Trabajo de Fin de Grado

Visor web 3D de nubes de puntos

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Juan Auserón Delgado

TUTORIZADO POR:
Agustín Rafael Trujillo Pino

Julio 2024

Agradecimientos

Agradecer primero de todo a mi tutor Agustín, por toda su paciencia y esperanza en este proyecto. Que aunque desapareciera del mapa durante periodos prolongados, siguió confiando en mí y en este proyecto.

A mi familia por apoyarme en este duro camino desde el primer año de carrera hasta el último, confiando en mí y en mi palabra de seguir adelante con todo.

A mis amigos por soportarme decir durante mucho tiempo que mañana terminaré el trabajo de fin de grado, para quedarme atascado durante semanas y no dar señales de vida.

A todos ustedes y muchos más les doy mil gracias por acompañarme en este viaje.

Resumen

Este Trabajo de Fin de Grado consiste en realizar un visor web 3D que permita visualizar una muestra de nubes de puntos capturadas con tecnología **LiDAR** mediante su reestructuración en Quadtrees, permitiendo diferentes niveles de detalle.

Además de permitir navegar sobre esas nubes de puntos e interactuar con ellos mediante una interfaz de usuario, como por ejemplo, aplicar un filtro a la nube de puntos por categoría o intensidad. Este proyecto podría ser de utilidad para aquellos usuarios que quieran observar y personalizar una muestra grande de nubes de puntos

Abstract

This Final Degree Project involves the development of a 3d web visor that allows the user to visualize a sample of a cloud point obtained with **LiDAR** technology that has been restructured into Quadtree, allowing different levels of detail.

Moreover, it allows the navigation of these cloud points and the interaction with them through an user interface, e.g., applying a filter to the cloud points to show the classification or intensity of each point. This project could be useful for those user who wants to observe and personalize the sample.

Índice general

1. Introducción **1**
- 1.1. Metodologías y recursos empleados . . . 2
- 1.2. Motivación . . . 2
- 1.3. Objetivos del proyecto . . . 2
- 1.4. Estado actual . . . 3
- 1.5. Organización del documento . . . 5

2. Análisis del problema **7**
- 2.1. Estudios previos y decisiones . . . 7
- 2.2. Profundización en la estructura y uso de Quadtree . . . 7
  - 2.2.1. Octree . . . 8
  - 2.2.2. Quadtree . . . 9
  - 2.2.3. Estructuras de Potree y Arena4D . . . 9

3. Tecnologías y librerías usadas **11**
- 3.1. Librerías para la aplicación web . . . 11
  - 3.1.1. JavaScript . . . 11
  - 3.1.2. Node.js . . . 11
  - 3.1.3. React . . . 11
  - 3.1.4. Three.js . . . 12
- 3.2. Librerías para la aplicación backend . . . 12
  - 3.2.1. Python . . . 12
  - 3.2.2. Laspy . . . 12
  - 3.2.3. Numpy . . . 12
- 3.3. Estructura de ficheros LIDAR . . . 13

4. Competencias específicas y aportaciones del trabajo **15**
- 4.1. Competencias trabajadas . . . 15
- 4.2. Aportaciones . . . 16

5. Desarrollo **17**
- 5.1. Casos de uso . . . 17
- 5.2. Desarrollo inicial . . . 18
- 5.3. Reestructuración en Quadtree . . . 20

5.4. Tratado de puntos . . . 25
5.4.1. Clasificación . . . 25
5.4.2. Intensidad . . . 27
5.4.3. Altura . . . 28
5.4.4. Color . . . 28
5.4.5. Material, vertexShader y fragmentShader . . . 29
5.5. Interfaz de usuario . . . 32
5.5.1. React . . . 32
5.5.2. Eventos y llamadas comunicación entre Componentes . . . 36
5.5.3. Interacción con el mapa de puntos . . . 38
5.5.4. Selección de puntos . . . 38

6. Conclusiones y trabajo futuro **44**

7. Anexos **48**
7.1. Manual de aplicación web . . . 48
7.1.1. Iniciar el proyecto . . . 48
7.1.2. Importar archivos . . . 48
7.1.3. Selección de puntos . . . 49
7.1.4. Ajustar intensidad de puntos . . . 49
7.1.5. Modificación de colores de clasificación . . . 50
7.1.6. Modificación de límites de altura . . . 52
7.1.7. Distancia de renderizado . . . 52
7.1.8. Cambios de color de fondo . . . 52
7.1.9. Perspectivas de cámaras . . . 52
7.2. Manual de aplicación generadora de Quadtree . . . 52
7.2.1. Preparación del programa . . . 53

Índice de figuras

1.1. Potree Viewer con el ejemplo por defecto y su configuración. 3
1.2. Ejemplo de interactividad en Potree Viewer 4
1.3. Página principal de plas.io 4
1.4. Página principal de Arena4D, propiedad de la compañía Veesus 5

2.1. Representación de un Octree con profundidad 2. 8
2.2. Ejemplo de representación de un Quadtree 9

5.1. Primera versión del proyecto del visor web. 19
5.2. Visualización por clasificación de puntos de un mapa. 20
5.3. Ejemplo de la organización de carpetas para un Quadtree. 23
5.4. Mapa de 521MB estructurado en Quadtree viendo solo el nodo base. 24
5.5. Mapa de 521MB estructurado en Quadtree viendo todos los nodos 24
5.6. Sectores del Quadtree. 25
5.7. Ejemplo de visualización con colores según los códigos de clasificación. 26
5.8. Ejemplo de visualización con colores según la intensidad de un punto. 27
5.9. Ejemplo de visualización con colores según la altura de los puntos. 28
5.10. Ejemplo de visualización con colores RGBA de un mapa. 28
5.11. Ejemplo de visualización cambiando el tamaño de los puntos a una más grande con coloreado según altura. 30
5.12. Interfaz inicial implementada con React 33
5.13. Interfaz completa de la aplicación web. 36
5.14. Punto único seleccionado en el mapa de puntos. 39
5.15. Configuración disponible en la interfaz de usuario para la selección de un punto. 40
5.16. Medición de la distancia entre dos puntos y su resultado. 40
5.17. Selección en grupo de puntos. 42
5.18. Cambio de atributos de un grupo de puntos seleccionado. 43

7.1. Zona para arrastrar o elegir el importar un archivo en la interfaz de usuario. 49
7.2. Modificar el rango del sensor para ajustarse a la intensidad. 50
7.3. Modificación de colores de clasificación en la interfaz de usuario. 51
7.4. Primer paso para iniciar la línea de comando 53
7.5. Segundo paso para iniciar la línea de comando 53
7.6. Tercer paso para iniciar la línea de comando 54
7.7. Ejemplo de inicio de script para la aplicación de Python 54

Índice de cuadros

3.1. Códigos de clasificación. [1] 14

Índice de Algoritmos

5.1. Clases Point y Rectangle 21
5.2. Clases controladora de los colores según el código de clasificación. 26
5.3. Creación del mapa completo de colores por código de clasificación 27
5.4. Creación del material y sus uniformes 29
5.5. Definición del Vertex Shader. 29
5.6. Definición del Fragment Shader. 30
5.7. Representación del componente menu. 33
5.8. Código html que representan los componentes de la selección de botones. . . 34
5.9. Bloque que realiza la lógica para la selección de punto y llama al método del componente Thre.js para realizar la lógica de visualización. 37
5.10. Bloque que controla si el boton shift está siendo pulsado. 37
5.11. Bloque que controla si mientras el botón shift está pulsado se está arrastrando el ratón por una zona. 38
5.12. Como se crea y obtiene el Raycaster de Three.js y como se obtiene el punto del mapa. 39
5.13. Evento que inicia la selección en grupo de puntos. 41
5.14. Evento que expande el área de selección en grupo. 41
5.15. Evento que termina el área de la selección. 41

Capítulo 1

Introducción

Poco a poco, la tecnología **LiDAR** va expandiendo sus fronteras. Desde la medición y análisis de la topografía en zonas necesitadas, a la conducción autónoma de coches eléctricos que ya parecen cosas del futuro.

Para empresas de distribución de energía eléctrica, como Endesa o Iberdrola, la necesidad de obtener y analizar la topografía alrededor de sus campos eléctricos, es una actividad fundamental para evitar problemas o accidentes mayores. Es por esto que la tecnología **LiDAR** es muy valiosa para estas empresas.

El **LiDAR** obtiene mapas de puntos obtenidos mediante el barrido de un terreno con un dispositivo de teledetección montado a un vehículo aéreo. Realizando esto en sus propios tendidos eléctricos, se obtiene un mapa 3D detallado de estos y sus alrededores, permitiendo a la empresa visualizar y capturar posibles ocurrencias en sus tendidos eléctricos, además de dar la posibilidad de automatizar varias de sus tareas en cuanto a detección de problemas o al análisis de los terrenos.

Estos mapas pueden ser de terrenos extremadamente extensos y teniendo en cuenta que un mapa muy pequeño puede llegar a los 100 kB de espacio en disco, un mapa de billones de puntos puede llegar a 1 TB de espacio en disco. Para mostrar esto en un visor interactivo se tiene que optar por algún método de muestreo que permita obtener más o menos detalle dependiendo de qué se está mirando en el mapa y qué no.

Este proyecto se concentra en desarrollar una aplicación web que permita observar grandes mapas de puntos con poca demanda de recursos, implementando niveles de detalle en el mapa de puntos para así obtener un rendimiento adecuado, además de mostrar un detalle de puntos suficiente. Para facilitar esta implementación de mejoras de rendimiento, se dividió el proyecto en dos partes, una parte que realiza una reestructuración de los puntos en una estructura **Quadtree**, y otra que contiene el entorno web, la lógica para controlar el mapa de puntos reestructurado y toda su interactividad.

1.1. Metodologías y recursos empleados

La metodología de trabajo usada para este proyecto es la metodología de **KANBAN**, aprovechando su naturaleza continua y su flexibilidad en cuanto a fechas de entrega. Esta metodología me permitió hacer entregas del proyecto en plazos irregulares pero completos, teniendo los contras de mi situación personal.

Acompañando a esta metodología, se hizo uso de la herramienta de **Trello**, que haciendo uso de su tablero, se organizaron las tareas necesarias de forma continua, marcando claramente el inicio, desarrollo y finalización de cada una.

Por último, se hizo uso de GitHub como controlador de versiones del desarrollo. Dicho GitHub se encuentra en este link para la aplicación web y este link para la aplicación generadora de Quadtree.

1.2. Motivación

Este proyecto fue elegido por la primitiva necesidad humana de expandir sus horizontes y de explorar lo desconocido. Al haber empleado varios años al desarrollo de proyectos orientados al backend, el deseo de aprender nuevas tecnologías y enfoques fue poco a poco agrandándose con el paso de los años. Entonces fue cuando, entre todas las ofertas que se ofrecían, este proyecto destacó por las herramientas en las que se tenía que trabajar. Ofrecía desarrollos y fronteras nuevas que hasta entonces no había escuchado, alimentando aún más el hambre de aprender y de emprender una aventura nueva y emocionante.

