}$, eliminating this two points wont affect the Delaunay triangulation of P. The triangulation is obtained by adding the points of $P \backslash \{p_0\}$ on a random order in the triangulation, fixing the triangulation by legalizing the edges of Delaunay and finally eliminating p_{-1} and p_{-2} As it is shown in figure1.2.

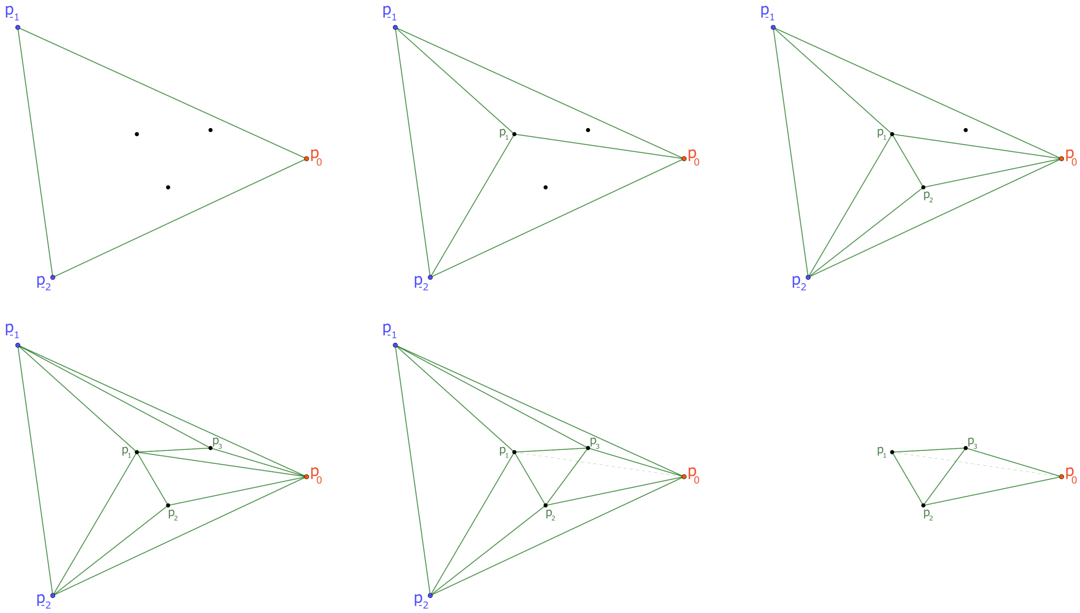
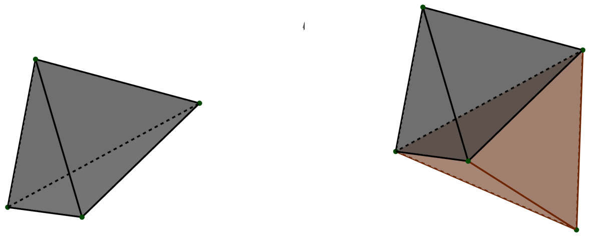


Figure 1.2: Randomized incremental algorithm.

Using this algorithm, the Delaunay triangulation on a plane can be computed in $O(nlogn)$ expected time, using $O(n)$ expected storage. Another option is also computing the Voronoi diagram and then calculate its dual, the Delaunay triangulation, which is achievable in linear time [11].

On the other hand, for the three dimensional case of the Delaunay triangulation, there are some considerations that must be taken since the properties of the Delaunay triangulation on a three or higher dimensional space are not as good as in the two dimensional case [6]. The three dimensional case for a triangulation will need at least four points to create the corresponding tetrahedron, (see figure 1.3).



(a) Delaunay tetrahedron (b) 3D Delaunay triangulation of 5 points

Figure 1.3: 3D Delaunay triangulation

According to [12], the algorithms for computing the Delaunay triangulation are divided in categories:

- *Local improvement algorithms*, which triangulate the set of points in an arbitrary way and then flip the incorrect edges so they become Delaunay.
- *Incremental insertion*, which is the same idea as in Randomized Incremental Algorithm, adding points one by one and updating the triangulation.
- *Higher dimension embedding* which uses a lifting map to a higher dimension, in the case of a 2 dimensional point set, the map leads to a paraboloid of revolution .

The issue with these algorithms is that they are thought for a volumetric reconstruction while for the case of surface, reconstruction we have another whole story, see section 2.3.

1.2 Problem statement

We seek to develop an algorithm for the surface reconstruction of a finite set of points (a point cloud) in a three-dimensional space. This algorithm must be able to maintain the topological relationships between the points in the plane, determining quantitatively how close the surface of the terrain corresponds to a roughly flat area or to a nearly singular surface, in order to triangulate efficiently with the correct technique and obtaining an accurate reconstruction of the terrain's topography. With this considerations, the objectives are described on section 1.3

1.3 Objectives

1.3.1 General Objective

Construct an algorithm for a Delaunay triangulation of a surface using a finite three dimensional set of points.

1.3.2 Specific Objectives

- Research different approaches to achieve the Delaunay triangulation of a surface using the information on the smoothness of the terrain.
- Choose a programming language to implement the algorithm for the surface Delaunay triangulation.
- Learn to obtain and filter LiDAR files containing three-dimensional points, corresponding to a terrain.

Chapter 2

Theoretical Framework

Throughout this chapter we will describe the theoretical basis on which the algorithm was developed; as well as useful information on the data used to perform the triangulation and important definitions.

2.1 Triangulation and Delaunay Triangulation

Before defining a triangulation [10], we must first introduce the notation to be used for a triangle, its vertices and its edges.

Definition 1 *A triangle t is defined as a geometric entity consisting on 3 different vertices, $V = \{v_1, v_2, v_3\}$, joined by 3 different edges (that are straight line segments), $E = \{e_1, e_2, e_3\}$, respectively, as shown in Figure 2.1. Notice that the vertices are not co-linear.*

Definition 2 (Triangulation) *Let P be a finite set of points. The triangulation $\mathcal{T}$ of P consists on a set of triangles t_i, $i = 1, ..., n$ such that:*

- *All points $p_i \in P$ are vertices on at least one triangle t_i.*
- *For $p_i \in P$ there exists at least one triangle $t_i \in \mathcal{T}$ such that $p_i \cap t_i = p_i$ i.e. p_i intersects the triangles at their vertices.*
- $\forall t_i, t_j \in \mathcal{T} : \quad \mathring{t}_i \cap \mathring{t}_j = \emptyset$ *i.e. the interiors of the triangles do not intersect*
- $\forall t_i, t_j \in \mathcal{T}, \exists e_k : \quad t_i \cap t_j$ *is either $\emptyset$, an edge e_k or a vertex v_i i.e. the triangles intersect at their edges or vertices.*

See Figure 2.1 for reference

The points to be triangulated must not be colinear, otherwise they will form a degenerated triangle.

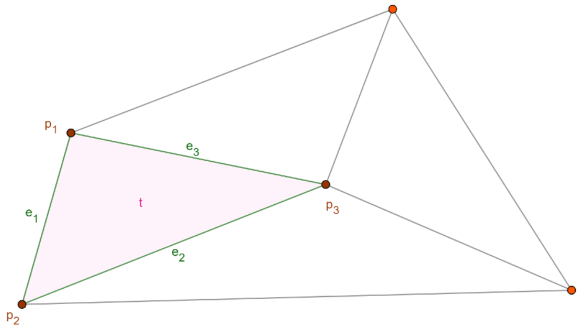


Figure 2.1: Delaunay triangulation on a small set of points.

For the purpose of a better understanding on the Delaunay triangulation, we will briefly define the Voronoi diagram based on the idea given in [9]. Given a set of non-colinear points V such that $|V| \geq 3$, one can imagine Voronoi polygons as the building blocks of a regular growth process. Each $v \in V$ will represent the nucleus of a single cell, and a cell's growth will halt when its border touches the border of another cell that is also expanding. The remaining cells will have divided the interior of the convex hull of V into a collection of non-overlapping closed convex polygons. By the time the process is complete, these polygons, together with the open polygons on the convex hull, define a Voronoi tessellation. See Figure 2.2.

From the Voronoi diagram, if the pairs of generator points whose Voronoi polygons share an edge are joined, a second tessellation is obtained, this tessellation is Delaunay, as long as the points are not cocircular. If not, when 4 points are cocircular we call it a Delaunay pretriangulation, which we can partition into triangles by non-intersecting line segments [13]. Once all the tessellation consists of triangles, we call it Delaunay triangulation. In Figure 2.2, we can see this construction from the Voronoi tessellation.

More formally we have the following definitions for the Delaunay triangulation, as well as the definitions of the empty circumcircle and boundary edge.

Definition 3 (Empty circumcircle of an edge) *Let $p_i, p_j \in P$. A circumcircle of an edge going from p_i to p_j, is a circle passing through p_i and p_j. This circumcircle is said to be empty if, and only if, its interior contains no other point $p_k \in P$.*

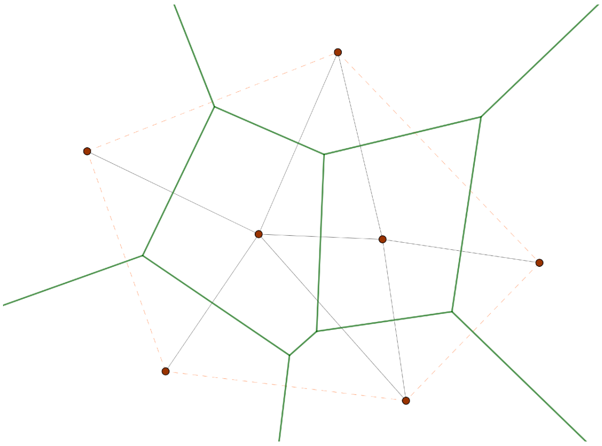


Figure 2.2: Voroinoi diagram in green lines. Convex hull is represented by the orange dotted lines.

Definition 4 (Delaunay edge) *Let P be a finite set of points and e_i an edge from p_i to p_j with $p_i, p_j \in P$. An edge e_i is Delaunay if, and only if, either it has an empty circumcircle or it is a boundary edge.*

In Figure 2.3a, both e_1 and e_2 are Delaunay edges. In the case of e_1, its circumcircle contains no other element of P. In the case of e_2, it has another point of P, however it corresponds to a boundary edge so it is still a Delaunay edge.

Definition 5 (Delaunay triangle) *A triangle is Delaunay if and only if its circumcircle is empty, i.e. the unique circle passing through its three points contains no point of P in its interior.*

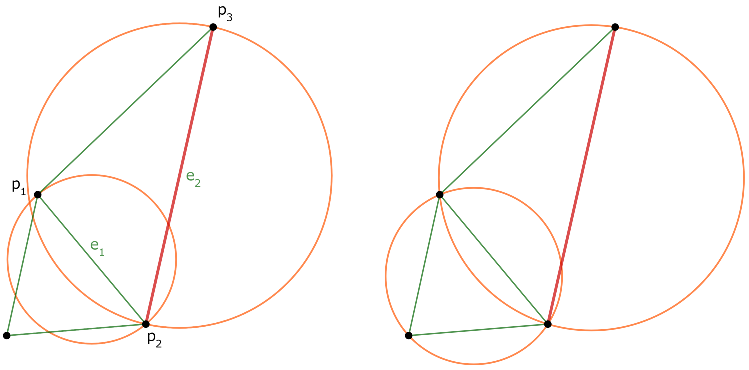
See Figure 2.3b. Both triangles are Delaunay.

Definition 6 (Delaunay triangulation) *A triangulation, $\mathcal{T}$, is a Delaunay triangulation if, and only if, all the edges in $\mathcal{T}$ are Delaunay.*

It is quite obvious that all triangles in a Delaunay triangulation are Delaunay too.

2.1.1 Edge flips

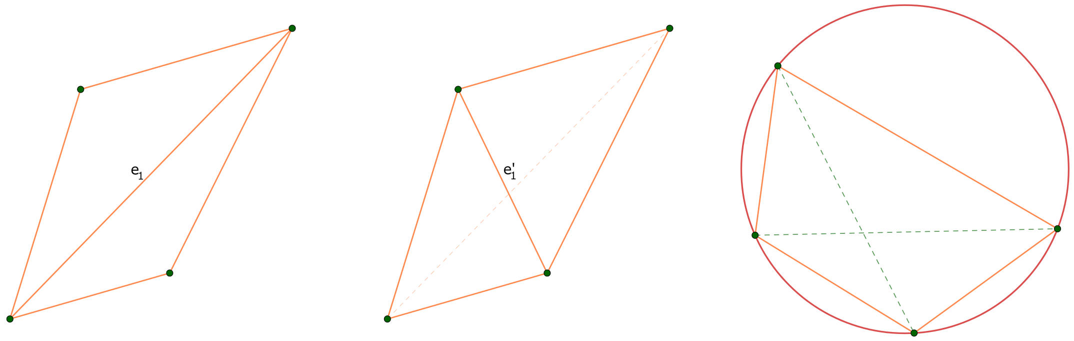
An edge flip is a useful tool used to correct non-Delaunay edges in a triangulation such that they become Delaunay. In figure 2.4a, e_1 is not Delaunay, hence the triangulation is not a Delaunay triangulation; in order to obtain the Delaunay triangulation an edge flip is necessary and we will end up with a Delaunay triangulation as in Figure 2.4b.