1.3. Objetivos del proyecto

El objetivo central de este proyecto es ofrecer una herramienta Web que permita visualizar los datos recogidos mediante **LiDAR** y que a su vez se pueda personalizar esa visualización. Además de permitir ver conjuntos de datos de tamaños considerables. Los objetivos principales de este proyecto, marcados en el TFT01, son los siguientes:

- Estudiar la tecnología de **Three.js** y las posibilidades que ofrece esta tecnología.
- Estudiar otros proyectos ya existentes en el mercado que tengan relación con el proyecto a desarrollar.
- Desarrollar el visor web mediante **Three.js** con la ayuda de otras librerías.
- Desarrollar la capacidad del visor de obtener, manipular y visualizar datos de ficheros obtenidos con **LiDAR**.
- Desarrollar la capacidad de que un usuario pueda manipular los datos visualizados con una interfaz de usuario.

- Mantener un rendimiento eficiente del uso de recursos para proporcionar un producto estable.
- Estudiar mejorías para el proyecto a desarrollar y comparar con productos ya existentes en el mercado.

1.4. Estado actual

Actualmente, existen varias aplicaciones destacables que hacen uso de **Three.js** para mostrar nubes de puntos en un entorno Web. Estas aplicaciones están increíblemente desarrolladas y extendidas. Además, algunas de estas aplicaciones ofrecen la posibilidad de inyectar su tecnología en tu propio proyecto de forma fácil y atractiva.

Una de estas aplicaciones es **Potree** [6], una aplicación gratuita y *Open source* con unas posibilidades excepcionales.

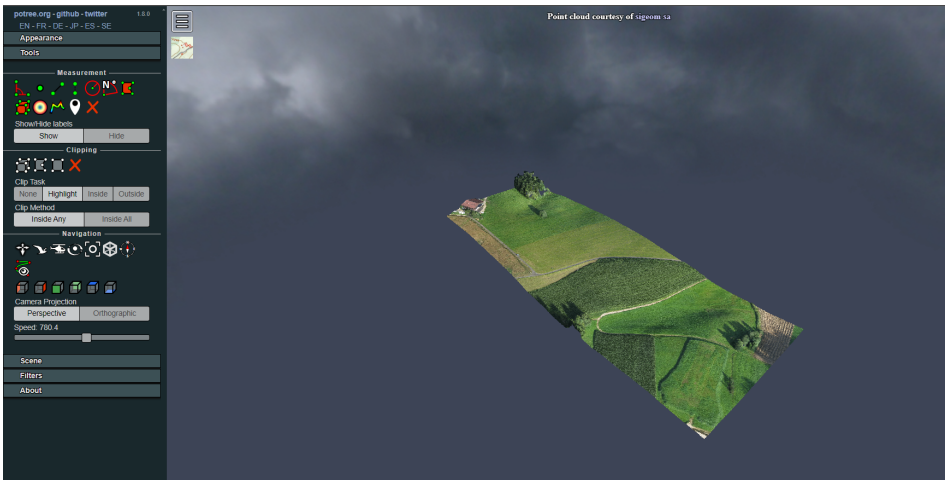

Ilustración 1.1: Potree Viewer con el ejemplo por defecto y su configuración. [https://potree.org/potree/examples/viewer.html]

Esta aplicación ofrece una gran base por la que cualquier usuario puede montar una aplicación web que permita mostrar mapas de puntos. Este también ofrece la posibilidad de funcionar como aplicación de escritorio o como aplicación web. Potree podría ser perfectamente la mejor base en cuanto a aplicaciones de visores de mapas de puntos, pero carece de algunas particulares que podrían haberse aprovechado, ya que hace uso de **Three.js**.

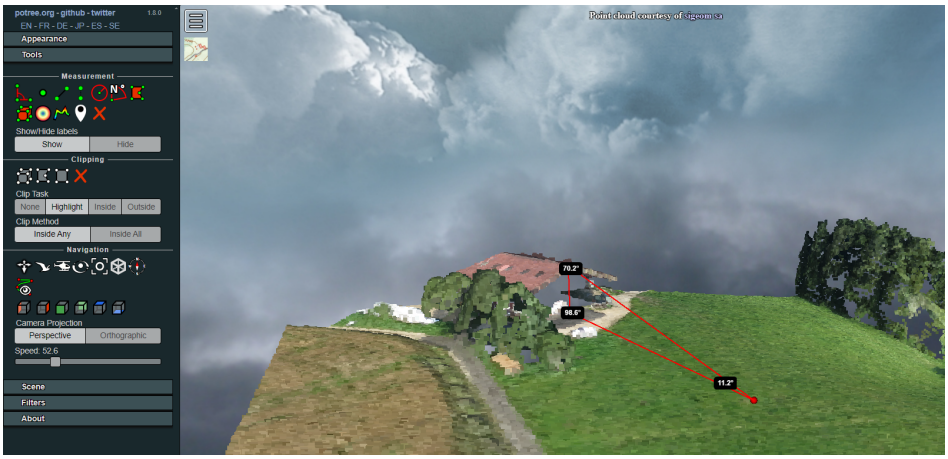

Ilustración 1.2: Ejemplo de interactividad en Potree Viewer [https://potree.org/potree/examples/viewer.html]

De todas formas, esta aplicación ofrece una de las mejores herramientas para la visualización de mapas de puntos de tamaños extraordinarios, esta es su PotreeConverter 2.0 [7], una herramienta que transforma mapas de puntos con billones de puntos en **Octree** para su visualización de forma más ligera y rápida.

Otra aplicación un poco menos destacable es **plas.io** [17].

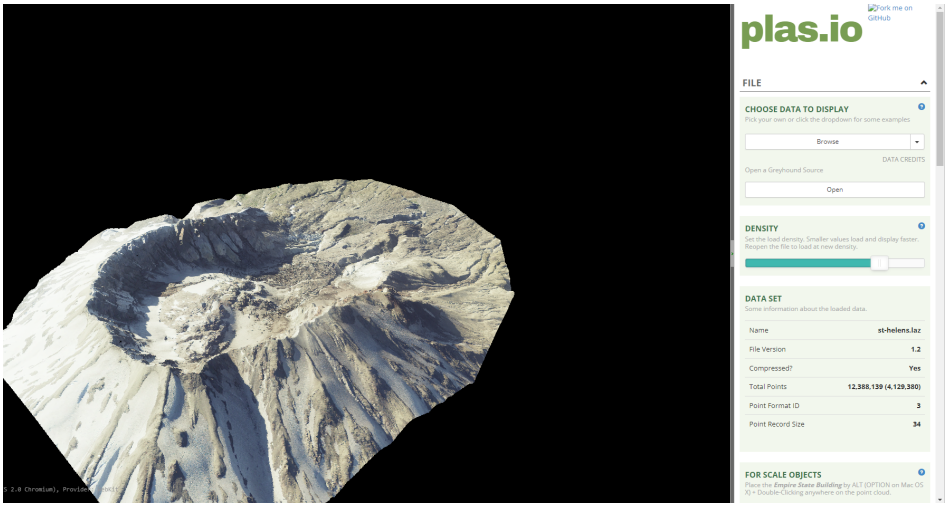

Ilustración 1.3: Página principal de plas.io [https://plas.io/]

Se trata de una web que permite visualizar mapas de puntos con una ligera configuración para manipular los puntos. Además de permitirte ver mapas ya cargados, te permite cargar tus propios mapas en la misma web. Puntos negativos de esta aplicación es la poca configuración que tiene, además de que, para aplicar algunas de sus configuraciones, necesitas recargar el mapa de puntos completamente. Además, la única forma posible de mejorar el rendimiento

del visor, es reduciendo el número de puntos a mostrar, ya que no realiza ningún tipo de nivel de detalle.

Existe otra aplicación de representación de mapas de puntos llamada Arena4D [14].

Ilustración 1.4: Página principal de Arena4D, propiedad de la compañía Veesus [https://www.veesus.com/arena4d/]

Esta aplicación, la cual es de escritorio, ofrece una personalización e interactividad extraordinaria. Permitiendo visualizar el mapa por colores, tipo de clasificación de cada punto, entre otros. Además, permite hacer operaciones interesantes, como simular la creación de sombras que podrían producir los objetos representados en el mapa, animaciones o incluso editar el mismo mapa de puntos. Un inconveniente de esta aplicación es que requiere un pago mensual o anual de 69,99 dólares estadounidenses mensuales para poder aprovechar todas sus funciones.

Ofrece una versión gratuita llamada VPC Viewer [16] que aporta una versión muy ligera del programa principal con funciones mínimas. También existe un convertidor gratuito llamado VPC creator [15], que transforma archivos, por ejemplo con formato .LAS, en su formato dedicado .VPC, ya que estos visores solo aceptan su formato dedicado para la correcta funcionalidad del mapa de puntos.

1.5. Organización del documento

Esta memoria se organiza en 7 capítulos y varios anexos:

- Capítulo 1 - Introducción.

  Capítulo que contiene una leve descripción del proyecto, sus objetivos y el cómo y por qué se eligió la idea para este proyecto, además de las metodologías usadas durante el proyecto.

- Capítulo 2 - Estudio del arte.

  Capítulo en el que se habla de todos los estudios previos que se hicieron para el proyecto, y los que se hicieron durante el desarrollo del mismo.

- Capítulo 3 - Tecnologías y librerías usadas.

  Apartado donde se entra en profundidad en las diferentes herramientas utilizadas en el proyecto, según en qué parte del proyecto nos encontremos. Además de una breve introducción a la estructura de los archivos utilizados.

- Capítulo 4 - Competencias específicas y aportaciones del trabajo.

  En este capítulo se hablará sobre las competencias desarrolladas en este proyecto y los beneficios diferenciables de este proyecto sobre otros del mercado.

- Capítulo 5 - Desarrollo.

  En este capítulo se explicará con más detalle el funcionamiento del proyecto y sus matices.

- Capítulo 6 - Conclusiones y trabajo futuro

  En este apartado final, se darán las conclusiones obtenidas del desarrollo y el trabajo futuro que podría llevarse a cabo en este proyecto.

- Anexo 1 - Manual de aplicación web

- Anexo 2 - Manual de aplicación de estructuración de datos

Capítulo 2

Análisis del problema

2.1. Estudios previos y decisiones

Dado al desconocimiento de las herramientas necesarias para desarrollar este proyecto, se requirió de realizar un estudio previo para así poder agilizar el desarrollo.

Se realizó una etapa de duración considerable de investigación sobre **Three.js** y todas sus posibilidades, haciendo grandes incisos en los objetos relacionados con la representación de nubes de puntos e incluso la generación de mundos, este último ayudando a la implementación de la estructura de un **Quadtree**. También se investigó sobre las peculiaridades de **React** para ofrecer una interfaz en una página.

Durante el estudio de las estructuras **Quadtree**, se dio a notar las dificultades de computar dinámicamente esta estructura directamente en un entorno web, por lo tanto, se decidió dividir el proyecto en dos partes. Una parte que contiene la aplicación web y otra que procesara los mapas de puntos antes de ser introducidos en la aplicación web.

Esta aplicación web contendrá la lectura del mapa reestructurado y la capacidad de interacción del mismo. Y la otra aplicación web se tratará de una aplicación sin interfaz de usuario que realice la adaptación y reestructuración de los mapas de puntos.