(a) Delaunay edges (b) Delaunay triangles

Figure 2.3: Delaunay edge and triangle with their circum circle

Sometimes, it is possible to find four cocircular points in a triangulation, in that case we can still obtain a triangular structure since it will not affect which internal edge is formed, see Figure 2.4c.



(a) Edge e_1, not Delaunay (b) Edge e_1 flipped to e_1' (c) Cocircular points

Figure 2.4: Edge flips and cocircular points

2.1.2 Delaunay triangulation properties

There are several wonderful qualities of the Delaunay triangulation. These characteristics make the Delaunay triangulation the preferred triangulation in many fields, such as computer graphics where skinny triangles are avoided. Some of these attributes include:

It maximizes the minimum angle over all triangles. When some triangles become too skinny, one of their angles becomes too small, that leads to numerical instability [10], and in the cases of rendering, it makes the process more complex and can lead to errors.

However, the Delaunay triangulation does not necessarily minimize the maximum angle [14] nor necessarily minimize the length of the edges.

Subgraphs of Delaunay triangulations. The Delaunay triangulation can also be seen as a graph and as such, there are several subgraphs to it such as the *De Gabriel graph*, the *relative neighbor graph* and *the Euclidean minimum spanning tree.* A De Gabriel graph is a graph where two points are connected by an edge if their circumcircle contains no other point in it. *The relative neighbor graph* is a graph where two points p and q are connected if and only if for some point s, $dist(p,q) < max\{dist(p,s), dist(q,s)\}$. *A minimum spanning tree* is a connected point set such that any vertex can be reached from any other vertex by exactly one path with smallest possible distance [11].

Lifted Circle. This is another interpretation for the circumcircle of three points and the Delaunay triangulation. Consider the points p_1, p_2 and p_3 which are projected onto the surface of a paraboloid of revolution defined by $f : (x,y) \to (x,y,x^2+y^2)$. Now consider the three-dimensional plane passing through these projected points. Then, we will be able to know if a new point p_4 is inside the circumcircle of p_1, p_2 and p_3 if it is bellow the plane. See Figure 2.5.

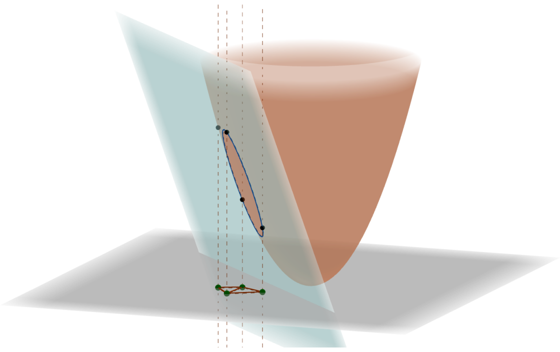


Figure 2.5: Projection of the Delaunay triangulation of 4 points on the paraboloid of revolution

In fact, each triangle of the Delaunay triangulation of a n-dimensional set of points corresponds to a facet of the convex hull of the projection of the points onto a n+1-dimensional paraboloid, and vice versa [15].

2.1.3 3D Delaunay triangulation

For the case of the three-dimensional space, the Delaunay triangulation becomes a Delaunay tetrahedrization, since it consists on tetrahedras, however the term triangulation will be still used through this work. Now let's define a few concepts.

Definition 7 (Tetrahedron) *This is a geometric entity formed by 4 different non coplanar vertices* $V = \{v_1, v_2, v_3, v_4\}$, *joined by 6 different edges (line segments)* $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ *forming 4 different faces* $F = \{f_1, f_2, f_3, f_4\}$.

See Figure 2.6. Notice that a face includes 3 edges and 3 vertices.

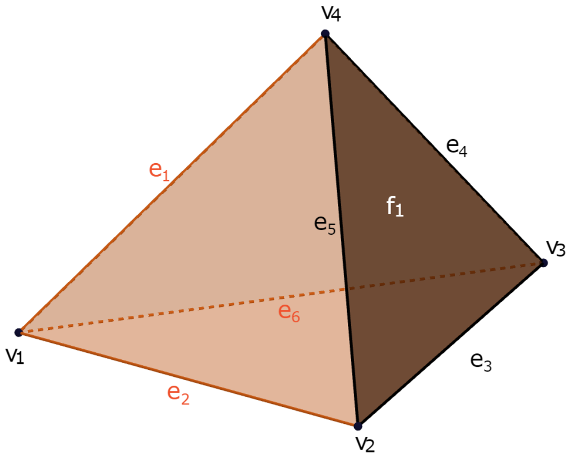


Figure 2.6: Tetrahedra: Vertices, edges and faces (only f_1 is labeled)

Definition 8 (Tetrahedralization) *Let* P *be a set of finite three-dimensional points. The tetrahedralization of* P *consists on a set of tetrahedra* $\mathcal{T}$ *such that:*

- *All points* $p_i \in P$ *are vertices on at least one tetrahedra.*
- $\forall t_i, t_j \in \mathcal{T}: \quad \mathring{t}_i \cap \mathring{t}_j = \emptyset$ *i.e. the interiors of the tetrahedra do not intersect*
- $\forall t_i, t_j \in \mathcal{T}: \quad t_i \cap t_j$ *is either* $\emptyset$ *or an edge* e_i, *a vertex* v_i *or a face i.e. the tetrahedra intersect at an edge, a vertex or a face unless they are disjoint.*

As in the two dimensional case, the idea of the growing cells that subdivide the space into a Voronoi tessellation is still valid for the n-dimensional case; moreover its dual is the n-dimensional Delaunay graph. Another property that holds in the n-dimentional case is the empty $n-$sphere: no $n+2$ points lie in the $n-$sphere passing through the $n+1$ vertices defining the corresponding unit ($n-$simplex) of the n-dimensional Delaunay cell complex [12]. In contrast to this, in the n-dimensional space, the Delaunay cell complex does not maximize the minimum angle [6].

Definition 9 (3D Delaunay Tetrahedralization) *A tetrahedralization $\mathcal{T}$ is a 3D Delaunay tetrahedralization if, and only if, for each tetrahedron $t_i \in \mathcal{T}$, its circumsphere passing through the vertices defining it, contains no other element of P in its interior.*

See Figure 2.7

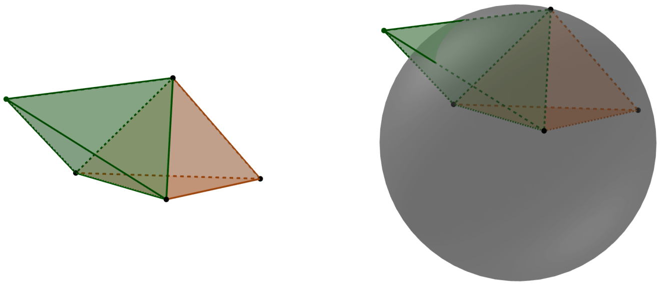


Figure 2.7: Delaunay triangulation of 5 points and circumsphere defining a Delaunay tetrahedron

2.1.4 Algorithms for computing 3D Delaunay Tetrahedralization

There are several algorithms from the 2D case than can be adapted for the 3D Delaunay tetrahedralization. In [6] we can find the following algorithms.

Incremental insertion: also known as the Bowyer–Watson algorithm, it inserts a vertex q into a Delaunay triangulation in four steps.

- Find one tetrahedron whose open circumsphere contains q.
- In the triangulation, find all the other tetrahedra whose open circumspheres contain q.
- Delete these tetrahedra.
- For each triangular face of the cavity, create a new tetrahedron joining it with q.

Gift wrapping: In two dimensions, the wrapping algorithm is similar to the process of winding a string around the set of points, then focusing on an edge, it finds the triangle by looking for a point in front of the edge. The same idea can be extended to $\mathbb{R}^3$; the algorithm constructs tetrahedra instead of triangles and maintains a dictionary of unfinished triangular facets.

Point location by walking: the algorithm traces a straight line through the triangulation, visiting tetrahedra that intersect the line until it arrives at a tetrahedron that contains the new vertex. It has no guarantee of a fast running time but can be improved with the following considerations: the vertices should be inserted in an order that has much spatial locality and each walk should begin at the most recently created solid tetrahedron.

2.2 Polygonal and triangular meshes

The general objective in mesh generation is to decompose a geometric space into elements. The elements are restricted in type and shape [15]. Meshes made out of polygons are important for object representation; there are several types of mesh which will be adequate depending on the application; while triangular meshes offer a smoother surface representation in contrast with other polygonal meshes, quadrilateral meshes offer symmetry, commonly used to create 3D characters or objects [10]; usually for achieving higher resolution, more triangles in the mesh are needed.

Definition 10 (Polygonal Mesh) *A polygonal mesh consists of a set of edges E, a set of vertices V and a set of faces F defined by connected polygons.*

The polygons in a mesh must meet at their edges or vertices and they shouldn't intersect their faces, see Figure 2.8. Generally speaking, it is fairly simple to change polygons with more than three edges to triangles. The difficulty can increase if the polygon is not convex but triangulating a polygon is a problem commonly studied in the Computational Geometry field, so there are several algorithms for computing it. In fact, an algorithm for a $n-$sided polygon, which does not require ordering of the vertices is described in [16], while two $O(nlog(n))$ run time algorithms are described in [17]. The benefit of a triangular mesh, is that it simplifies storage since triangles are planar [10].

Of course, there are several issues when dealing with a three-dimensional mesh; for instance, there is a geometric issue: knowing where the triangles are in the space. A topological issue: how the triangles are connected, (see Figure 2.9). And a representation

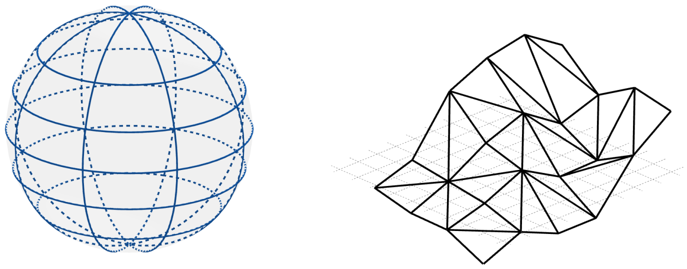
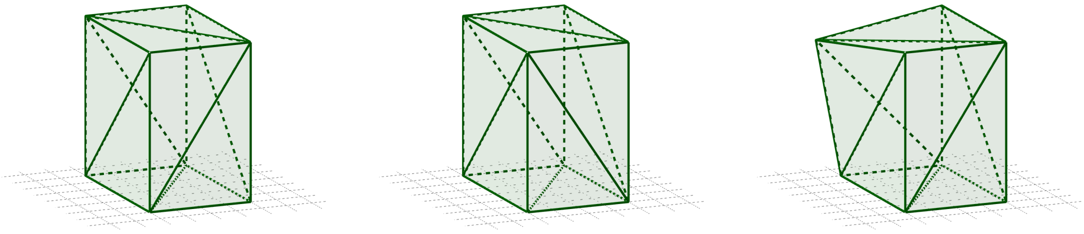


Figure 2.8: Polygonal and triangular mesh

issue: the data structure for storing the triangles in order to use graphics hardware to plot the mesh.



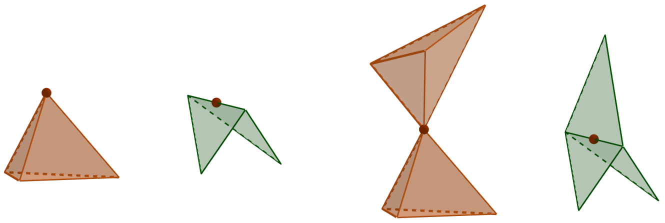
(a) Triangular mesh of a cube (b) Different topology (c) Different geometry

Figure 2.9: Geometry vs Topology on meshes

2.2.1 Mesh topology

Mesh topology constraints based only on the way that triangles link to one another, disregarding vertex locations, can be used to formalize the notion that meshes behave like surfaces. Many algorithms require a mesh with known connection to work [18]. Some requirements are presented below.

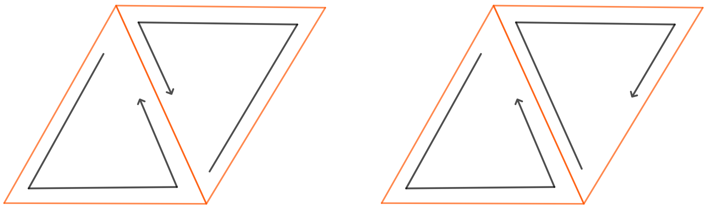
- Be a manifold: any sufficiently small patch has an invertible mapping of that patch onto a unit disk. Formally, a shape is a manifold if any sufficiently small patch is homeomorphic to a disk (i.e., it looks like a surface everywhere on the mesh) [10]. A manifold mesh is defined as one that has no gaps and separates the space on the inside of the surface from the space outside [18]. An easy way to check that a mesh is a manifold is to check that for every edge $e_i \in E$, e_i is shared by exactly two triangles and, for every vertex $v_i \in V$, v_i has a single, complete loop of triangles around it. See Figure 2.10.



(a) Manifold vertex and edge (b) Non-manifold vertex and edge

Figure 2.10: Manifold and not manifold objects

- Orientation: There should be a consistent idea of where is the "front" and the "back" of the triangles. The front of a triangle is the side from which the triangle's three vertices are positioned in counterclockwise order; this is how we determine the orientation 2.11a and 2.11b. In Figure 2.12, we have a mesh of a Möbious band, which is not orientable; of course in practice it is quite rare to find this kind of mesh.



(a) Consistently oriented (b) Inconsistently oriented

Figure 2.11: Orientation of triangles in a mesh

2.2.2 Mesh representation

Let's consider the following mesh made of 3 triangles (see Figure 2.13). Here v_0, v_1, v_2 and v_3 are three-dimensional points defining the mesh. To show the data structures, the C programming language will be used.

These three triangles can be stored as independent entities, like this:

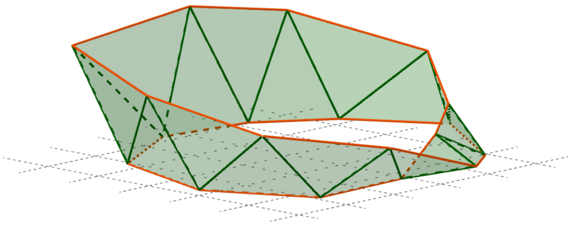


Figure 2.12: Möbious band, not orientable

```

struct Triangle {
    double triangle[3][3];  // list of 3 vertices with 3 coordinates
};
```

As a result, we will store v_2 three times and the rest of the vertices two times, or three times per triangle. Other option is to share the common vertices and store only four vertices, resulting in a shared-vertex mesh; this data structure uses pointers so that the triangles store the location of the vertices containing the vertex data [18]:

```
struct Vertex {
    double X, Y, Z;
};

struct Triangle{
    struct Vertex triangles[3];  //list of vertices
};
```

The entries in the array “triangles” are references to the Vertex structure, pointers to be precise. In practice, we store vertices and triangles in arrays, with the references to triangles and vertices handled by storing indices:

```

struct Triangle * Mesh [50]; // list of pointers of 50 triangles called mesh
/*
Mesh[0] // this is a pointer to the first triangle
```

```
(Mesh[0].triangles)[1] //this is a pointer the second Vertex of the triangle 0
(Mesh[0].triangles)[1].X // this is the x coordinate to the second vertex of triangle 0
*/
```

Of course, there are simpler and more efficient ways of implementing the mesh part, (this is just a personal favourite and serves to observe the idea of storing references). This method of storing shared vertex triangles in a mesh is called an *indexed triangle mesh* [18], it looks as follows.

Table 2.1: Indexed triangle mesh vertices and triangles

(a) Vertices

v0	x_0	y_0	z_0
v1	x_1	y_1	z_1
v2	x_2	y_2	z_2
v3	x_3	y_3	z_3

(b) Triangles

t0	v1	v3	v2
t1	v3	v0	v2
t2	v0	v1	v2

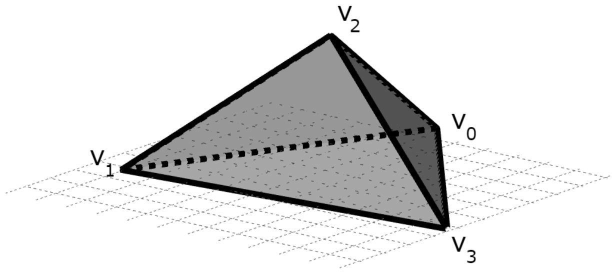


Figure 2.13: Triangular mesh made of 3 triangles in 3D space

2.3 Surface reconstruction and Delaunay

Meshes serve as a discrete representation of continuous space. According to Shewchuk in [19], meshes based on triangles or tetrahedra are very useful for applications such as interpolation, rendering, and numerical methods. Sadly, there is no metric that consistently separates good and bad space representations, it is a general rule to avoid tiny or extremely large angles since they lead to numerical errors [15]. In practice, we try maximizing the minimum angle of the triangles or tetrahedra, because in this way, we obtain well-shaped figures with bounds on these minimum and maximum angles.

Many mesh generation algorithms are based on *Delaunay triangulations*, as they are effective in practice and theory. For example, in [19], the authors present an implementation that generates excellent meshes, usually surpassing theoretical bounds, effectively eliminating tetrahedra with small or large dihedral angles. This implementation is based on an algorithm that can generate a conforming mesh of Delaunay tetrahedra whose circumradius-to-shortest edge ratios are not greater than two. Using a complex of vertices, constraining segments, and planar straight-line constraining facets in E^3, as input. More specifically, this and other related algorithms come from the family of *Delaunay Refinement* technique. These algorithms work by keeping “a Delaunay or constrained Delaunay triangulation, which is refined by inserting carefully placed vertices until the mesh meets constraints on triangle quality and size.” [19].

The use of Delaunay techniques for mesh generation of surfaces is not new. For example, in 1980, in fields such as engineering, we already see the use of triangulation to guide vertex creation [20]. Nevertheless, the first good Delaunay refinement algorithm is due to Paul Chew [21], which takes as its input the region to be meshed in the form of a set of vertices and segments that define it. The algorithm continues adding vertices generating a two-dimensional constrained Delaunay triangulation where the triangles angles are bounded between 30 and 120 degrees. Later, this algorithm was generalized to three dimensions for unconstrained point set inputs [22]. The output of all these algorithms are uniform meshes, whose units are approximately of the same size.

The problem with uniform meshes is that they usually produce many more triangles or tetrahedra than necessary [19], making the computational cost of this algorithm bigger than needed when using the meshes for applications. However, other authors have proposed algorithms producing meshes of well-shaped triangles whose sizes are graded [19].

All these algorithms’ effectiveness relies on the favorable features of Delaunay triangulation presented in Section 2.1.2. The greatest advantage is that when doing meshes, the central question for a refinement algorithm is, “where should the next vertex be inserted?”

The reasonable answer tends to be "as far from other vertices as possible." Since Delaunay triangles are constructed so that no vertices are in the interior of its circumcircle, it results in a good search structure for finding points far from other vertices when adding these points for the mesh generation [19].

Delaunay refinement algorithms are not the only algorithms for surface reconstruction based on Delaunay Triangulation. In fields such as astronomy, a Delaunay Density Estimator Method is used to render a fully volume-covering reconstruction of a density field from a set of discrete data points [23]. This method is based on the stochastic geometric aspect behind Delaunay tessellation of a point set. The tessellation is a volume cover tiling of space into tetrahedra [24], and is related with the Voronoi tessellation, since they are dual because the vertex of a Voronoi cell is the nucleus of a Delaunay vertex, and viceversa.

2.4 LiDAR technology

LiDAR (Light Detection and Ranging) [25], is a remote sensing method that uses pulsed laser to measure the height of objects on the ground or the terrain's topography. The light pulses generate precise three-dimensional information about the shape of the Earth and its surface characteristics by sensing the light that reflects off of the surface of objects and the earth.

To calculate the distance traveled by the emitted light, the LiDAR system measures the time it takes to reach back from the ground, then converted to elevation. These measurements use a GPS, identifying the light's (x, y, z) location, and an Internal Measurement Unit which provides the orientation [26].

Since the light energy used in the LiDAR system is a collection of photons, some of the protons might travel farther and reach the floor, while some others can reflect on a higher part. This results in multiple reflections recorded from one pulse of light and, thus, two LiDar Systems, one that records peaks in the waveform curve of a pulse and another that records the full waveform of the pulse. The first is called a *Discrete Return LiDAR system* and the second *Full Waveform LiDAR system* [26]. In figure 2.14, the light emitted by the LiDAR system is represented in blue while the reflecting one is in green. For a Full Waveform, all the data from the green arrows is processed, while on the Discrete Return, only the information from a few selected green arrows.

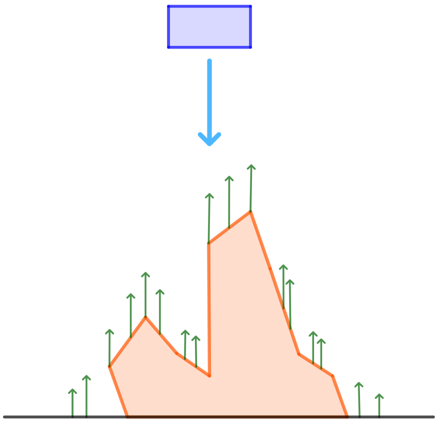


Figure 2.14: LiDAR light energy used to measure objects in the ground

Whether it is a discrete or full waveform system, LiDAR point clouds are often available as discrete points in .las file format. .las is a format supported by the American Society of Photogrammetry and Remote Sensing (ASPRS). It supports the exchange of any 3-dimensional (x, y, z)-tuple [27]. There is also the .laz format; in comparison with .las format, .laz is a highly compressed version of .las.[26]

Most of the time, the .las files contain huge sets of points, and to view the terrain, convert the file to another format and do many more operations with the (x, y, z) points, we can use LAStools. LAStools is an open software (there is a closed version, too) developed by Martin Isenburg for raidlasso GmbH. It is a cluster of programs that can be used individually or together to process enormous LiDAR data sets [28].

2.4.1 LiDAR applications

LiDAR technology is used in many areas, some applications follow [29]:

- **Wide-area, Corridor and City Mapping, and Urban Planning:** LiDAR technology is used to produce incredibly accurate digital city models and locate features such as new construction, changes in vegetation, light poles, streets, etc. Which can be used for city planning.
- **Vegetation Mapping:** The capacity of LiDAR to accurately measure the vertical

structure of individual trees and forest canopies makes it the ideal tool for natural resource management, forest inventory, ecological analysis, fire control planning, and conservation planning.

- **Power Lines:** LiDAR can be used for mapping transmission line utilities, the ground under the line, the location of the towers and poles, the sag on the wires, and the up growth of any vegetation in the facilities that can damage the lines.
- **Coastal Mapping:** LiDAR is ideal to monitor coastal habitats, to map the uneven topography and effectively monitor the land and marine borders.
- **Archaeological and Historical Surveys:** There are structures from ancient civilization that have been discovered by a LiDAR scan, most of the time this structures are hidden by some vegetation in the area.

2.4.2 LiDAR and smartphones

LiDAR is a rapidly growing technology that continues to advance in terms of power, accuracy, and speed. Such advancements should result in new application spaces for LiDAR technology, in particular devices like smartphones. In [30], it is demonstrated how well the iPhone performs at capturing raster picture data and orientation, on par with analog compass-clinometers and reflex/mirror-less cameras. In [31], a system that uses the LiDAR sensors found in contemporary cellphones, is developed to ascertain fluid characteristics. With just one drop, the system can distinguish between coagulated and un-coagulated blood as well as milk with various fat contents. In [32], a mobile app uses iOS devices with LiDAR capabilities and Clew3D, to locate orientation and mobility markers along a recorded path and determine their relative locations to generate automated directions for a trajectory.

2.5 KD-tree and spatial indexing

Unless stated otherwise, background material in the following section is collected from [33]

2.5.1 Binary Trees

In computer science, trees are data structures consisting of nodes and leaves where we store data. A tree can be defined recursively as follows: a tree is either empty or it has a root whose children recursively are trees. The most famous trees are *Binary Trees* which are characterized by having only two children for each node, namely the left child and the right child. Binary trees can be ordered and oriented or not.

Binary trees are often used to store data that needs to be accessed in form of queries or searches. A binary search tree is a binary tree that imposes the following constraint on its nodes: the search key for a node A must be bigger than the values for all nodes in the left sub-tree of A, and less than the values in the right sub-tree of A. These type of trees are useful for storing data in form of registers and two-dimensional data, however they are poorly structures when working with spatial data with more dimensions.

2.5.2 KD-trees

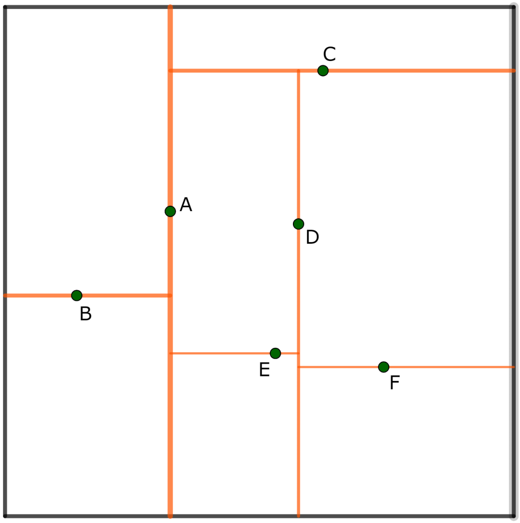
KD-trees were invented in 1970s by Jon Bentley as a modification to binary search trees for efficient processing of multi-dimensional search keys [34]. Because KD-trees are used specially for multidimensional searches, it is a good idea to think of them in geometrical terms.

Let us suppose we have a set of spatial points describing a space. Any non-leaf node in a KD-tree can be thought of as inherently generating a splitting *hyperplane* that splits the space into two halves; this is shown in Figure 2.15a, the hyerplane passing through A splits the space into two halves. The left sub-tree of that node represents points to the left of this hyperplane, while the right sub-tree represents points to the right of this hyperplane. Every node in the tree is connected to one of the k dimensions, with the hyperplane perpendicular to that dimension's axis (See Figure 2.15).