2.2. Profundización en la estructura y uso de Quadtree

Como ya se mencionó en la sección anterior, se realizó un estudio sobre la estructura **Quadtree** para implementar niveles de detalle o **LOD**, abreviación de *Levels of detail*.

Primero se intentó observar otras aplicaciones que implementasen este tipo de estructuras:

- **Plas.io** no hace uso de estructuras de datos como **Quadtree** u **Octree**.
- **Potree** hace uso de una estructura **Octree** para la representación de sus mapas de puntos.

- **Arena4D** hace uso de una estructura **Octree** para la representación de sus mapas de puntos.

Antes de profundizar en el estilo de cada una de las estructuras utilizadas, primero vamos a definir dos tipos de estructuras importantes para este proyecto

2.2.1. Octree

Un **Octree** es una estructura de datos que se basa en la división de una región en ocho divisiones iguales. Esto continuará hasta que todos los datos hayan sido organizados, o hasta que se llegue a un número máximo de divisiones.

Este tipo de estructuras se denominan estructuras de árboles, ya que cada nodo representa un *árbol* cuyas subregiones se denominan *hijos*. Un *hijo* se denomina como *hijo hoja* si este ya no tiene más nodos hijos.

Esta estructura trabaja en entornos de tres dimensiones, a causa de que divide el espacio obtenido en octantes de tamaños iguales, dando como resultado un cubo dividido en ocho zonas.

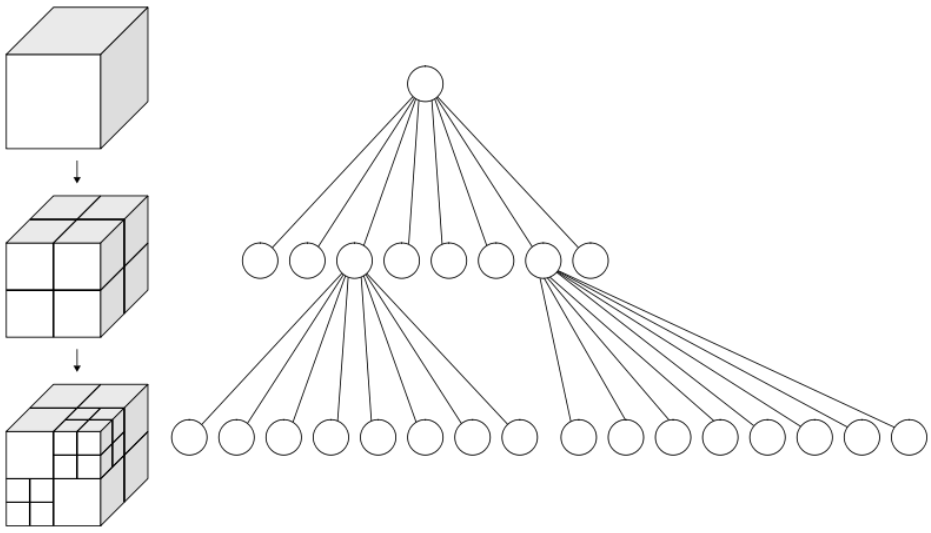

Ilustración 2.1: Representación de un Octree con profundidad 2.
[https://es.wikipedia.org/wiki/%C3%81rbol_octal]

Los **Octree** son usualmente usados para la mejora de rendimiento en cuanto al renderizado y visualización de modelos 3D, gracias a su naturaleza de dividir el espacio en volúmenes y no en planos como hace la estructura de **Quadtree**.

2.2.2. Quadtree

Un **Quadtree** es una estructura de datos igual a un **Octree**, pero este se diferencia en el número de subdivisiones de cada región. Mientras que en un **Octree** se hacen divisiones de regiones en ocho subregiones iguales, esta estructura divide regiones en cuatro subregiones iguales.

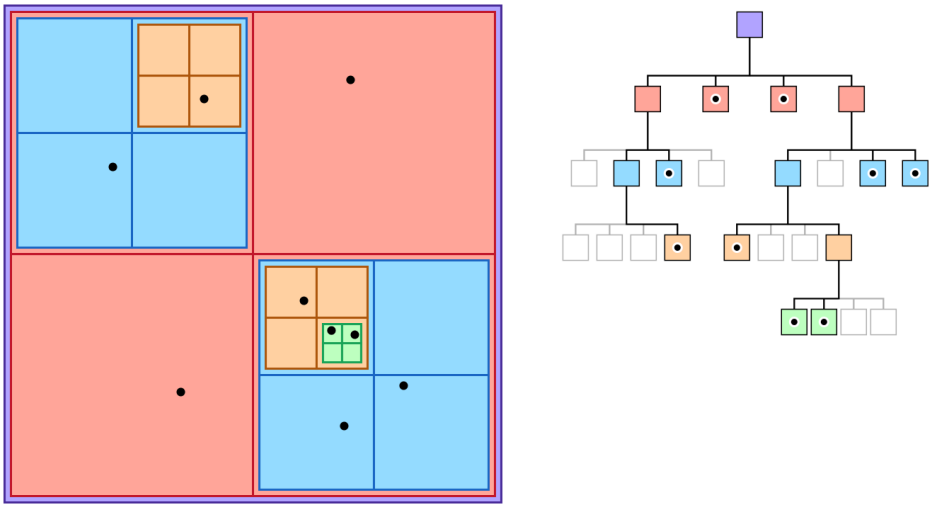

Ilustración 2.2: Ejemplo de representación de un Quadtree
[https://github.com/bikemap/BMQuadtree]

Este tipo de estructuras de datos son de gran utilidad para mejorar la eficiencia de entornos de dos dimensiones. Un ejemplo de su uso es el renderizado de terrenos en videojuegos. Esto es porque, al entrar en un mundo muy extendido, como suelen pasar en juegos de mundo abierto, es muy costoso cargar todo el mundo en máximo detalle. Por lo tanto, se utiliza esta estrategia para cargar con detalle únicamente la región donde se encuentra el jugador.

Para lograr esto, se utiliza la posición del jugador como punto de referencia. Este punto se sitúa en el **Quadtree** y se profundiza en la región a la que más cercana del centro se encuentra. Se seguirá profundizando recursivamente hasta que se llegue al hijo hoja más pequeño.

Para este proyecto, se tuvo muy en cuenta una implementación hecha por SimonDev [8], que propuso un ejemplo de generación de mundo mediante **Quadtree** en un entorno web, con el uso de workers de JavaScript. Permitiendo una generación con cargas de forma paralela y fluida.

2.2.3. Estructuras de Potree y Arena4D

Como ya se comentó anteriormente, **Potree** y **Arena4D** hacen uso de estructuras de datos para realizar una submuestra de mapas de puntos. Ambos usan la estructura de **Octree** pero sus implementaciones difieren ligeramente, pero tienen los mismos errores entre ellos.

Potree realiza un sub muestreo mediante Poisson-Disk. Este algoritmo trata la entrada de puntos dependiendo de la distancia entre ellos. Si la distancia es muy pequeña, el punto a añadir es descartado, si la distancia es mayor a una distancia determinada, se acepta el punto en la muestra. Este algoritmo es simple y muy veloz, pero provoca patrones en la muestra notables visualmente. Además, **Potree** añade los puntos sin haberlos ordenado previamente, lo que puede provocar patrones aún más visibles si el orden de los puntos aportados no es favorable. En **Arena4D** también se muestran estos patrones visuales, dando la posibilidad de que use un sub muestreo de Poisson-disk, que también se ve afectado por el orden de los puntos aportados.

Con el paso de los años, **Potree** lanzó una nueva actualización a su conversor de ficheros para adaptarlos a **Octree** que mejora su visualización y su velocidad. Esta nueva estrategia fue lanzada con el nombre **PotreeConverter 2.0**.

Esta nueva estrategia se diferencia de las anteriores realizando una estructura **Out-of-core**, utilizando el espacio en disco disponible para guardar información en chucks del mapa preparados para su lectura y organización en **Octree**. Además, para esta versión se hizo uso de una selección de puntos aleatoria en su sub muestreo para indicar si un punto pertenece a una región o a una región *hijo*, evitando así los patrones visuales que resultando del sub muestreo mediante Poisson-disk [7].

Capítulo 3

Tecnologías y librerías usadas

3.1. Librerías para la aplicación web

Para la porción de la aplicación web, se usaron las siguientes librerías y tecnologías.

3.1.1. JavaScript

Este lenguaje fue el seleccionado por el tutor para poder utilizarse la librería de **Three.js**. Este lenguaje orientado a objetos da muchas facilidades para elementos dinámicos, característica que ayudó mucho al desarrollo del proyecto.

3.1.2. Node.js

Node.js es un entorno de ejecución basado en eventos de forma asíncrona, que sirvió en el proyecto como base para lanzar la aplicación a un servidor local. Esta herramienta se usó al principio, cuando no se hacía uso de React, ya que este mismo tiene otra forma de lanzar la aplicación a un servidor local.

3.1.3. React

Biblioteca de JavaScript de código abierto para el desarrollo de interfaces de usuario que facilita sustancialmente las aplicaciones web "en una sola página", esto es, cuando una aplicación web no se refresca con ventanas nuevas, al contrario que otras aplicaciones más convencionales. Además, ayuda considerablemente con la aportación de reactividad de una página web.

React está basado en **componentes**, estos son: "zonas independientes y reutilizables de código para elementos de interfaz de usuario. Estos representan partes diferentes de una página web y contienen su estructura y comportamiento" (Geeksforgeeks) [3]

3.1.4. Three.js

Three.js es una librería de JavaScript utilizada para crear y mostrar escenas con objetos 3D. De esta librería, se concentraron los esfuerzos en el objeto de *Points*. Este objeto es capaz de mostrar una gran cantidad de puntos a partir de varios atributos asignados a este objeto, siendo perfecto para el desarrollo de este proyecto.

3.2. Librerías para la aplicación backend

Por otro lado, la parte del proyecto, que se centra en la reestructuración de los mapas de puntos, se utilizaron las siguientes herramientas.

3.2.1. Python

Este lenguaje de alto nivel multiparadigmático es uno de los lenguajes de programación junto con Java y R. Este destaca por su versatilidad y su facilidad de aprendizaje, además de su simple sintaxis basada en espacios y no en delimitadores como Java.

Este lenguaje fue elegido por su facilidad de entender, aprender e implementar, además de tener gran soporte de librerías con las que poder expandir un proyecto. Esto alimentaba la necesidad de crear una sub parte del proyecto capaz de manipular grandes cantidades de dato y poder manipularlos para su uso en la parte de aplicación web.

3.2.2. Laspy

Librería de Python para la lectura, modificación y escritura de archivos LAS. Estos son los archivos obtenidos por dispositivos LiDAR. Esta librería facilita mucho el uso de estos archivos y da la posibilidad de manipular libremente estos datos.

3.2.3. Numpy

Librería de Python que aporta funcionalidades y facilidades para la creación y manipulación de vectores y de grandes vectores multidimensionales. Todo esto con un gran soporte de operaciones matemáticas para operar en ellas.