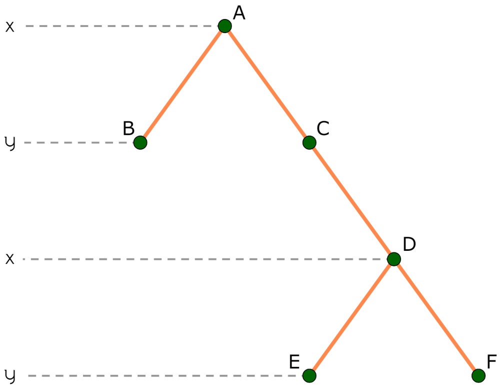
The properties of KD-trees include:

- Each level has a cutting dimension.
- We cycle through the dimensions as we walk down the tree.
- Each node contains the information of a point $P = (x, y)$.

To search for a point (x', y') we only need to compare coordinates from the cutting dimensions, i.e., if cutting dimension is x, then we ask: is $x' < x$?



(a) KD-tree hyperplanes



(b) KD-tree

Figure 2.15: KD-tree example

2.5.3 KD-Trees Nearest Neighbor

One the most famous uses of KD-trees is its application to solve the problem of finding the nearest neighbor of a point. Nearest Neighbor Queries are very common, the idea is that given a point P, we want to find the point Q in the data set such that it is the closest to

P.

A first approach to solve this problem is finding the cell of the hyperplane defined by P and return the point it contains. However, this does not work because the nearest point to P in space may be far from P in the tree. Thus, the correct idea is traversing the whole tree, but making two modification to prune the search space:

1. Keep a variable C to store the closest point found so far. Prune sub-trees once their bounding boxes say that they cannot contain any point closer to P than C.
2. Look for subtrees in the order that maximizes the possibility of pruning.
3. If the distance between P and the current C is greater than the distance between P and a new candidate, then no point in the bounding box of sub-tree can be closer to P than C. Thus, there is no reason to search said sub-tree.
4. However, if there is a point closer to P than C, then update the new point.
5. Recursively search for the sub-tree closer to P: First search the sub-tree that would contain P as if we were inserting P below.

The cost of this algorithm at its worst case is $O(n)$, however in practice, the run time is closer to: $O(2^d + \log n)$. To summarize, the most important steps that occur in this algorithm are: *Storing partial results:* keep best results so far, and update. *Pruning:* reduce search space by eliminating irrelevant trees, thus saving storage and computational resources. *Traversal order:* visit the most promising sub-tree first.

2.5.4 Spatial Indexing

The concept of using KD-trees for neighborhood computation is related to the concept of *Spatial Indexing.* As [33] explains: “spatial indexing is much like the index in the back of a book, which can be used to easily find the page, where a certain notion occurs.” In this way, a spatial database indexes over points in space provides easy access to them without the requirement of going through the whole database. It is clear, how KD-trees are useful for spatial indexing, as each node represents a point, and you don’t need to search for the whole space when looking for some specific point. However, KD-trees are not the only data structures useful for spatial indexing, we also have *Quadtrees* and *Octrees.*

2.5.5 Quadtree and Octree

Similar to KD-tree, quadtrees are spatial data structures that split a d-dimensional space recursively into 2^d equally sized cells. In our three-dimensional case, quadtrees are called octrees. Given a point set P, the construction of a three-dimensional octree starts with a bounding cube around P, which will be the initial cell of the octree. Recursively, each cell is subdivided into eight equally sized cube cells until every cell of the octree only contains one point $p_i \in P$. We used the standard enumeration for these cells, as shown in Figure 2.16, here 0 refers to the hole cube.

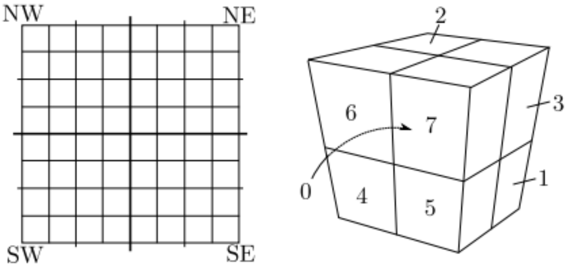


Figure 2.16: The labeling of a Quadtree and an Octree [1, 2].

This procedure gives a tree where the initial bounding cube acts as the root. Each internal node has eight children, corresponding to the eight equally sized cube cells that the cell corresponding to the considered node is split into. The leaves of the tree either represents a point from the input set P or are empty. An excellent introduction to Octrees and their implementation can be found in [1, 2]. To end this section, it is good to notice that octrees are better than KD-trees for procedures such as insertion and deletion of points since in KD-trees, we need to rebuild the whole tree, and in an octree, we can do it optimally.

2.6 Covariance matrices and representation of surfaces

One way to do it when storing points representing a surface is by arranging them into a matrix. We can characterize a surface with the covariance matrix of local descriptors rather than the descriptors themselves [35]. Covariance matrices have been used for texture classification, object detection and tracking, face recognition, and action recognition. The advantage of using covariance matrices is that they provide a natural way to store fusing multi-modal features about a surface without normalizing.

In [35], the authors represent a three-dimensional surface as a set of overlapping patches $\{P_i, i = 1...m\}$, each patch P_i come from around a representative point $p_i = (x_i, y_i, z_i)^t$. For each point $p_j = (x_j, y_j, z_j)$, $j = 1, ..., n_i$, in the patch P_i, they compute a feature vector f_j, of dimension d, that contains the local geometric and spatial properties of the patch. If we let $\{f_j\}_{j=1...n}$ be the d-dimensional feature vectors computed over all points inside P_i, then a three-dimensional patch P_i can be represented as a $d \times d$ covariance matrix X_i:

$$X_i = \frac{1}{n}\sum_{j=1}^{n}(f_j - \mu)(f_j - \mu)^t, \tag{2.1}$$

where μ is the mean of the feature vectors computed in the patch P_i. By definition, this covariance matrix is a symmetric matrix, whose diagonal elements represent the variance of each feature and its off-diagonal elements represent their respective correlations.

Another way to see the use of covariance matrices for representing a surface is through Algebraic Geometry and the use of vector fields. As [3] explains, vector fields are good models to represent surfaces since they can encode the surface normal (a vector perpendicular to the tangent plane of the surface at a point P) together with the nearest neighbors in each point, thus enabling nearest neighbor search in linear time complexity. As shown in Figure 2.17, the vector field of a three-dimensional surface is composed of a regular grid made of cubic nodes with side length L.

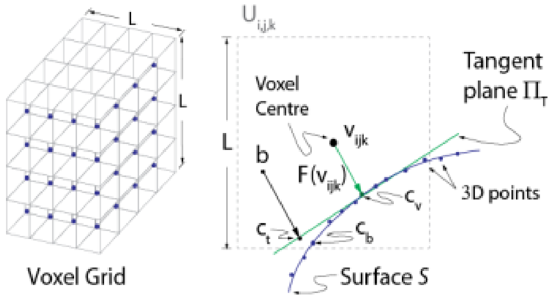


Figure 2.17: The vector Field implicit representation of a surface, taken from [3].

Each cube in the grid is indexed by the coordinates of its center $v_{i,j,k}$ in a reference frame W_r. The covariance matrix $C_{i,j,k}$ of the points falling in cube ijk is defined as

$$C_{i,j,k} = \frac{1}{n}\sum_{i=1}^{n}(p_i - \bar{p})(p_i - \bar{p})^t, \tag{2.2}$$

where $\bar{p}$ is the mean vector defined as

$$\bar{p} = 1/n\sum_{i=1}^{n}(p_i) \tag{2.3}$$

Notice the similarities between equation 2.1 and equation 2.2. If the cube size L is small enough, we can assume that the object surface in the cube can be approximated by the plane Π_T tangent to the surface. The normal vector to this tangent plane is the eigenvector corresponding to the smallest eigenvalue λ_{min} of the covariance matrix $C_{i,j,k}$. Thus, the idea behind the use of covariance matrices, is that we can extract the eigenvectors of this matrix, obtaining the tangent plane which is a two dimensional approximation of the surface, and over this plane, we can compute the Delaunay Triangulation of the patches. Also, the eigenvalues of the covariance matrix are indicators of the roughness of the surface in a given point, taking into consideration the other points from the patch.

2.7 Tangent plane and tangent cone

Consider a two-dimensional curve $y = f(x)$, and a straight line that "just touches" said curve at a particular point c. This line is referred to as the *tangent line* to a plane curve. Space curves and curves in n-dimensional Euclidean space have a similar definition, the plane that "just touches" a surface at a certain location is the tangent plane to that surface. This concept on the tangent is one of the most fundamental ideas in differential geometry; in higher dimensions, we might speak about the tangent space.

In [36] they define the tangent space to an affine variety X at a point x as the set of all lines through x tangent to X. In the three-dimensional space we will have a surface. Let $z = f(x, y)$ be continuous, the tangent plane at the point $P = (x_0, y_0, z_0)$ can be defined with the following equation:

$$A(x - x_0) + B(y - y_0) + C(z - z_0) = 0 \tag{2.4}$$

with the normal vector to the plane being $n = (A, B, C)$.

2.7.1 Tangent cone

From [4] we will describe the following subsection.

For a regular parametric surface, there are two tangent cones: T_u for a variable parameter u and a constant parameter v, and $\mathbb{T}_v$ for a variable parameter v and a constant parameter u. $\mathbb{T}_u$ of a parametric surface is a set of points in $\mathbb{E}^3$ such that a position vector of a point in $\mathbb{T}_u$ corresponds to a tangent vector of the surface with respect to u, at some u and v. $\mathbb{T}_v$ is defined analogously. A *normal cone* $\mathcal{N}$, is a set of points in $\mathbb{E}^3$ such that a position vector of a point in $\mathcal{N}$ corresponds to a normal vector on the surface, at some u and v. A *visibility cone* $\mathcal{V}$, is a set of points in $\mathbb{E}^3$ such that any line parallel to the position vector of a point in $\mathcal{V}$ intersects with the surface at most once for all u and v. Figure 2.18 illustrates cones for regular parametric surfaces.

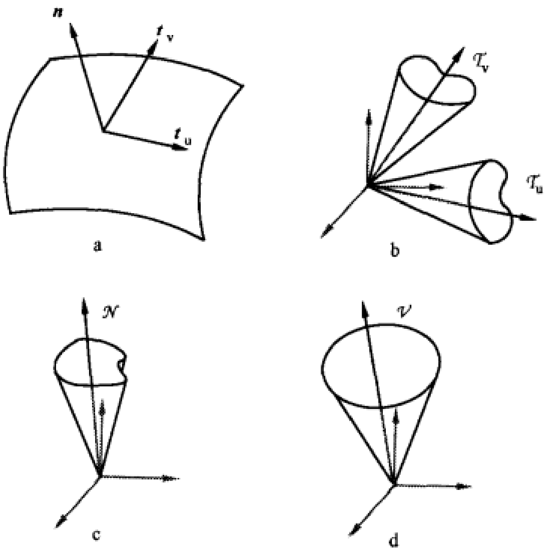


Figure 2.18: Cones of a parametric surface. (a) A regular parametric surface. (b) Two tangent cones. (c) Normal cone. (d) Visibility Cone [4]

For our purposes, we assume the surface is regular so that a normal vector exists at each point on the surface.

Chapter 3

Methodology

The Delaunay triangulation is a good choice for terrain representation since maximizing the minimum angle will enable a good visualization of the data. However, it becomes a bit difficult to implement since the 3-dimensional case is not that trivial. Moreover, it is important to take into account the size of the data set to be used, since it will take a huge amount of run time, even with a fully efficient implementation.

This chapter will describe all steps taken from the description of the problem to the analysis of the obtained results, as well as the pseudo-code for the algorithm and some recommendations for its implementation.

3.1 Phases of Problem Solving

3.1.1 Description of the Problem

We want to obtain a surface reconstruction by computing a Delaunay triangulation on a three-dimensional set of points. The algorithm must not use a projection of the points on a plane, since it does not give an accurate representation of the terrain when singularities are present: a sharp edge, a failure, steep slopes, holes or the top of a sharp mountain, see Figure 3.1. Around these singularities, we can have errors in the triangulation.

Therefore, the z coordinate must be taken into account and so, the relation between the points of the data set, meaning that the algorithm must be able to identify quantitatively when the surface to be reconstructed corresponds to a roughly flat smooth surface or a nearly singular one.

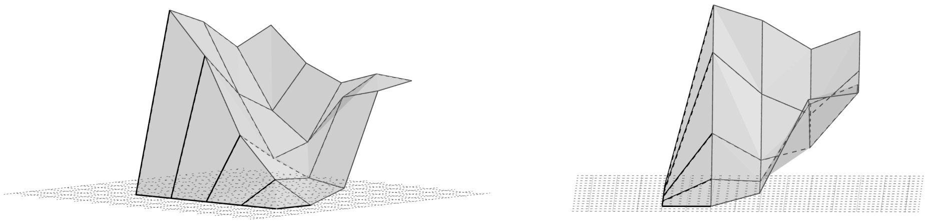


Figure 3.1: Examples of irregular terrains with singularities.

3.1.2 Analysis of the Problem

We want to develop an algorithm that does not do a projection on the plane; instead we want the algorithm to analyse the topography of the terrain (to a certain level), we must find an indicator that allows the building of the Delaunay triangulation. For the purpose of achieving this, we will use the information that can be obtained with the variance and covariance matrix of each point and its closest nearest neighbors, with the central point taken as the origin of the vector space associated to the affine Euclidean space.

By computing the eigenvalues and eigenvectors of the variance and covariance matrix of each point, we will obtain a geometrical interpretation of each point: if a point is a singular point, corresponding to a terrain failure, a hole or steep slopes for example; or if is flat, smooth and continuous terrain, in a local sense. This information will be obtained from the smoothness index α which corresponds to

$$\alpha = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}, \tag{3.1}$$

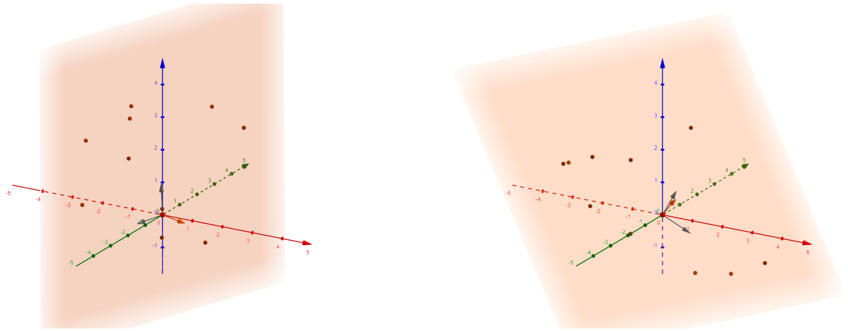
where $\lambda_1 \geq \lambda_2 \geq \lambda_3$ are the eigenvalues of the variance and covariance matrix. From Equation 3.1, we can obtain these possible outcomes:

- $\alpha = 0$: this only happens when there are no neighbors for a point p_i, meaning it is an outlier on the set or the points are less dense in some area.
- $\alpha = \frac{1}{3}$: this means the point is singular or the terrain is irregular, hence for the triangulation the tangent cone should be used for the triangulation.
- $\alpha = 1$: meaning the terrain is flat and continuous, however it is less likely to obtain

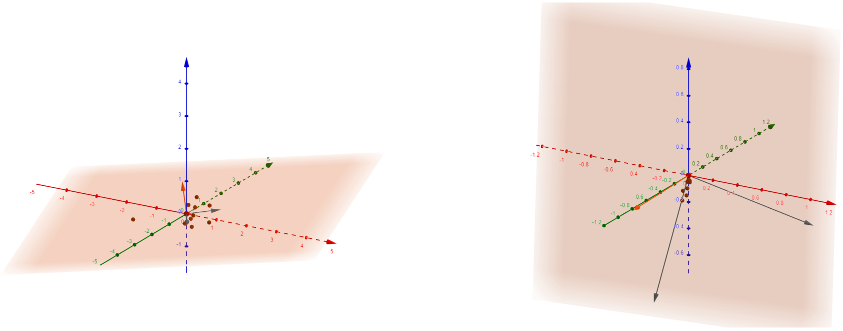
because of the noise in the data set.

- $\alpha \in \left(\frac{1}{3}, 1\right)$: this is the most likely scenario, if α is closest to 1 it indicates a smooth surface.

Figure 3.2 shows the location of points on the data set corresponding to four different values of α. When the points are more scattered the values of α are lower as in Figures 3.2a and 3.2b. It is important to notice that when α is closer to 1 or even 1, it does not necessarily mean an horizontal portion of the terrain, it can also mean vertical continuity as it is shown in Figure 3.2d.



(a) $\alpha = 0.34$ (b) $\alpha = 0.5$



(c) $\alpha = 0.7$ (d) $\alpha = 1$

Figure 3.2: Graphic representation of α

With the points projected on the tangent plane we can compute the Delaunay triangulation in two dimensions; keeping the reference to the original points will allow to plot the Delaunay triangulation in its original three dimensional space obtaining the Surface reconstruction.

3.1.3 Algorithm Design

The algorithm is explained bellow as step by step guide and the following notation is going to be used for the purpose of a better explanation:

Let P be a set of thee dimensional points such that for every $p_i \in P$, there is a set N_{p_i} containing the nearest neighbors of p_i, also elements of P. An element $p_j \in P$ will be an element of N_{p_i} if $p_j \neq p_i$.

There will be cases in which the points are more densely distributed in some areas, it might happen that a ball centered in some point p_i will enclose hundred of neighbors, while the same ball centered in another point p_j will have none.

The algorithm:

1. **Find 10 nearest neighbors of each point:** for each point $p_i \in P$ we can choose ϵ small enough and work with the elements p_j inside a ball $d(p_i, p_j) < \epsilon$. Either $|N_{p_i}| = 10$, $1 < |N_{p_i}| < 10$ or $|N_{p_i}| > 10$. In the first case we are done; in the second case we can increase the radius of the ball and in the last case we can sort the elements of N_{p_i} and choose the 10 closest ones to p_i.

2. **Compute the Variance-Covariance matrix of each point and its 10 nearest neighbors:** using equation 2.1 we will compute the Variance-Covariance matrix Σ_i. We need to make an axis chance before computing Σ_i for each point, p_i will move to the coordinates $(0, 0, 0)$ and the nearest neighbors in N_{p_j} will move accordingly. This is a local axis change.

3. **Compute the eigenvalues and eigenvectors:** we need the eigenvalues $\lambda_1 > \lambda_2 > \lambda_3$, and their corresponding eigenvectors, v_1, v_2 and v_3, of the Variance-Covariance matrix Σ_i.

4. **Compute the smoothness index:** using the eigenvalues λ_1, λ_2 and λ_3, obtained before and equation 3.1 we will obtain a value α for each Σ_i

5. **Use the tangent plane or tangent cone to project the neighbors:** with the smoothness index α we will know in which cases to use the tangent plane and which ones the tangent cone. In the case of the tangent plane, the eigenvectors v_1 and

v_2 will generate the tangent plane, while v_3 will represent the normal vector to the plane. Using the normal vector, we can project the points $n_{i_j} \in N_{p_i}$ on the tangent plane.

6. **For the tangent plane:** from the projection of p_i and the corresponding $n_{i_j} \in N_{p_i}$ on the tangent plane, we compute the Delaunay triangulation. With that information, we return the triangles to the original points

7. **Flips and fixes:** What remains is to check for non Delaunay triangles, holes and mistakes in the triangulation. For the non-Delaunay triangles, we can make an edge flip if is possible and the surface will be triangulated.

3.1.4 Implementation

For the implementation, a LiDAR data set was obtained from the National Geographic Institute of the Government of Spain, the link: https://centrodedescargas.cnig.es/Centro Descargas/buscador.do; under the file name "PNOA-2014-BAL-356-4304- ORT -CLA-CIR.laz", a .laz file from 2014, with a density of 0.5 pts/m^2 and 8.65 MB. The set corresponds to a small region of Ibiza, specifically the one depicted in Figure 3.3. The same region can be seen from the Google Maps app, it corresponds to Figure 3.4, where the topography of the terrain can be appreciated and finally, on Figure 3.5, we can see the 2'833.554 points in the set by using LAStools program.

From Figure 3.5, we can notice some of the points define the surface of the sea and the others the surface of land with some mountains and relatively smooth surface. On the other hand, there is also an area in the sea where there are less points. In Table 3.1, we see the density of the points by sector.

Table 3.1: Density of points in the poin cloud set

0	400	800	1200	1600	2000
2000	103737	123556	125780	170829	272336
1500	84415	49372	103601	294939	305282
1000	13104	11957	245686	161974	113317
500	35678	77214	125585	155330	259862

For the implementation of the algorithm, the C programming language was mainly used

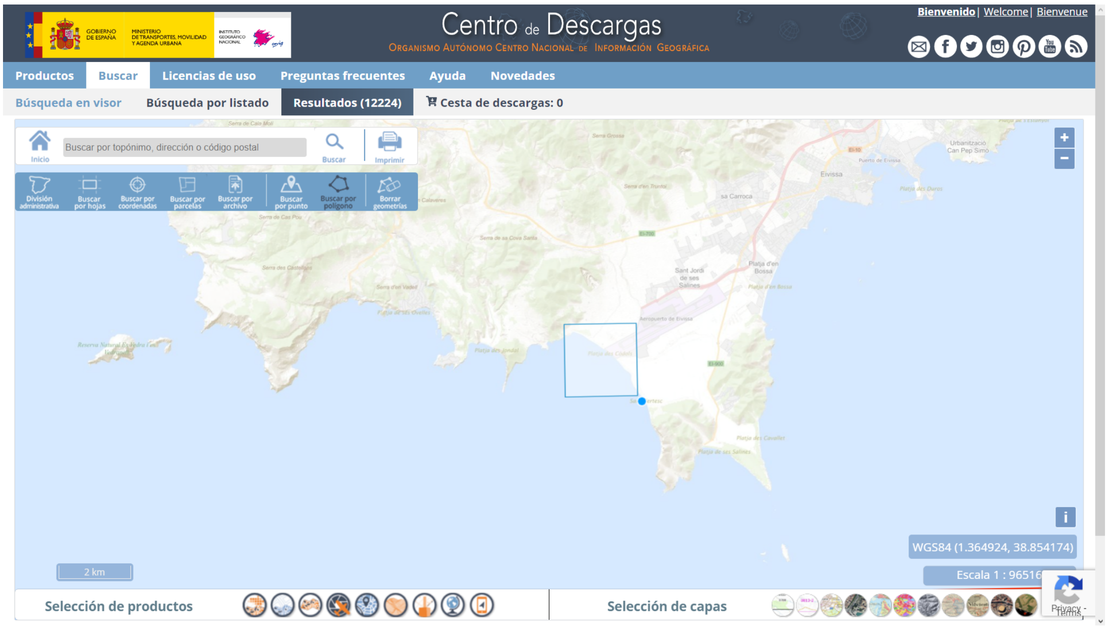


Figure 3.3: Region of Ibiza from where the LiDAR data set was taken. Image comes from the National Geographic Institute of the government of Spain's web page.

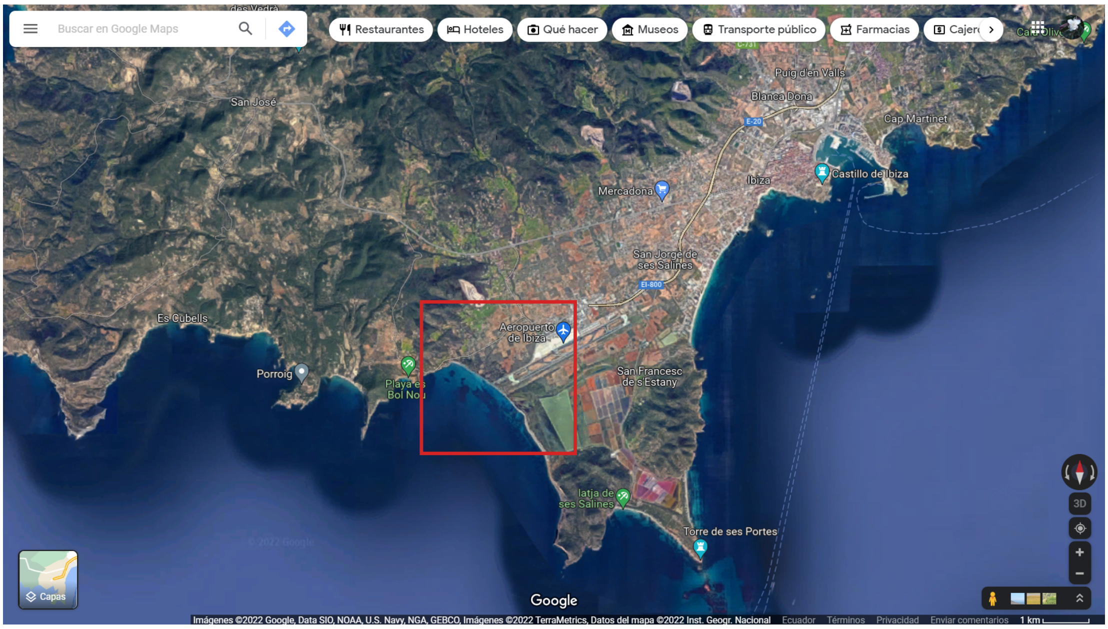


Figure 3.4: Data set view using Google Maps: approximate location of the point set.

for processing the point clouds and the nearest neighbor queries. RStudio was used for computing the eigenvalues and eigenvectors of the variance and covariance matrix as well for the triangulation with the "tripack" library, a constrained two-dimensional triangulation package. LAStools was used to observe the point clouds and change the file format from .las to .xyz and .txt for the analysis; and gnuplot was used for the 3D visualization. The