3.3. Estructura de ficheros LIDAR

Como ya se ha comentado anteriormente, los ficheros ofrecidos por dispositivos LiDAR en los que se contienen los datasets de nubes de puntos masivos. Estos ficheros tienen la terminación .LAS

Un fichero .LAS se compone de tres partes:

1. Encabezado

   Este contiene la información de todo el mapa de puntos, como puede ser el número total de puntos o el formato de los puntos guardados.

2. VLR

   Este apartado opcional se denomina como *"Variable Length Record"* o Registros de tamaño variable. Este puede contener dimensiones extras del mapa de puntos o una descripción, entre otros.

3. Registro de puntos

   En este apartado del archivo se encuentra la información guardada secuencialmente de cada punto. La información guardada en estos registros pueden cambiar según el formato de los puntos. Actualmente, existen 10 formatos de puntos. También existen versiones de archivos que son compatibles con determinados formatos de puntos, siendo cuanto más nuevos, más compatibilidades tiene. Estas versiones albergan de la versión 1.0 hasta la versión 1.4. Esta última versión tiene un bloque extra después del registro de puntos, que es el EVLR o *Extended Variable Length Record* que son lo mismo que los bloques VLR pero con más capacidad de cargamento, ya que permite el uso de uint64.

Otro apartado que debe de tener su propia mención, es la clasificación de puntos. Durante la versión 1.0, un punto podía caer en uno o más tipos de clasificación, la cual este formato no podía soportar. Fue cuando a partir de la versión 1.1 que se añadió un marcador más para indicar el estatus del punto [1].

Además, el indicador del tipo de clasificación que se le asigna al punto, está estandarizado por el **ASPRS** para aquellos archivos que tienen desde la versión 1.1 hasta la 1.4.

Cuadro 3.1: Códigos de clasificación. [1]

Valor de clasificación	Significado
0	Nunca clasificado
1	No asignado
2	Terreno
3	Vegetación baja
4	Vegetación media
5	Vegetación Alta
6	Edificio
7	Punto Bajo
8	Reservado
9	Agua
10	Ferrocarril
11	Superficie de la carretera
12	Reservado
13	Protector de cable (señal)
14	Conductor de cable (fase)
15	Torre de transmisión
16	Conector de la estructura de cables (aislante)
17	Plataforma del puente
18	Ruido alto
19-63	Reservado
64-255	Definible por el usuario

Capítulo 4

Competencias específicas y aportaciones del trabajo

4.1. Competencias trabajadas

Aquí hablaremos de las competencias realizadas en este proyecto.

- **CII017:** *Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.*

En el proyecto se diseñó una interfaz con la intención de que fuera visible y suficientemente detallada, pero que a su vez se diferenciase ligeramente de las aplicaciones que ya están en el mercado.

- **CII08:** *Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.*

En este proyecto el lenguaje seleccionado para el desarrollo fue indicado por el Tutor, este se trata de JavaScript, con el objetivo de hacer uso de las herramientas y librerías que este lenguaje de programación ofrece, sobre todo, para aprovechar el uso de **Three.js**.

- **IS01:** *Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.*

Durante el desarrollo del proyecto, se tuvo muy en cuenta el realizar código limpio siguiendo pautas de **Clean Code**, como puede ser el mantener una estructura correcta del código escrito o el correcto y significativo nombramiento de variables insertadas en el mismo.

4.2. Aportaciones

Este proyecto se diferencia de otros ofreciendo características que algunas de las aplicaciones destacables del mercado no ofrecen de primeras. Pero sirve como una buena base para futuras mejoras y nuevas características. Y la división del proyecto en dos partes le proporciona una modularidad perfecta para poder evolucionar el proyecto, incluso de forma paralela.

La principal idea de este Trabajo de Fin de Grado no era sobrepasar a las demás aplicaciones disponibles en el mercado, sino de ofrecer un punto de vista diferente de una tecnología que, se podría pensar, ya está bien extendida y examinada. Pero incluso con esto, existen pocas aplicaciones que ofrezcan lo que este proyecto ofrece principalmente:

- Un entorno web, fluido y rápido, que ofrezca una interacción considerable con un mapa de puntos, que además acepte diferentes formatos de los mismos.
- Una herramienta especializada en la reestructuración de mapas de puntos independiente y accesible.

Capítulo 5

Desarrollo

5.1. Casos de uso

En el estado actual del proyecto, la aplicación web tiene las siguientes funcionalidades:

1. Importar archivos LAS de nubes de puntos de tamaños pequeño, medio y grande.
2. Importar carpetas contenedoras del resultado de la reestructuración en **Quadtree**.
3. Seleccionar uno o un grupo de puntos con un cuadrado de selección personalizable del mapa de puntos.
4. Editar las características de estos puntos seleccionados, como puede ser su valor de intensidad, clasificación o color RGBA.
5. Editar los colores representativos de cada tipo de clasificación de punto
6. Mostrar y modificar un mapa de calor de la altura del mapa para una mejor visualización
7. Modificar el tamaño de los puntos representados.
8. Si el mapa representado es un **Quadtree**, mostrar con cubos la representación de los límites de cada una de las regiones que conforman el **Quadtree**. Además de aumentar o disminuir la distancia mínima requerida para renderizar de una región del **Quadtree**.
9. Medir la distancia desde un punto seleccionado a otro.
10. Desplazarse libremente por el mapa de puntos representado.
11. Visualizar el mapa de puntos según si se quiere ver la clasificación, intensidad, altura y color de cada uno de los puntos.
12. Cambiar las perspectivas de la cámara mediante botones asignados.

Por otro lado, la aplicación que reestructura los mapas en **Quadtree** tiene las siguientes funcionalidades:

1. Importar archivos LAS de nubes de puntos de tamaños pequeño, medio y grande.
2. Reestructurar archivos LAS en carpetas que representen cada una, una región del **Quadtree**.
3. Aceptar archivos con y sin apartado de colores RGB.

5.2. Desarrollo inicial

Una vez realizado un estudio extenso del estado actual2, se empezó montando el entorno de desarrollo. Para ello, se eligió el IDE **Visual studio code** por su relativa facilidad en programas escritos en JavaScript y anteriores experiencias en el uso de **React**. También fue necesario incluir la librería de **Node.js**, ya que la librería de **React** requiere de la misma para funcionar.

Una vez implantadas e inicializadas estas librerías en el proyecto, se añadió la librería de **Three.js**. Esto requería de la creación de cuatro componentes indispensables.

- Una escena donde poder mostrar el entorno de tres dimensiones.
- Una cámara con la que poder visualizar y controlar la perspectiva del entorno.
- Una fuente de luz para poder visualizar los objetos añadidos a la escena.
- Un objeto 3D a visualizar en la escena.

Una vez implementados estas pruebas, se comprueba que el entorno está montado correctamente y se continúa el proyecto. Este siguiente obstáculo se trataría de añadir el fichero en. LAS a **Three.js** para su visualización en la escena. Aquí empezó uno de los primeros contratiempos del proyecto.

Para la carga de archivos en formato .LAS se hizo uso de la librería *@loaders.gl/las*, que hace la lectura de estos archivos, y de la librería *@loaders.gl/core*, que se trata de la librería padre que acompaña a todos los importadores que tiene **loaders.gl**. Estas librerías tienen solo unas pocas características destacables a la hora de leer mapas de puntos con formato .LAS, como puede ser el cambiar el tipo de profundidad de color, pero no destaca por nada más.

Esta librería dio la oportunidad de obtener todos los datos de un mapa de puntos, pudiendo ahora leer todos los atributos que contiene y traspasarlos a objetos **Three.js** para su correcta visualización. Pero por su gran cantidad de información y el desconocimiento al uso de estas herramientas, provocó que el avance en este punto del proyecto se ralentizase, pero finalmente se obtuvo la solución a cómo obtener y visualizar estos mapas de puntos de esta manera.

El archivo obtenido del importador *@loaders.gl/las*, ofrecía un apartado de atributos que contenían los registros de los puntos en formatos de array con:

- Posición

- Clasificación
- Intensidad
- Color RGBA

Para poder ser tratados por **Three.js**, estos atributos deberán de ser transformados en un objeto de tipo **Points**. Este objeto 3D permite mostrar un conjunto de puntos en un solo contenedor, permitiendo acceder y modificar todos o grupos de puntos. Para poder crear este objeto de puntos, se requiere de un **material**, que describe la apariencia de los puntos [12], y una **geometría** que describe las características del mismo objeto, como puede ser la posición de los puntos [11].

Para la geometría de los puntos, se transformaron los datos obtenidos por el importador a una clase de propia de **Three.js**, que se trata de un BufferAttribute, la cual permite el almacenamiento de datos para un atributo y que además agiliza la comunicación con la GPU [10].

De primera mano, se decidió de forma temporal recorrer el array obtenido con el objetivo de adaptar la posición de cada punto teniendo el primer punto como posición de referencia [0,0,0] y adjuntarlo al **BufferAttribute** del objeto de tipo **Points**. Una vez añadido el objeto **Points** a la escena del proyecto, obtendremos la representación visual del mapa de puntos.

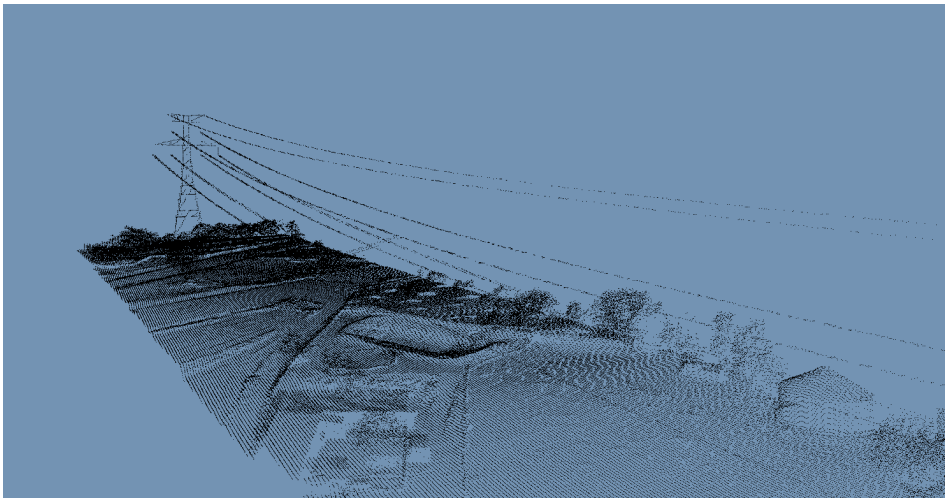

Ilustración 5.1: Primera versión del proyecto del visor web.

Al finalmente lograr mostrar los puntos obtenidos, se pasó a añadir otra información útil de los puntos, estos son la clasificación y la intensidad. Para ello, se realizó una metodología similar a la posición: Se recorre el array obtenido del fichero, se adjunta al **BufferAttribute** y se añade el atributo al objeto **Points**.