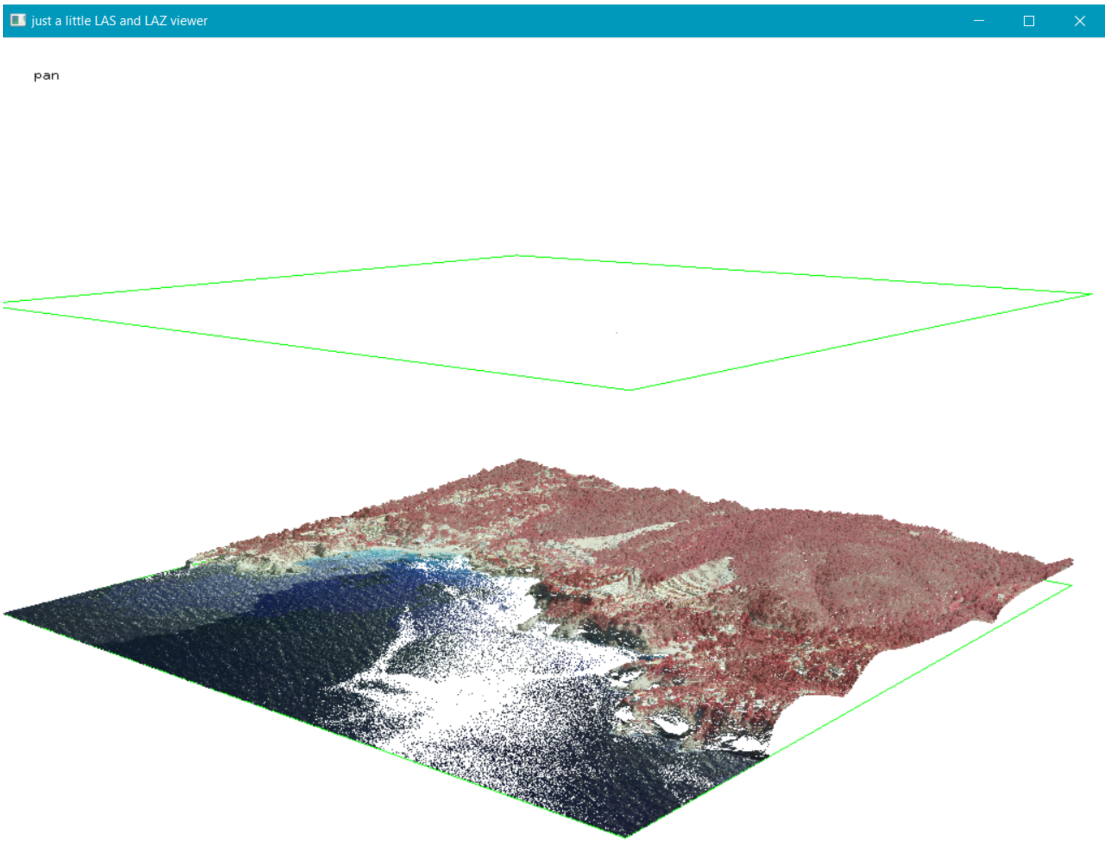


Figure 3.5: LiDAR data set viewed using LAStools

KD-tree implementation was taken from a free source: https://github.com/jtsiomb/kdtree. It is a non adaptative KD-tree with nearest neighbor queries.

3.1.5 Testing

For a better analysis on the algorithm implementation, from the main area, 3 sections were chosen for triangulation (see Figures 3.6, 3.7 and 3.8). Section 1 is a smooth flat area, corresponding to a region of the sea located in the lower left part of the original data, with 13104 points without prepossessing. Section 2 corresponds to a boundary between sea and land with small elevations but still a flat surface, it has 77214 points without prepossessing. Finally, Section 3 corresponds to an irregular part of the land, this section has a big amount of vegetation and 294939 points.

Since the LiDAR technology uses a beam of light, there is a chance to obtain outliers in the data corresponding to places where the light reflects or is absorbed by the ground or an object. A clear example is 3.6, where there is a white spot, probably due to the

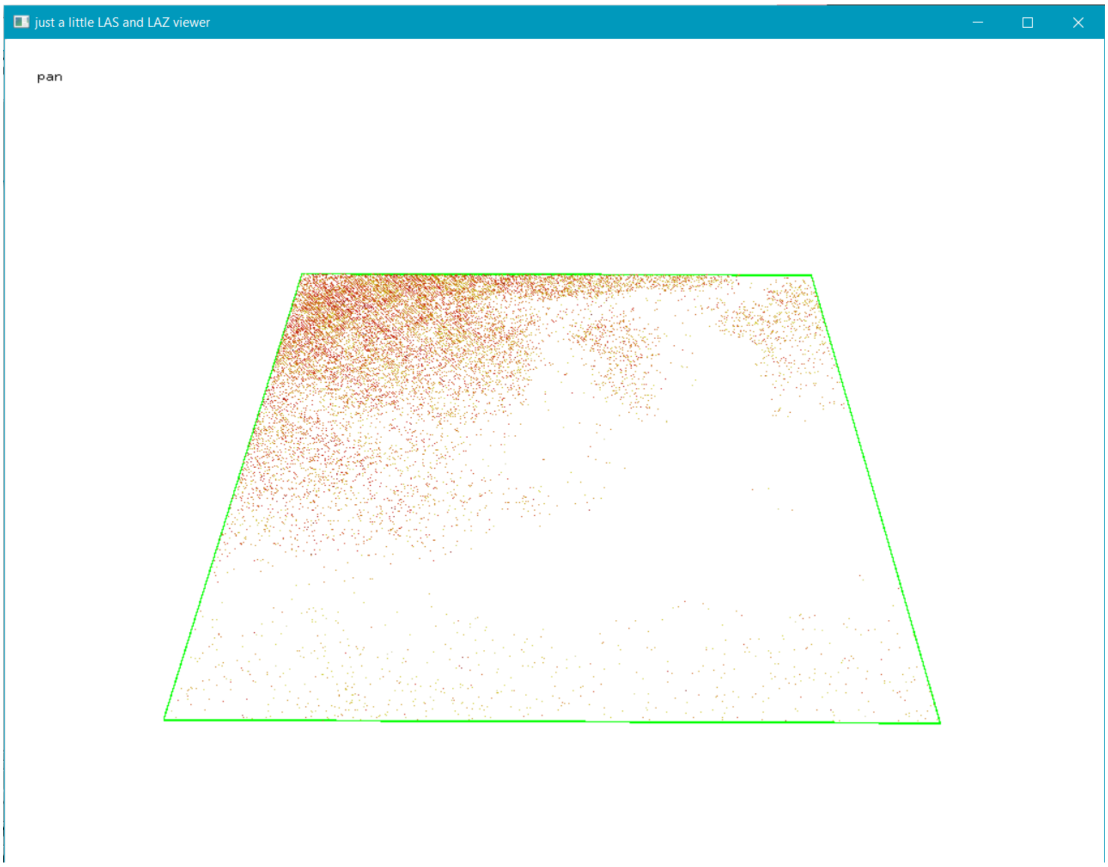


Figure 3.6: Section 1: $(x, y, z) \in [0, 400) \times [500, 100) \times \mathbb{R}$

sea waves. Hence the first step taken was to use the KD-tree implementation to find the nearest neighbor of each point; with this we can obtain the first data filtering. This was computed on the entire set, resulting on table 4.1. The next step is to compute the ten nearest neighbors of each set.

After that, when computing the variance and covariance matrix, and analyzing the values of α, the points with lower values of α will be omitted since they will need to be triangulated with the tangent cone. The other points will be triangulated by using the tangent plane and then plotted.

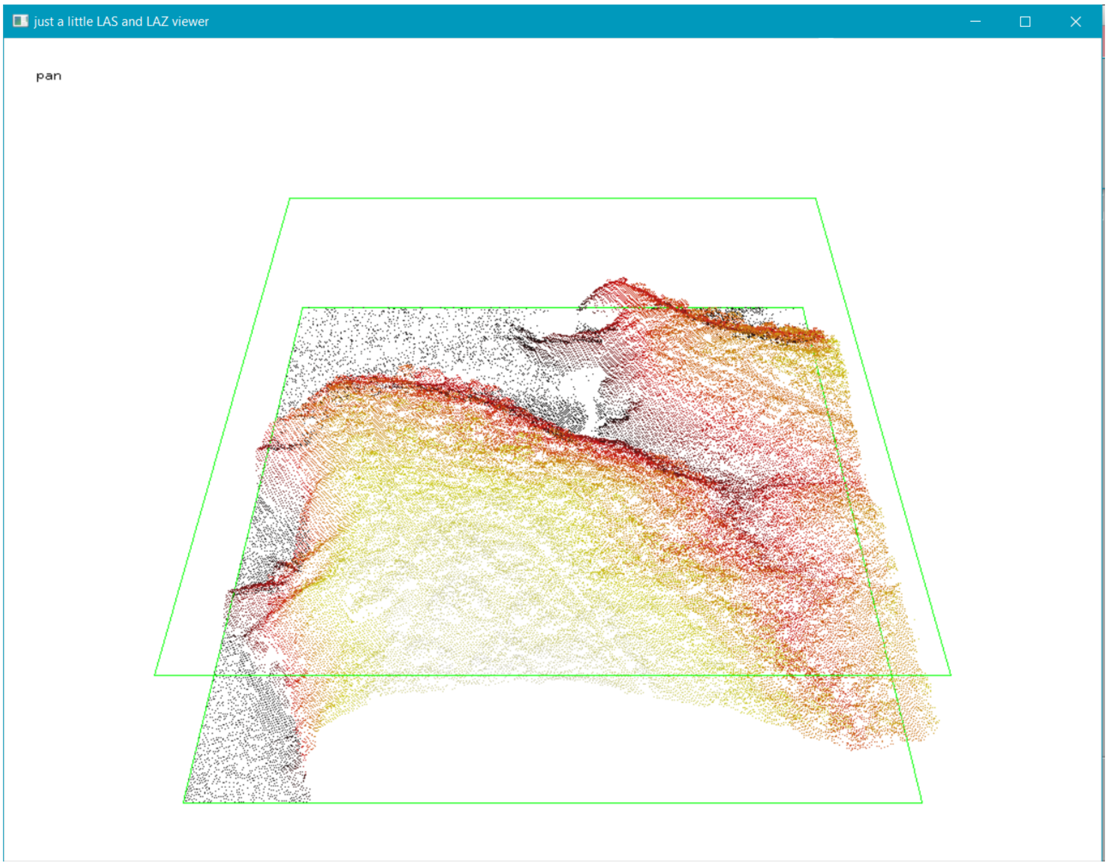


Figure 3.7: Section 2: $(x, y, z) \in [400, 800) \times [0, 500) \times \mathbb{R}$

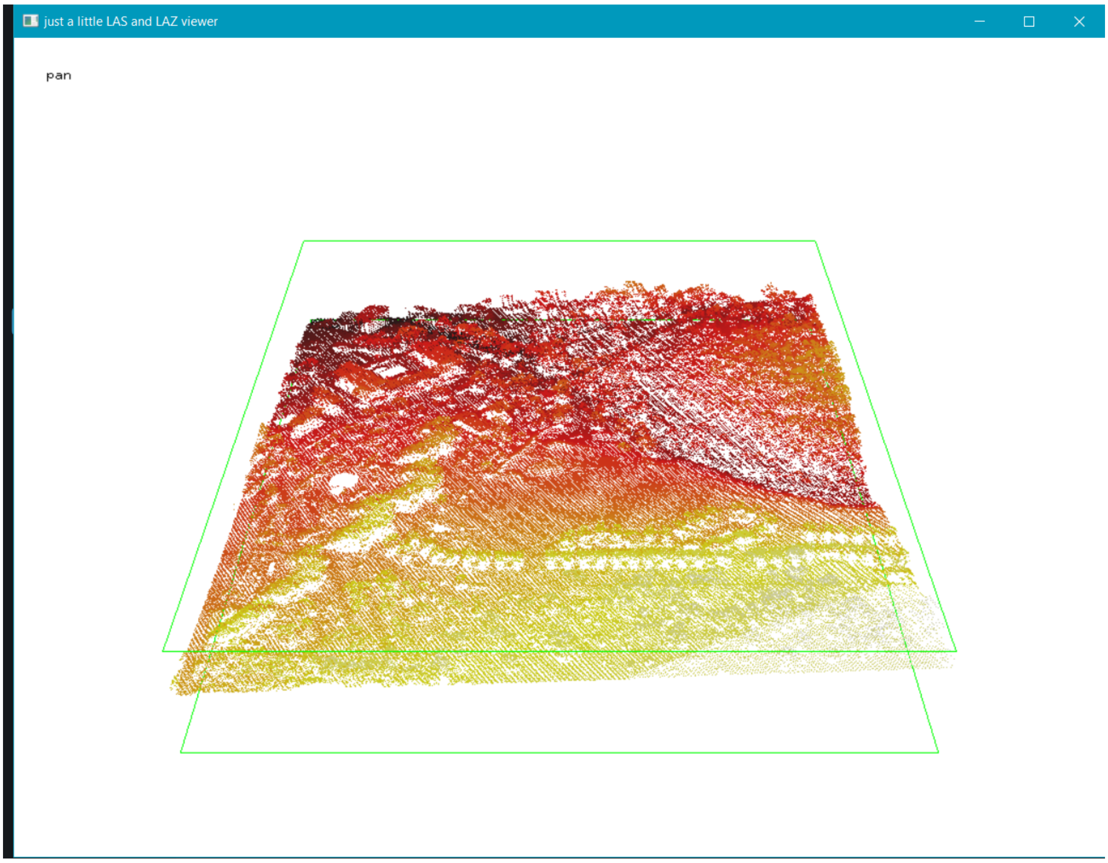


Figure 3.8: Section 3: $(x, y, z) \in [1200, 1600) \times [1000, 1500) \times \mathbb{R}$

Chapter 4

Results and Discussion

From the development of the algorithm to the implementation, there are several observations to notice, they will be mentioned in chronological order:

Nearest neighbors: First, the nearest neighbor was computed for each point in the point cloud, as well as the respective distance, for two purposes: the first was to find outliers. Due to the way LiDAR technology works, erroneous data can be found in the point cloud due to the light pulse being absorbed or reflected by an object. Secondly, because the minimum spanning tree is a subgraph of the Delaunay triangulation, i.e., the nearest neighbor information can serve as a reference for the triangulation.

The problem resulting from this method is that due to the density of the points, there was more than one neighbor with the same distance for a said point in denser areas. In contrast, there was no neighbor at all in fewer dense areas, and it was necessary to recursively extend the neighbors' search radius for certain points. In Table 4.1 we can see the intervals of the distance to the nearest neighbor and its frequency.

To decide the number of data to use, we analyzed the frequencies by distance intervals with the nearest neighbor to decide the number of data to use. After this, we calculated the standard deviation between distance intervals. Then, we used this standard deviation with the average, the number of intervals, and a value of $\alpha = 0.005$ to obtain a confidence interval value of 99.95%. Finally, we considered the value of the largest number of frequencies, and 12 confidence intervals of $99,5\%$ were subtracted to obtain a significance limit of the information. It was indicated that the cut should be done at a frequency of 26984.57. We decided to include more points in the data set to triangulate, hence from $d > 4$, the points were considered outliers, and thus, the first filter was done. However it would be important to consider some scattered areas in the point cloud where more points may be needed.

Table 4.1: Nearest neighbor histogram for the whole point cloud

Interval	Frequency	Interval	Frequency	Interval	Frequency
[0.0 – 0.5)	1000614	[8.5 - 9.0)	48	[17.0 - 17.5)	1
[0.5 – 1.0)	844332	[9.0 - 9.5)	39	[17.5 - 18.0)	2
[1.0 – 1.5)	728719	[9.5 - 10.0)	22	[18.0 - 18.5)	2
[1.5 – 2.0)	202926	[10.0 - 10.5)	32	[18.5 - 19.0)	2
[2.0 – 2.5)	37507	[10.5 - 11.0)	14	[19.0 - 19.5)	2
[2.5 – 3.0)	10460	[11.0 - 11.5)	22	[19.5 - 20.0)	1
[3.0 – 3.5)	4238	[11.5 - 12.0)	12	[20.0 - 20.5)	1
[3.5 – 4.0)	1916	[12.0 - 12.5)	10	[20.5 - 21.0)	0
[4.0 – 4.5)	841	[12.5 - 13.0)	6	[21.0 - 21.5)	2
[4.5 – 5.0)	534	[13.0 - 13.5)	10	[21.5 - 22.0)	0
[5.0 – 5.5)	366	[13.5 - 14.0)	8	[22.0 - 22.5)	0
[5.5 – 6.0)	204	[14.0 - 14.5)	5	[22.5 - 23.0)	2
[6.0 – 6.5)	210	[14.5 - 15.0)	5	[23.0 - 23.5)	0
[6.5 – 7.0)	149	[15.0 - 15.5)	5	[23.5 - 24.0)	0
[7.0 – 7.5)	128	[15.5 - 16.0)	4	[24.0 - 24.5)	0
[7.5 – 8.0)	87	[16.0 - 16.5)	2	[24.5 - 25.0)	0
[8.0 – 8.5)	60	[16.5 - 17.0)	3	25.0	1

Computing the 10 nearest neighbors: This process was carried out to later analyze the terrain in pieces small enough to approximate a flat surface. We used the KD-tree implementation in C, which allowed us to make a ball around each point and work with the points that stayed inside. Afterward, we only had to sort them by distance using the bubble method. This step was done recursively with the following values for the radius $0.10, 0.20, 0.3, 0.5, 1, 2, 3, 4, 5, 7, 8, 10, 12, 15, 20, 25$. In the densest areas of the point cloud, a radius of 1 returned about 150 thousand points; hence there was the need to implement the restriction of allowing a maximum of 500 points to sort; if the number of points surpassed this value, we needed to reduce the radius.

As a result, we obtained 16 iterations. In the last iteration, there were points with less than ten neighbors. The points with less than three neighbors were considered possible outliers. They were left for further processing when the triangulation should be reviewed globally and because they will generate numerical problems when computing the variance and covariance matrix and the eigenvalues and eigenvectors.

Covariance Matrix: This part of the implementation was quite simple and smooth;

the only drawback was that for the last iteration of the ten nearest neighbors, in some cases, the index of the same point was filtered as a neighbor, which resulted in some numerical errors. This issue was easily solved by defining an index representing the ”non-neighbor” and remembering it for the subsequent steps. The next step was implemented in R, so it was necessary to adapt the output information in the correct format.

Eigenvalues and eigenvectors: The implementation in R is shown below. In each line are the values of the Covariance matrix by columns, and for each line, the eigenvalues and eigenvectors are computed and stored per line in separate files.

```
a <- read.csv("VC_matrix.csv", header = FALSE, sep = ",", dec = ".")
values<-c(0,0,0)
vectors<-c(0,0,0,0,0,0,0,0,0)
for (i in 1:2833489) {
  c <- a[i, , drop = FALSE]
  f1 <- a[i, 1:3, drop = FALSE]
  f2 <- a[i, 4:6, drop = FALSE]
  f3 <- a[i, 7:9, drop = FALSE]
  d <- as.matrix(f1, nrow=3, ncol=3, byrow = TRUE)
  d <- rbind(d, unlist(f2))
  d <- rbind(d, unlist(f3))
  ev <- eigen(d)
  val <- ev$values
  vec <- ev$vectors
  c1 <- vec[1:3, 1, drop = FALSE]
  c2 <- vec[1:3, 2, drop = FALSE]
  c3 <- vec[1:3, 3, drop= FALSE]
  G<-c(c1,c2,c3)
  vectors <- rbind(vectors,t(G))
  values <-rbind(values,t(val))
  print(i)
}
values <- values[-1, ]
vectors <- vectors[-1, ]
write.csv(values, file = "eigenval.txt",row.names = FALSE)
write.csv(vectors, file = "eigenvec.txt",row.names = FALSE)
```

Smooth index for tangent plane: Once the eigenvalues and eigenvectors for each point were obtained, we checked the smoothness index. In general, we obtained the table 4.2 results for the whole file. There was no value of $\alpha = 1/3$, but we obtained quite close values. We used a similar method as with the first filtering for the table 4.2b. We obtained the values corresponding to the maximum value within the intervals, the standard deviation, and the number of data. A value of $\alpha = 0.005$ was set to obtain a confidence interval of 99.95%. Finally, from the maximum value of frequencies of the smoothness index obtained, four confidence intervals of 99.95% were subtracted to obtain the limitation of the information. It was indicated that the cut should be done at a frequency of 39,03. With this information, we established a range: points for which their value of $\alpha < 0.36$ would use the tangent cone; if $0.36 \leq \alpha < 0.37$, we can use both tangent plane and tangent cone, and for $\alpha > 0.37$, we can use the tangent plane.

Table 4.2: α histogram for the whole point cloud

(a) Interval $[0.3 - 1.0]$

interval	frequency
$[0.3, 0.4)$	830
$[0.4, 0.5)$	94033
$[0.5, 0.6)$	920068
$[0.6, 0.7)$	871731
$[0.7, 0.8)$	441396
$[0.8, 0.9)$	231456
$[0.9, 1.0)$	273962
1.0	13

(b) Interval $[0.3 - 0.4]$

interval	frequency
$0.30 - 0.31$	0
$0.31 - 0.32$	0
$0.32 - 0.33$	0
$0.33 - 0.34$	0
$0.34 - 0.35$	1
$0.35 - 0.36$	5
$0.36 - 0.37$	31
$0.37 - 0.38$	86
$0.38 - 0.39$	268
$0.39 - 0.4$	439

Choosing Section 1,2 and 3: Sections 1, 2, and 3 were chosen to study the algorithm's validity. Section 1 corresponds to a relatively flat and fairly smooth area, corresponding to a part of the sea. Section 2 corresponds to a more mountainous terrain without much vegetation and maintains the differentiability of the terrain but with interesting areas of non-differentiability. Section 3 corresponds to an area with much vegetation; both treetops and the ground are represented in the point cloud; for this section, quite low and frequent values of α would be expected. Tables 4.3, 4.4, and 4.5 show each section's values of α and from analyzing them, we know these 3 sections can be triangulated using the tangent plane.

Projection points and R: From each of the points, we obtained the projections on the respective tangent plane, which was computed with the eigenvectors corresponding to the two largest eigenvalues. Hence it was necessary to consider only the x and y coordinates

Table 4.3: Values α in Section 1

Interval	value	α max	α min
(0.3 – 0.4]	0	1	0.501951
(0.4 – 0.5]	0		
(0.5 – 0.6]	3781		
(0.6 – 0.7]	5597		
(0.7 – 0.8]	2915		
(0.8 – 0.9]	703		
(0.9 – 1.0]	86		
1.0	1		

Table 4.4: Values of α in Section 2

Interval	value	α max	α min
(0.3 – 0.4]	2	0.946453	min: 0.394795
(0.4 – 0.5]	408		
(0.5 – 0.6]	5956		
(0.6 – 0.7]	5126		
(0.7 – 0.8]	1368		
(0.8 – 0.9]	205		
(0.9 – 1.0]	18		
1.0	0		

Table 4.5: Values of α in Section 3

Interval	value	α max	α min
(0.3 – 0.4]	66	1	min: 0.360152
(0.4 – 0.5]	6174		
(0.5 – 0.6]	64788		
(0.6 – 0.7]	87648		
(0.7 – 0.8]	67202		
(0.8 – 0.9]	34073		
(0.9 – 1.0]	33998		
1.0	0		

to triangulate two dimensions using the tripack library in R with the tri.mesh function. Implementation is included.

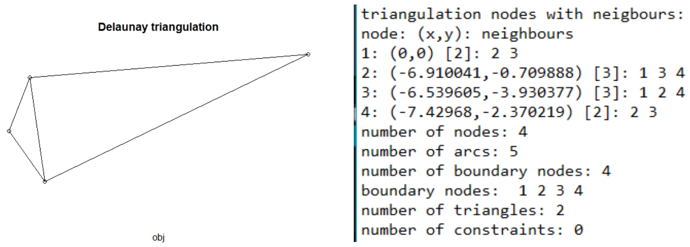
```
library(tripack)
a <- read.csv2("sec1_tr.csv", header = FALSE, sep = ",", dec = ".",
    stringsAsFactors = FALSE, encoding = "UTF-8")
#a <- read.csv("test.csv", header = FALSE, sep = ",", dec = ".")
# sec1 26162  sec2 154420  sec3 587774
values<-c(0,0)
vectors<-c(0,0,0,0,0,0,0,0,0)
for (i in seq(1,26162,2)) {
  print(i)
  cx1 <- a[i, , drop = FALSE]
  cy1 <- a[i+1, , drop = FALSE]
  cx <- t(cx1)
  cy <- t(cy1)
  obj <- tri.mesh(cx,cy, duplicate = "remove")
  sum <-summary(obj)
  numn <- sum$n
  numtr <- sum$nt
  num <- c(numn,numtr)
  values <-rbind(values,num)

  tr <- triangles(obj)
  vectors <- rbind(vectors,tr)

  capture.output( print(obj), file = "we_s1.txt",append = TRUE, type=
     c("output", "message"), split = FALSE)
}

values <- values[-1, ]
vectors <- vectors[-1, ]
write.csv(values, file = "sec1_n_tr.txt",row.names = FALSE)
write.csv(vectors, file = "sec1_t.txt",row.names = FALSE)
```

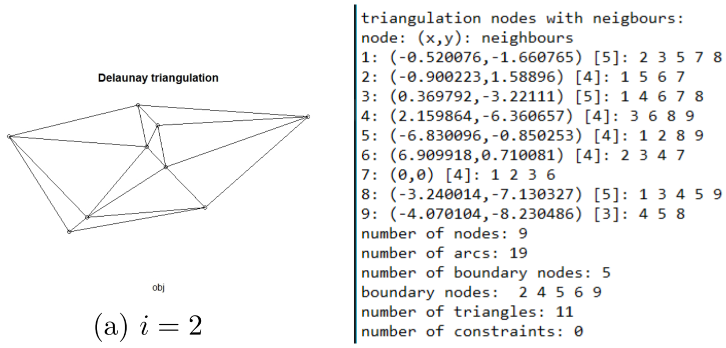
For one entry, i.e., a point, and its neighbors' projections on the tangent plane, we obtain the information on the nodes and its triangulation, which can be tracked back to the original points. In image 4.1 and image 4.2 we can see the Delaunay triangulation of the first two inputs corresponding to Section 1 point cloud and the respective data file used for the complete Delaunay reconstruction. Notice that with the "duplicate = "remove"" command in tri.mesh, we eliminate repeated values, this solves the problem if a point has missing neighbors or repeated ones.



```
triangulation nodes with neigbours:
node: (x,y): neighbours
1: (0,0) [2]: 2 3
2: (-6.910041,-0.709888) [3]: 1 3 4
3: (-6.539605,-3.930377) [3]: 1 2 4
4: (-7.42968,-2.370219) [2]: 2 3
number of nodes: 4
number of arcs: 5
number of boundary nodes: 4
boundary nodes:  1 2 3 4
number of triangles: 2
number of constraints: 0
```