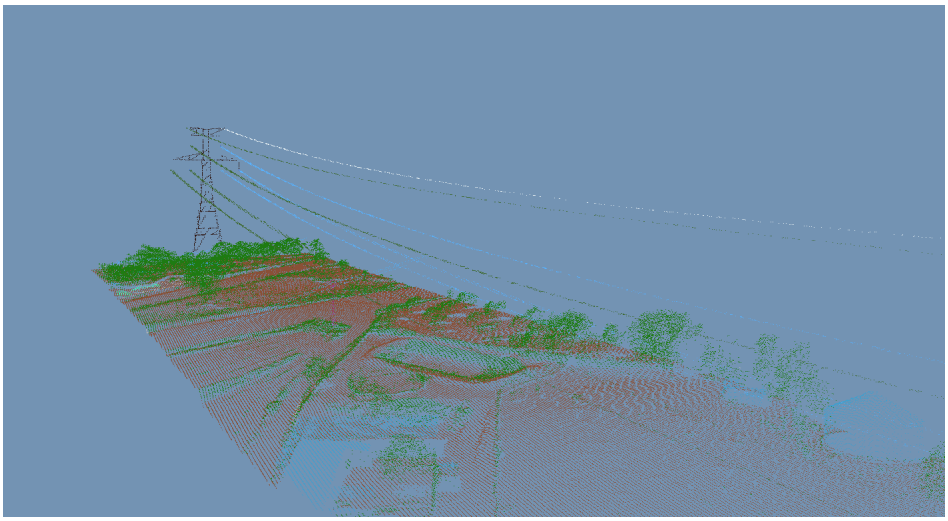

Ilustración 5.2: Visualización por clasificación de puntos de un mapa.

Esta forma de asignar los atributos de los puntos era funcional, pero para nada optimizada, ya que para mapas de puntos de tan solo 150.000 puntos, el programa tardaba considerablemente en cargar los puntos, ya que tenía que recorrer tres veces el mapa para obtener todos los puntos. Y con puntos de tamaños mediano-pequeños, la aplicación no lograba cargar los puntos y lanzaba una excepción de memoria insuficiente.

5.3. Reestructuración en Quadtree

Una vez implementada una versión base del proyecto, que aunque funcional, carecía de rendimiento suficiente, se continuó con el siguiente paso del proyecto: Usar una estructura de datos para mejorar la eficiencia del proyecto.

Tras haber investigado las otras aplicaciones similares del mercado, en un primer punto se decidió implementar una estructura de datos **Octree**. Esta estructura de datos se implementó directamente en JavaScript, en el que se definió un límite de puntos que debería de tener una región antes de poder considerarse lleno y un límite máximo de profundidad. Una vez se alcanzará el límite de puntos de una región, esta se dividía en ocho regiones y se marcaba como lleno. Si este nodo estaba lleno, pero se situaba en el límite de profundidad, se insertaba en este nodo igualmente.

Para rellenar este **Octree** se obtenía el número total de puntos a insertar y se hacía un subconjunto de objetos obteniendo un punto de cada n puntos. Siendo n el resultado de la división del número total de puntos por el número de puntos que debería tener cada región. Una vez rellenado el nodo raíz del **Octree**, se inserta de forma secuencial los demás puntos a insertar. Esto se hizo para obtener un primer nodo equitativo por todo el mapa y así evitar zonas muy pobladas del mapa y otras zonas con escasos puntos.

Continuando el proyecto, se decidió usar en vez de un **Octree**, usar un **Quadtree** debido

a su menor complejidad y similitud con otros programas de generación de mundos. Con esto, se implementó la estructura **Quadtree** que siguió la misma metodología para insertar puntos.

Para esta implementación se definió una clase **Point** y una clase **Rectangle**, siendo la primera una representación con de un punto único con toda la información necesaria, y la segunda la representación del área de una región del **Quadtree**. Esta clase **Rectangle** sería útil para separar las operaciones sobre las dimensiones de una región y si un punto pertenece a una región u a otra.

```
export class Point{
    constructor(x, y, z, r, g , b, intensity){
        this.x = x;
        this.y = y;
        this.z = z;
        this.r = r;
        this.g = g;
        this.b = b;
        this.intensity = intensity;
    }
}

export class Rectangle {

    constructor(x, z, width, height) {
        this.x = x;
        this.z = z;
        this.width = width;
        this.height = height;
    }

    contains(point) {
        return (
            point.x >= this.x - this.width &&
            point.x <= this.x + this.width &&
            point.z >= this.z - this.height &&
            point.z <= this.z + this.height
        );
    }

    intersects(range) {
        return !(range.x - range.width > this.x + this.width ||
            range.x + range.width < this.x - this.width ||
            range.z - range.height > this.z + this.height ||
            range.z + range.height < this.z - this.height);
    }

    distanceToCenter(x, z) {
        return Math.sqrt(((x - this.x) ** 2) + ((z - this.z) ** 2));
    }
}
```

Algoritmo 5.1: Clases Point y Rectangle

Para el cálculo de qué regiones se deberían mostrar en la escena , se hizo de forma que, obteniendo la posición de la cámara como punto de referencia, se profundizara si la posición de la cámara estaba dentro del cuadrante de la región y además estaba lo suficientemente

cerca del centro de la región. Si ambas eran correctas, se profundizaba en el **Quadtree** y se mostraban los puntos de ese nodo.

Esta implementación funcionaba con mapas de puntos pequeños y medianos, y ofrecía un rendimiento aceptable. Pero si se intentaba profundizar hasta el límite del **Quadtree** y este contenía un mapa lo suficientemente grande, el rendimiento se veía altamente afectado, ya que tenía que acceder una cantidad de hijos para comprobar si la cámara se situaba en su región, provocando ralentizaciones e incluso la terminación de la aplicación por exceder el número de llamadas consecutivas.

Para evitar este error se decidió crear los nodos del Quadtree conforme se vaya entrando en esas regiones. Esto ayudó a la aplicación a iniciar sin problemas, pero provocando atascos en la aplicación y la inevitable excedencia de llamadas y excepciones de uso de memoria.

En este punto del proyecto, se investigaron diferentes implementaciones de **Quadtree** en otros tipos de aplicaciones. Entonces fue cuando se encontró la implementación de Simon-Dev [8] basado en la generación de mundos extensos.

Este **Quadtree** se diferenciaba del desarrollado actualmente por el uso de **Workers**, esto son threads en paralelo que ejecutan partes de código. Para esta implementación se desarrolló un **WorkerPool** que contiene todos los workers que ejecutarán una parte del código en paralelo, y otra clase personalizada que contiene un worker y sus callbacks para un mejor tratamiento de datos.

Con la inclusión de los Workers, la aplicación web mejoró considerablemente en cuanto a rendimiento, ya que todos los procesos para crear nuevos nodos y añadirlos al **Quadtree** se pasaban a un hilo de desarrollo diferente. Lo que esto no solucionó este arreglo son los problemas de memoria, ya que se seguía sobrepasando el límite de memoria cuando se ejecutaba gran parte del **Quadtree**.

Para intentar solventar estos errores de memoria se decidió separar la creación del **Quadtree** a un programa completamente separado de la aplicación web. Dando total libertad de elección a un nuevo lenguaje y entorno.

Como el objetivo de cualquier Trabajo de Fin de Grado es aprender, se eligió el lenguaje **Python** para la aplicación que generará el nuevo **Quadtree**. Una parte que ayudó a la elección de este lenguaje es la fácil curva de aprendizaje que tiene y su accesibilidad. Además, se encontró de forma rápida una librería para leer y obtener todos los datos necesarios para importar archivos al programa Python. En esta aplicación aparte, se realiza el mismo proceso de creación de **Quadtree** pero en un entorno diferente, por lo que se tuvo implementar una forma de comunicar los datos obtenidos en esta aplicación a la aplicación web.

Como primera solución se pensó en preparar una API para hacer peticiones de tipo Rest y obtener los datos necesarios. Pero con la influencia de guardar la estructura de datos de forma **Out-of-core** como el desarrollado para PotreeConverter [7], se implementó en esta aplicación que el **Quadtree** se guardara en archivos dentro del disco duro. Para ello, se desarrolló que, cada vez que se terminase de rellenar una región del **Quadtree**, se guardara todos los atributos preparados para la aplicación web en un archivo JSON, que estará dentro de una carpeta con un nombre de máximo n dígitos, siendo n la profundidad máxima del

Quadtree. Luego, el dígito más a la derecha del nombre, representará el **Quadtree** al que pertenece, mientras que los que estén a la izquierda representarán el camino a seguir dentro del **Quadtree** para poder alcanzar el **Quadtree** objetivo. Si, por ejemplo, tenemos la carpeta con nombre "034", significa que, para acceder a este **Quadtree** debemos pasar por el nodo base 0, luego por el nodo 3 del nodo base y por último al nodo 4.

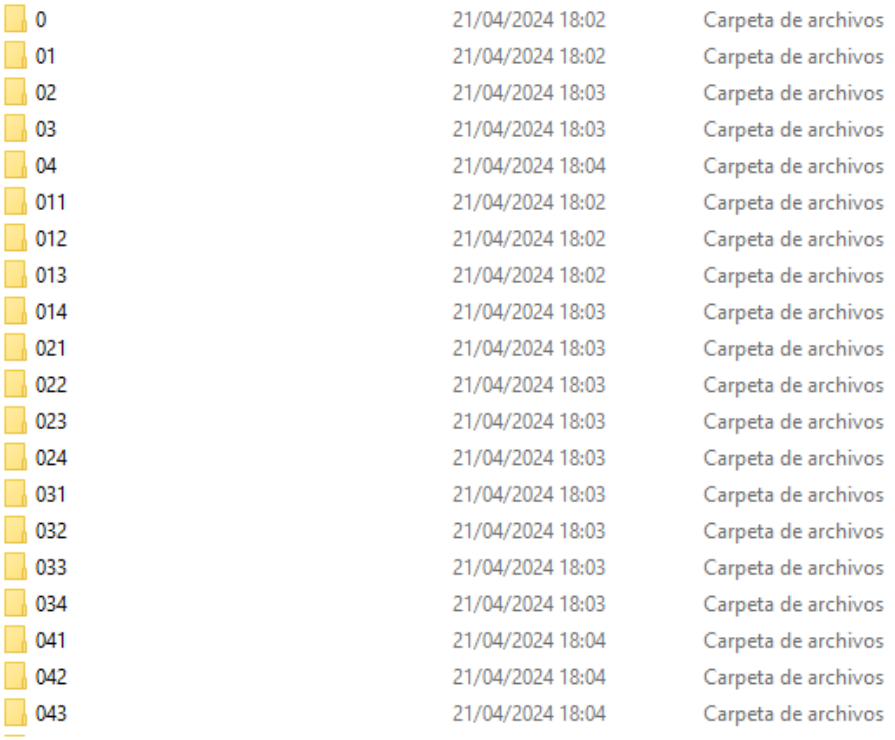

Ilustración 5.3: Ejemplo de la organización de carpetas para un Quadtree.

Tras esta implementación, se tuvo que adaptar el **Quadtree** original para que se cargue completamente a partir de los archivos importados. Esto se realizó usando promesas para obtener de forma paralela las regiones para renderizar los puntos. También se implementó que cada región del **Quadtree** guardase un objeto **Points** con un material en común para todo el **Quadtree**, para así poder controlar las representaciones de **Points** de forma unificada. Esto también facilita la muestra de puntos, ya que el objeto **Points** tiene un método que permite mostrar o esconder el objeto de la renderización de la escena.