(a) $i = 1$ (b) Information file

Figure 4.1: Delaunay triangulation of the projection of point i = 1

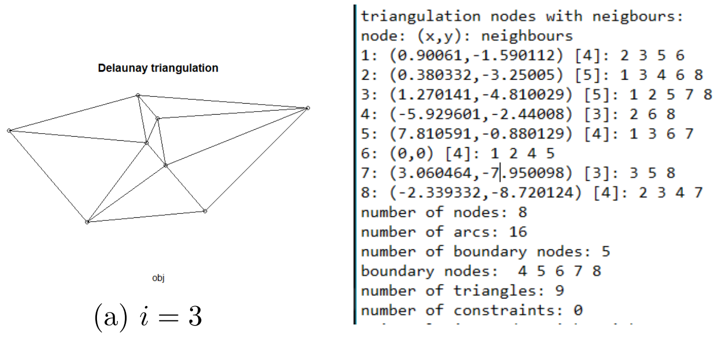


(a) $i = 2$

```
triangulation nodes with neigbours:
node: (x,y): neighbours
1: (-0.520076,-1.660765) [5]: 2 3 5 7 8
2: (-0.900223,1.58896) [4]: 1 5 6 7
3: (0.369792,-3.22111) [5]: 1 4 6 7 8
4: (2.159864,-6.360657) [4]: 3 6 8 9
5: (-6.830096,-0.850253) [4]: 1 2 8 9
6: (6.909918,0.710081) [4]: 2 3 4 7
7: (0,0) [4]: 1 2 3 6
8: (-3.240014,-7.130327) [5]: 1 3 4 5 9
9: (-4.070104,-8.230486) [3]: 4 5 8
number of nodes: 9
number of arcs: 19
number of boundary nodes: 5
boundary nodes:  2 4 5 6 9
number of triangles: 11
number of constraints: 0
```

(b) Information file

Figure 4.2: Delaunay triangulation of the projection of point i = 2



(a) $i = 3$

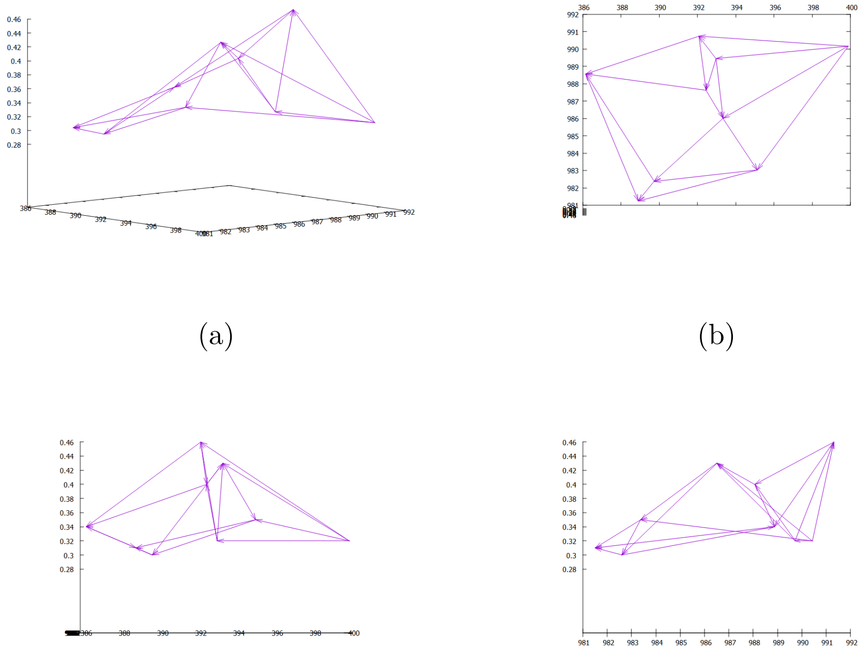
```
triangulation nodes with neigbours:
node: (x,y): neighbours
1: (0.90061,-1.590112) [4]: 2 3 5 6
2: (0.380332,-3.25005) [5]: 1 3 4 6 8
3: (1.270141,-4.810029) [5]: 1 2 5 7 8
4: (-5.929601,-2.44008) [3]: 2 6 8
5: (7.810591,-0.880129) [4]: 1 3 6 7
6: (0,0) [4]: 1 2 4 5
7: (3.060464,-7.950098) [3]: 3 5 8
8: (-2.339332,-8.720124) [4]: 2 3 4 7
number of nodes: 8
number of arcs: 16
number of boundary nodes: 5
boundary nodes:  4 5 6 7 8
number of triangles: 9
number of constraints: 0
```

(b) Information file

Figure 4.3: Delaunay triangulation of the projection of point i = 3

Delaunay triangulation: With the data obtained from the R algorithm, it remains to search for the original values keeping the relation with the triangles. It is not that evident for the whole set, so the result will be shown for the three points shown in Figures 4.1, 4.2 and 4.3, in Figure 4.4.

The Delaunay reconstruction of the whole sections can be visualized using the following link: https://drive.google.com/drive/folders/1zbDYU7ZJIccDSaLFJSN JwYdA1nWNdsZ D?usp=sharing.



(a) (b)

(c) (d)

Figure 4.4: Delaunay triangulation of the first three points of the set and their nearest neigbors

Chapter 5

Conclusions

Due to the study of the material corresponding to the Delaunay triangulation and the covariance matrix, an algorithm was developed to reconstruct surfaces defined by three-dimensional points. On the other hand, the information provided by the eigenvalues and eigenvectors of the covariance matrix of a nearby set of points helped to determine the smoothness level of the point surface locally; therefore, the smoothness of a terrain can be quantitatively analyzed using this method.

Additionally, a cloud of LiDAR points corresponding to one of the coasts of Ibiza was obtained. With this point cloud, the algorithm's implementation was mainly done in the C programming language, except for certain parts of the algorithm that require much more work and programming knowledge for which R was used. With R, the eigenvalues and eigenvectors of the covariance matrix were obtained, as well as the Delaunay triangulation of the points projections.

This “local” triangulation obtained from R is the basis of surface triangulation of the entire set and, therefore, terrain reconstruction. Finally, regarding the efficiency and accuracy of the algorithm, more tests and analyzes must be done to specify something more conclusive; due to the limitations of the computational capacity, the performance of the algorithm could not be studied in a better way since it will be more significant to evaluate on big point clouds. Moreover many parts of the implementation can be improved.

Bibliography

[1] T. Brinkhoff, “Geodatenbanksysteme in theorie und praxis-einführung in objektrelationale geodatenbanken unter besonderer berücksichtigung von oracle spatial (2. aufl.).” 2008.

[2] H. Bäcklund and N. Neijman, “Automatic mesh decomposition for real-time collision detection,” 2014.

[3] M. Eskandari and D. Laurendeau, “Covariance based differential geometry segmentation techniques for surface representation using vector field framework,” 2020.

[4] D.-S. Kim, P. Y. Papalambros, and T. C. Woo, “Tangent, normal, and visibility cones on bézier surfaces,” Computer Aided Geometric Design, vol. 12, no. 3, pp. 305–320, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016783969400016L

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Computational geometry: Algorithms and applications, 3rd ed. Berlin, Germany: Springer, 2010.

[6] S.-W. Cheng, T. K. Dey, and J. Shewchuk, Delaunay Mesh Generation. Filadelphia, PA, USA: Chapman & Hall/CRC, 2016.

[7] Apple. Apple maps introduces new ways to explore major cities in 3d. [Online]. Available: https://www.apple.com/newsroom/2021/09/apple-maps-introduces-new-ways-to-explore-major-cities-in-3d/

[8] Samsung. Cómo jetbot 90 ai+ de samsung está reinventando la limpieza. [Online]. Available: https://news.samsung.com/pe/como-jetbot-90-ai-de-samsung-esta-reinventando-la-limpieza

[9] T. D. Lee and B. J. Schachter, “Two algorithms or constructing a delaunay triangulation,” International Journal of Computer & Information Sciences, vol. 9, pp. 219–242, 1960.

[10] J. A. Barentzen, J. Gravesen, F. Anton, and H. Aanaes, Guide to computational geometry processing: Foundations, algorithms, and methods. London, England: Springer, 2012.

[11] E. Zancolo, “Minimum spanning trees using delaunay triangulation,” Master’s thesis, Klagenfurt University, July 2008.

[12] P. Maur, “Delaunay triangulation in 3d,” Master’s thesis, University of West Bohemia in Pilsen, January 2002.

[13] A. Okabe, B. N. Boots, K. Sugihara, and S. N. Chiu, Spatial tessellations: Concepts and applications of Voronoi diagrams, 2nd ed. London, Chichester, Inglaterra: John Wiley & Sons, 2000.

[14] H. Edelsbrunner, T. S. Tan, and R. Waupotitsch, “An $o(n^2 \log n)$ time algorithm for the minmax angle triangulation,” SIAM journal on scientific and statistical computing, vol. 13, pp. 994–1008, 1992.

[15] H. Edelsbrunner, Cambridge monographs on applied and computational mathematics: Geometry and topology for mesh generation series. Cambridge, Inglaterra: Cambridge University Press, 2006.

[16] G. Toussaint, “Efficient triangulation of simple polygons,” The visual computer, vol. 7, pp. 280–295, 1991.

[17] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, “Triangulating a simple polygon,” Information processing letters, vol. 7, pp. 175–179, 1978.

[18] S. Marschner, P. Shirley, M. Ashikhmin, M. Gleicher, N. Hoffman, G. Johnson, T. Munzner, E. Reinhard, W. B. Thompson, P. Willemsen, and B. Wyvill, Fundamentals of computer graphics, 4th ed. Boca Raton: A K Peters/CRC Press, 2021.

[19] J. R. Shewchuk, "Tetrahedral mesh generation by delaunay refinement," in Proceedings of the fourteenth annual symposium on Computational geometry, 1998, pp. 86–95.

[20] W. H. Frey, "Selective refinement: A new strategy for automatic node placement in graded triangular meshes," International Journal for Numerical Methods in Engineering, vol. 24, no. 11, p. 2183–2200, 1987.

[21] L. P. Chew, "Guaranteed-quality triangular meshes," CORNELL UNIV ITHACA NY DEPT OF COMPUTER SCIENCE, Tech. Rep., 1989.

[22] T. K. Dey, C. L. Bajaj, and K. Sugihara, "On good triangulations in three dimensions," in Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, 1991, pp. 431–441.

[23] W. Schaap and R. Van De Weygaert, "Continuous fields and discrete samples: reconstruction through delaunay tessellations," arXiv preprint astro-ph/0011007, 2000.

[24] B. N. Delone, "Sur la sphere vide," Bull. Acad. Science USSR: Class Sci. Math., pp. 793–800, 1934.

[25] (NOAA). What is lidar. [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html

[26] L. A. Wasser. The basics of lidar - light detection and ranging - remote sensing. [Online]. Available: https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics

[27] Laser (las) file format exchange activities. [Online]. Available: https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities

[28] J. Soto. Lastools para trabajar con datos lidar de manera profesional. [Online]. Available: https://geoinnova.org/blog-territorio/lastools-para-trabajar-con-datos-lidar-de-manera-profesional/

[29] Lidar applications. [Online]. Available: https://www.newport.com/n/lidar-applications

[30] S. Tavani, A. Billi, A. Corradetti, M. Mercuri, A. Bosman, M. Cuffaro, T. Seers, and E. Carminati, "Smartphone assisted fieldwork: Towards the digital transition of geoscience fieldwork using lidar-equipped iphones," Earth-science reviews, vol. 227, p. 103969, 2022.

[31] J. Chan, A. Raghunath, K. E. Michaelsen, and S. Gollakota, "Testing a drop of liquid using smartphone lidar," Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies, vol. 6, pp. 280–295, 2022.

[32] H.-S. Harriman, D. Ahmetovic, S. Mascetti, D. Moyle, M. Evans, and P. Ruvolo, "Clew3d: Automated generation of o&m instructions using lidar-equipped smartphones," ACM, 2021.

[33] M. Skrodzki, "Neighborhood computation of point set surfaces," in Book: Masterarbeit am Institut fur Mathematik der Freien Universitat, 2014.

[34] J. L. Bentley, "K-d trees for semidynamic point sets," in Proceedings of the sixth annual symposium on Computational geometry, 1990, pp. 187–197.

[35] H. Tabia, H. Laga, D. Picard, and P.-H. Gosselin, "Covariance descriptors for 3d shape matching and retrieval," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 4185–4192.

[36] I. Shafarevich, Basic algebraic geometry 1, 3rd ed. Moscow, Russia: Springer, 2014.