Esto aporta a la aplicación web un rendimiento bastante adecuado, como ejemplo, con un mapa de más de 500 Mb, el visor renderiza entre 4 a 7 FPS, mientras que con nuestro **Quadtree**, obtenemos entre 50 y 57 FPS.

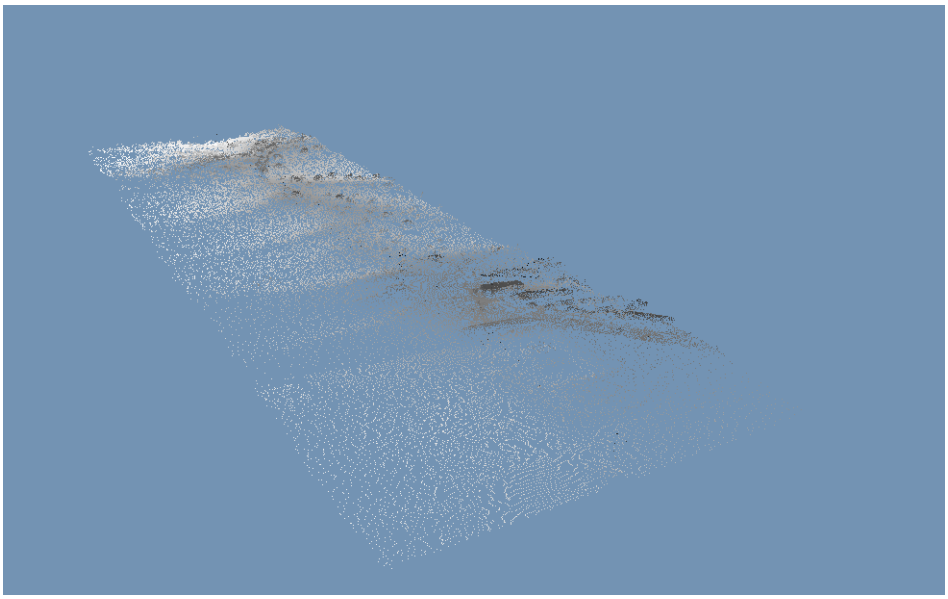

Ilustración 5.4: Mapa de 521MB estructurado en Quadtree viendo solo el nodo base.

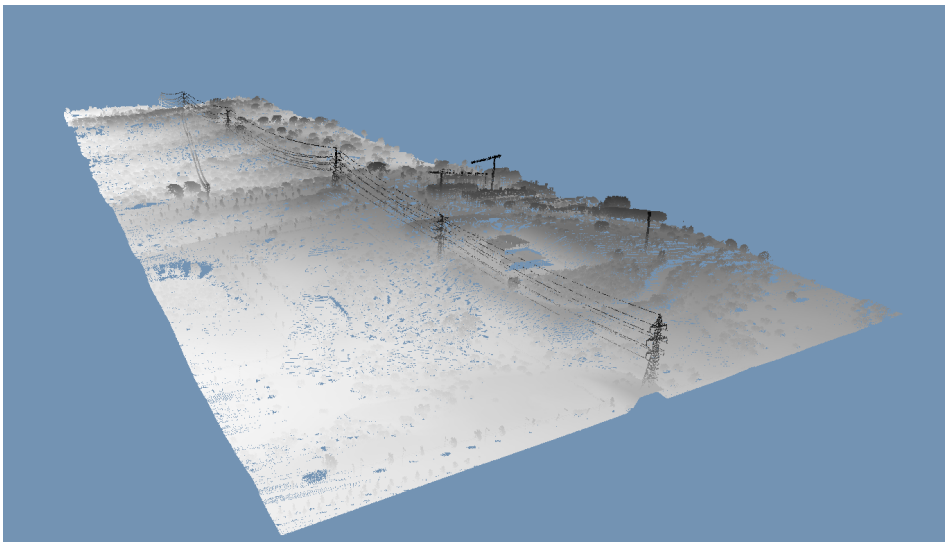

Ilustración 5.5: Mapa de 521MB estructurado en Quadtree viendo todos los nodos

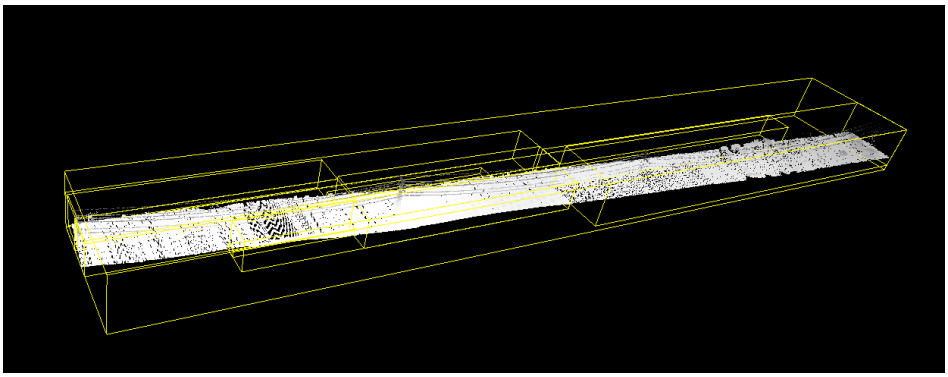

Ilustración 5.6: Sectores del Quadtree.

5.4. Tratado de puntos

Una vez obtenido un rendimiento correcto durante todo el tiempo de ejecución, el proyecto pasa a una nueva etapa donde se concentra en el tratado del registro de puntos y sus peculiaridades. En esta sección hablaremos más en profundidad de los atributos de cada atributo que ofrece los archivos .LAS.

5.4.1. Clasificación

Como ya hablamos en el apartado del estudio del arte 2, los ficheros .LAS tienen estandarizados los tipos de clasificación. Es por esto que se tiene que implementar en el proyecto web, alguna solución para poder obtener estos valores estandarizados para su representación. En versiones iniciales de la aplicación, se transformaba el número de la clasificación obtenido de los ficheros a un color RGB para su mejor visualización en la escena de la web, aumentando el tamaño del atributo por tres a consecuencia de que, por cada valor de clasificación, se tenía que dar tres valores de colores. Esto, aunque facilitase la lectura de los valores de clasificación, hacía más engorrosa la trata de datos, por lo que se creó un mapa que contiene como clave el tipo de clasificación, y como valor los tres valores de colores RGB.

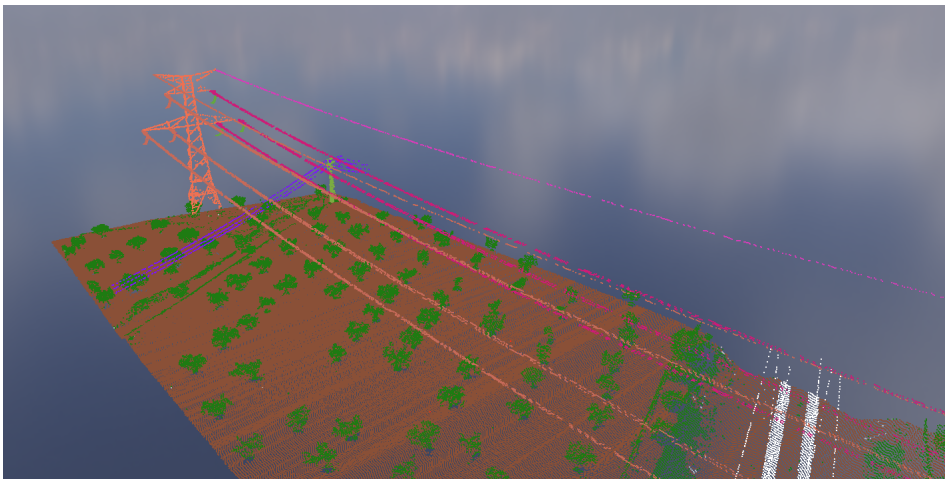

Ilustración 5.7: Ejemplo de visualización con colores según los códigos de clasificación.

Para los valores estandarizados mostrados en la tabla de los códigos de clasificación 3.1 se aportaron valores que intentaran representar al máximo la clasificación que representan. Para aquellos códigos de clasificación que no están aún estandarizados o han sido asignados por el usuario que ofrece el archivo .LAS, se asignó un color aleatorio RGB.

```
export class ClassificationColor {
    constructor(){
        this.map = new Map();
        this.map.set(0, [82, 82, 82]);
        this.map.set(1, [183, 183, 183]);
        this.map.set(2, [138, 80, 55]);
        this.map.set(3, [30, 123, 18]);
        this.map.set(4, [47, 187, 29]);
        this.map.set(5, [67, 248, 44]);
        this.map.set(6, [61, 161, 208]);
        this.map.set(7, [255, 44, 22]);
        this.map.set(8, [255, 236, 0]);
        this.map.set(9, [0, 138, 229]);
        this.map.set(10, [255, 255, 255]);
        this.map.set(11, [208, 167, 78]);
        this.map.set(12, [0, 255, 204]);
        this.map.set(13, [187, 0, 255]);
        this.map.set(14, [4, 0, 255]);
        this.map.set(15, [255, 136, 0]);
        this.map.set(16, [255, 0, 149]);
        this.map.set(17, [199, 156, 78]);
        this.map.set(18, [148, 0, 37]);
    }

    getColor(key) {
        if(this.map.has(key)){
            return this.map.get(key);
        }else{
            this.map.set(key, [ Math.floor(Math.random() * 255),
                        Math.floor(Math.random() * 255),
                        Math.floor(Math.random() * 255) ]);

            return this.map.get(key);
```

```
        }
    }
}
```

Algoritmo 5.2: Clases controladora de los colores según el código de clasificación.

```
const colorMap = new Float32Array(256 * 3);
const colorMapReserved = new ClassificationColor();
for (let i = 0; i < 256; i++) {
  colorMap[i * 3] = colorMapReserved.getColor(i)[0] / 255;     // R
  colorMap[i * 3 + 1] = colorMapReserved.getColor(i)[1] / 255; // G
  colorMap[i * 3 + 2] = colorMapReserved.getColor(i)[2] / 255; // B
}
for (let i = 11; i < 256; i++) {
  colorMap[i * 3] = Math.floor(Math.random() * 255) / 255;     // R
  colorMap[i * 3 + 1] = Math.floor(Math.random() * 255) / 255; // G
  colorMap[i * 3 + 2] = Math.floor(Math.random() * 255) / 255; // B
}
```

Algoritmo 5.3: Creación del mapa completo de colores por código de clasificación

5.4.2. Intensidad

La intensidad de recogida del registro de puntos es la fuerza de con la que el láser es reflejado en ese punto [2].

Estos valores pueden ir desde el 0 hasta el valor 65536, dependiendo del sensor que se esté utilizando.

En este proyecto, se ha hecho uso de una escala de negros grises para representar la intensidad de cada punto, siendo cuanto más alto es el valor, más blanco será su representación en la web.

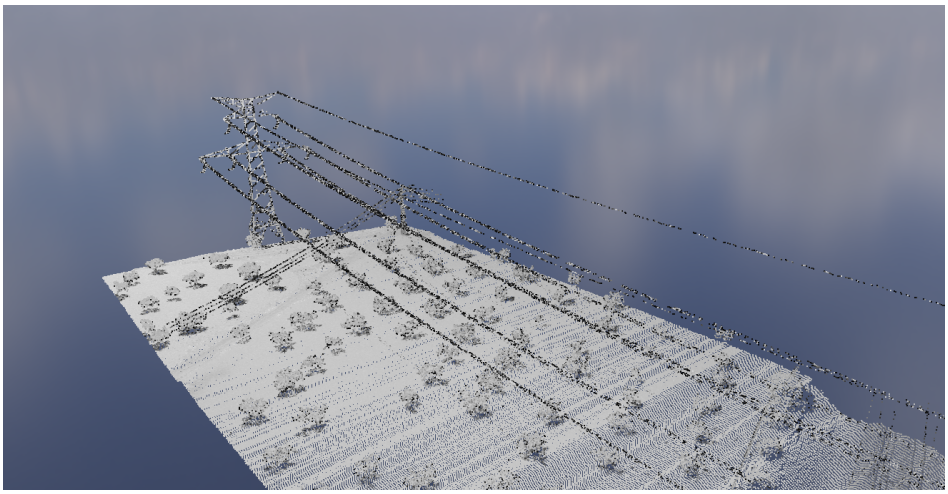

Ilustración 5.8: Ejemplo de visualización con colores según la intensidad de un punto.

5.4.3. Altura

Esta representación es la que se visualiza por defecto al iniciar o importar un fichero válido en la aplicación web. Este es una representación en escala de grises que muestra un color u otro dependiendo de la altura del punto en referencia a la altura máxima del mapa. Por defecto, si un punto está cerca de la altura máxima, más oscuro se verá.

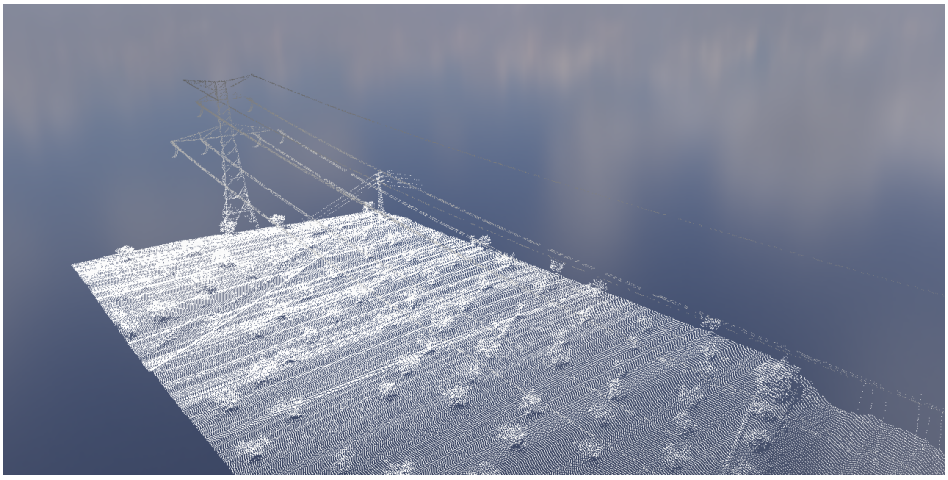

Ilustración 5.9: Ejemplo de visualización con colores según la altura de los puntos.

5.4.4. Color

Opcionalmente, los ficheros .LAS ofrecen un campo extra con información sobre el color de este punto. Normalmente, ofrecen los valores en formato RGB, en un rango de 0 a 255. En la visualización de los puntos, se tuvo que controlar si el fichero aportado ofrece dicha información. Si es así, se tratará esta información y se hará uso de la intensidad aportada anteriormente para crear conjuntos de colores RGBA, que permiten una mejor representación de los verdaderos colores de los puntos obtenidos.

Ilustración 5.10: Ejemplo de visualización con colores RGBA de un mapa.

5.4.5. Material, vertexShader y fragmentShader

Todos estos atributos mencionados anteriormente necesitan ser visualizados por el usuario. Esto se hace posible con la creación de un **Material** procedente de la librería de **Three.js**. Este **Material** contendrá toda la lógica que indica qué color y forma tomará cada uno de los puntos.

Para la creación de este **Material** se tiene que definir varias variables uniformes que son pasadas al código de los *shaders*.

```
lasMaterial = new THREE.ShaderMaterial({

    uniforms: {
      maxHeight: { value: maxHeight },
      realativeHeight: { value: materialParam.materialThreshold },
      negativeRelaHeight: { value: materialParam.negativeThreshold },
      pointTexture: { value: textureBall },
      isClassification: { value: materialParam.isClassification },
      isIntensity: { value: materialParam.isIntensity },
      isRGB: { value: materialParam.isRGB },
      pointSize: { value: materialParam.pointSize },
      maxHeight: { value: materialParam.maxHeight },
      r: { value: materialParam.r },
      g: { value: materialParam.g },
      b: { value: materialParam.b },
      r2: { value: materialParam.r2 },
      g2: { value: materialParam.g2 },
      b2: { value: materialParam.b2 },
      border: { value: materialParam.border },
      isLasOnly: { value: true },
      colorMap: {
        value: colorMap
      },
      intensityRange: { value: materialParam.intensityRange }


    },
```

Algoritmo 5.4: Creación del material y sus uniformes

Este código de shaders se divide en dos zonas, el **vertexShader** trata el procesamiento de cada uno de los vértices [5] del objeto al que pertenece el material, y el **fragmentShader** que trata los *fragmentos* obtenidos de la división de esquemas primitivos aportados al shader [4].

```
vertexShader: `
      varying vec2 vertexUV;
      varying vec3 vertexNormal;
      varying vec3 vertexPosition;
      varying vec3 classificationColor;
      varying vec4 rgbColor;
      varying float intensityColor;
      uniform float intensityRange;

      uniform float pointSize;
      uniform float colorMap[256];
      attribute float classification;
      attribute float intensity;
```

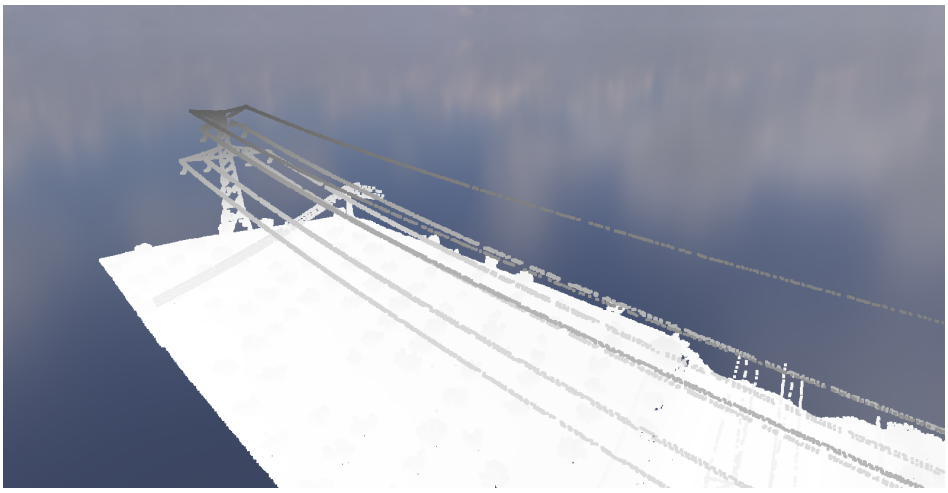

Ilustración 5.11: Ejemplo de visualización cambiando el tamaño de los puntos a una más grande con coloreado según altura.

```
      attribute vec4 COLOR_0;

      void main() {
        vertexPosition = position;
        vertexUV = uv;
        vertexNormal = normal;
        int mapValue = int(classification*3.0);
        classificationColor = vec3(colorMap[mapValue],colorMap[mapValue+1],colorMap[mapValue+2]);
        intensityColor = intensity/intensityRange;
        rgbColor = COLOR_0;

        gl_PointSize = pointSize;
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position , 1.0);
      }
      ‘,
```

Algoritmo 5.5: Definición del Vertex Shader.

En el bloque de código de **Vertex Shader**, preparamos los colores y características de cada uno de los puntos que más tarde, en el **Fragment Shader**, trataremos. Estas características pueden ser el tamaño del punto o los colores asignados a la clasificación. Para este último, ofrecemos al **Vertex Shader** el mapa de colores según el código de clasificación mostrado anteriormente, para así obtener los colores RGB necesarios para su representación.

```
fragmentShader: ‘
      uniform float maxHeight;
      uniform float realativeHeight;
      uniform float negativeRelaHeight;

      varying vec3 classificationColor;
      varying float intensityColor;
      varying vec4 rgbColor;


      uniform bool isClassification;
      uniform bool isIntensity;
```

```
uniform bool isRGB;
uniform bool isLasOnly;

uniform float r;
uniform float g;
uniform float b;
uniform float r2;
uniform float g2;
uniform float b2;
uniform float border;


varying vec2 vertexUV;
varying vec3 vertexNormal;
varying vec3 vertexPosition;

void main() {
  vec3 value =  mix(vec3(r,g,b), vec3(r2,g2,b2), smoothstep(negativeRelaHeight, realativeHeight, vertexPosition.y));

  vec2 c = abs( gl_PointCoord - vec2( 0.5 ) ) * border;
  float f = step( c.x, 0.6 ) * step( c.y, 0.6 );
  if(isClassification){
      float r = classificationColor[0];
      float g = classificationColor[1];
      float b = classificationColor[2];
      gl_FragColor = vec4(mix(vec3(0.0,0.0,0.0),vec3(r,g,b),f), 1.0);
  }else if(isIntensity){
    gl_FragColor = vec4(mix(vec3(0.0,0.0,0.0),vec3(intensityColor, intensityColor, intensityColor),f), 1.0);
  }else if(isRGB){
      float r = rgbColor[0]/255.0;
      float g = rgbColor[1]/255.0;
      float b = rgbColor[2]/255.0;
      float alpha = rgbColor[3]/255.0;
    gl_FragColor = vec4(r,g,b,alpha);
  }else {
    value = mix(vec3(0.0,0.0,0.0),value,f);
    gl_FragColor = vec4(value, 1.0);
  }
}
‘
```

Algoritmo 5.6: Definición del Fragment Shader.

En la zona de Fragment Shader, se encuentra la lógica que controla los colores de cada uno de los puntos. Aquí se seleccionará el tipo de visualización que se quiera ver según los parámetros asignados en los uniformes del material. Como se puede observar, en todas las opciones el array del color final se guarda en una variable llamada **gl_FragColor**, esta es una variable global que indica el color final de un punto [18]. Esta variable almacena un vector de tamaño cuatro, cuyos valores son RGB y un valor Alpha, con un rango de 0 a 1 aceptando decimales.

Otras características interesantes de este bloque es el uso de la función **smothstep** para el coloreo de puntos según su altura. Esta función realiza una interpolación polinómica de Hermite entre dos valores, estos es entre el 0 y el 1, cuando se sitúa dentro de unos límites. Para nosotros estos límites serán la altura máxima de nuestro mapa y la altura mínima. Como resultado obtenemos una transición fluida entre valores [19].

5.5. Interfaz de usuario

Con toda la personalización disponible, se entró en la etapa de implementar una interfaz de usuario donde poder interactuar de forma dinámica con el mapa de puntos. De forma inicial se implementó una interfaz de usuario de una librería llamada **dat.GUI** [Team], que proporciona una versión ligera capaz de modificar datos en nuestro proyecto JavaScript.

Como interfaz piloto, **dat.GUI** ofrecía de manera sencilla una mínima interfaz de usuario suficiente para la comprobación dinámica de la aplicación. Esta nos permitió adaptar el programa a cambios dinámicos de la aplicación en cuanto a todas las variables esenciales para la visualización de puntos, como puede ser el cambio de límites en la visualización de alturas, el cambio dinámico de tamaños de punto, o la selección del tipo de visualización que queremos renderizar.

Pero la simplicidad de esta interfaz no sería suficiente para un Trabajo de Fin de Grado. Aunque sea útil para programas pequeños o para programas iniciales, en nuestro caso requeríamos de una interfaz más ajustada para nuestros objetivos. Es por esto que se tomó la decisión de usar **React** para implementar la interfaz.

5.5.1. React

Se decidió utilizar **React** por su gran capacidad de reactividad y por su facilidad de crear interfaces de usuario en una sola ventana. Además de ser una librería de gran interés por su relevancia comercial actualmente, con grandes empresas como Airbnb y Microsoft haciendo uso del mismo.

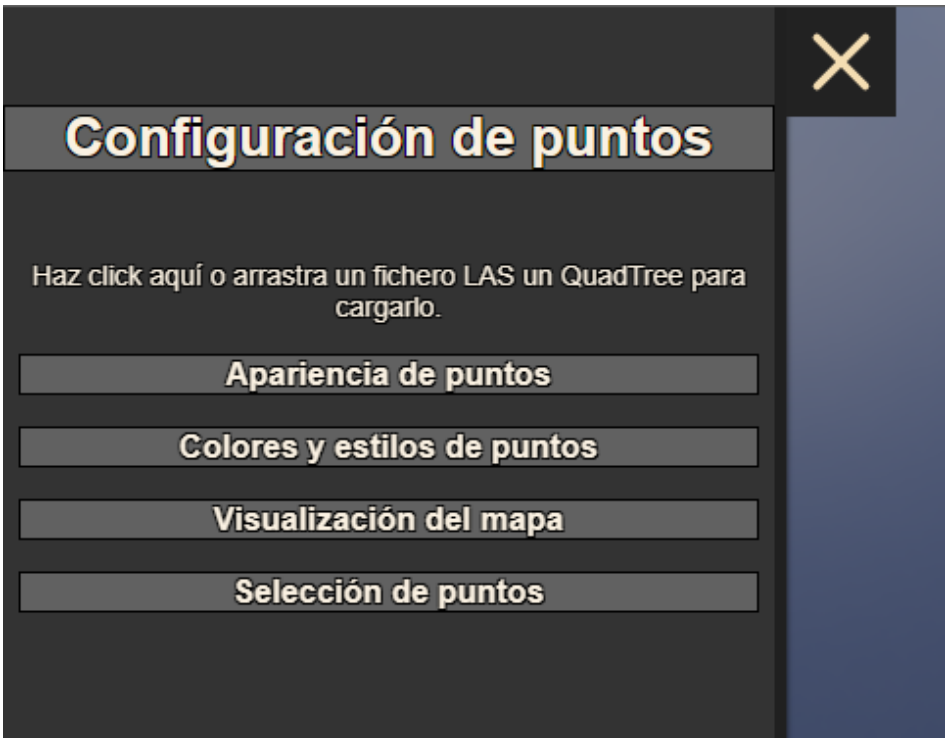

Ilustración 5.12: Interfaz inicial implementada con React

Esta librería basada en componentes, hace posible la independencia de distintos bloques de código, capaces de ser reutilizables por el resto del código. En nuestro proyecto, creamos varios componentes.

- Componente Three.Js:

Componente contiene toda la lógica relacionada con la visualización de la escena y comportamiento del entorno 3D. Este devuelve la visualización del entorno 3D.

- Componente Menu:

Componente que se utiliza para representar un menú de hamburguesa y contiene la lógica para su animación. Si se hace clic en el menú, este se desplazará a la derecha para permitir mostrar el contenido del menú y activará su animación.

```
window.addEventListener("load", function(e){
        const menu = document.querySelector('.hamburger-menu');
        const menuContent = document.querySelector('.off-screen-menu');
        menu.addEventListener('click', ()=>{
            menu.classList.toggle('active');
            menuContent.classList.toggle('active');
        });
  });

const Menu = () =>{
    return (
        <div className="hamburger-menu">
                <div className="menu menu-top">
                </div>
```

```
                <div className="menu menu-mid">
                </div>
                <div className="menu menu-bottom">
                </div>
            </div>
    )
}
```

Algoritmo 5.7: Representación del componente menu.

- Componente Menu_content:

Componente que contiene todos los elementos HTML que conforman el menú, además de sus operaciones y lógicas. Este es uno de los componentes más extensos, por detrás del componente de Three.js. En este componente es donde se situará la mayor parte de la interactividad del mapa de puntos.

Como ejemplo, tomaremos el menú interactivo de la selección de puntos. Estos son un grupo de puntos que determinan el tipo de selección de puntos que se quiere disponible o si se quiere medir una distancia entre dos puntos. Estos se conforman de tres puntos de tipo radio que permite la selección única de un tipo de selección.

```
<div className="menu-container-body hidden" id="selectPuntosContainer" >
                    <h3>Tipo de seleccion: <span id="pointType"></span></h3>
                    <br></br>
                    <div onChange={seleccionTipoPunto}>
                        <label className="btn btn-default-grey selected
                         btn-sm" id="buttonPuntoUnico">
                            <input type="radio" name="tipo" value="single"
                            style={{ display: "none" }} defaultChecked />
                            <img src="colors/singlePoint.png" width="100%" />
                        </label>
                        <label className="btn btn-default-grey btn-sm"
                        id="buttonPuntoGrupo">
                            <input type="radio" name="tipo" value="group"
                            style={{ display: "none" }} />
                            <img src="colors/multipleSelect.png" width="100%" />
                        </label>
                        <label className="btn btn-default-grey btn-sm"
                        id="buttonPuntoMedir">
                            <input type="radio" name="tipo" value="measure"
                            style={{ display: "none" }} />
                            <img src="colors/ruler.png" width="100%" />
                        </label>
                    </div>
                    <br></br>
                    <div id="distanceContainer" className="hidden">
                        <hr class="dashed"></hr>
                        <h4 id="distanceValue">Distancia entre puntos: </h4>
                        <hr class="dashed"></hr>
                    </div>
                </div>
```

Algoritmo 5.8: Código html que representan los componentes de la selección de botones.

- Componente Menu_content:

Componente que contiene todos los elementos HTML que conforman el menú, además de sus operaciones y lógicas. Este es uno de los componentes más extensos, por detrás del componente de Three.js. En este componente es donde se situará la mayor parte de la interactividad del mapa de puntos. Aquí podemos:

- Cambiar la apariencia de los puntos.
- Cambiar el estilo de visualización de los puntos y personalizarlos.
- Cambiar la visualización del mapa. Esto es la distancia de renderizado, los sectores del **Quadtree**, el fondo de la escena o la perspectiva de la cámara.
- Cambiar el tipo de selección de puntos.

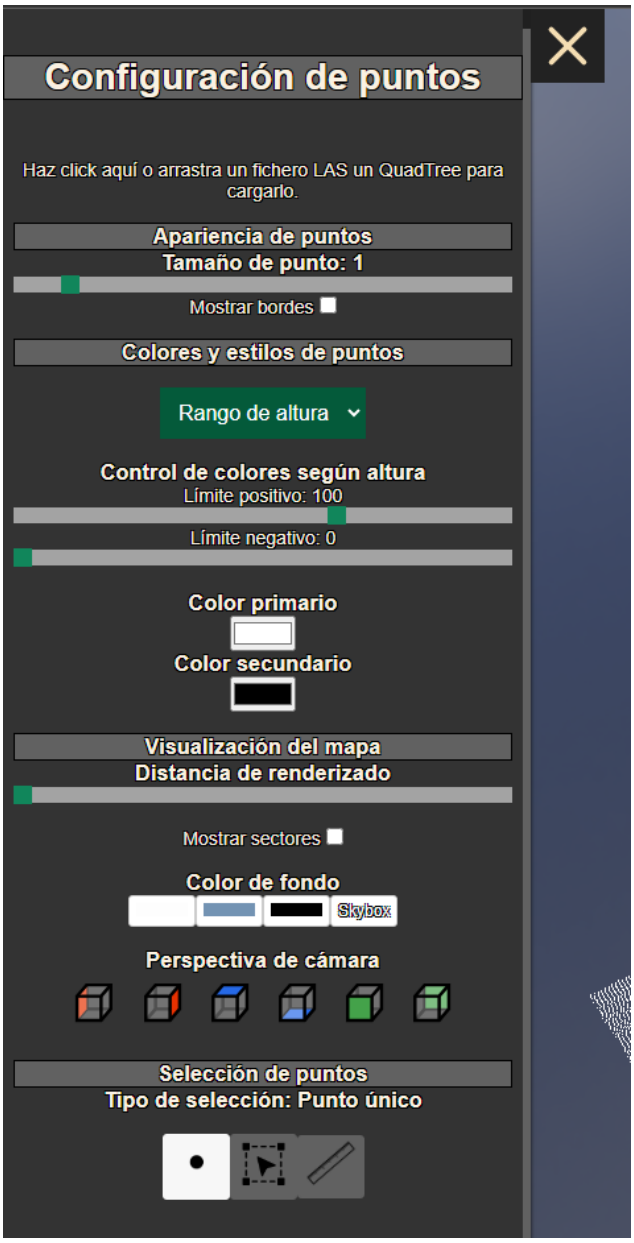

Ilustración 5.13: Interfaz completa de la aplicación web.

5.5.2. Eventos y llamadas comunicación entre Componentes

Con una interfaz base lista, se pasó a iniciar la comunicación entre el componente Three.js y el componente menu_content. Esto se llevó a cabo mediante la exportación e importación de variables globales y métodos. Esto se podría refactorizar para hacer uso de la comunicación que aporta React. Pero para simplificar la comunicación se implementó de esta manera.

Siguiendo el ejemplo de los botones de selección de puntos, tras aportar lógica que marca el botón como seleccionado, se llama a un método del componente Three.js que activará la

lógica de selección elegida en el botón.

```
function seleccionTipoPunto(event) {
    switch (event.target.value) {
        case "single":
            document.getElementById("puntoGroupSelectContainer").classList.add("hidden");
            document.getElementById("puntoGroupSelect").classList.add("hidden");
            document.getElementById("singleGroupTitle").textContent = "Punto seleccionado";
            document.getElementById("buttonPuntoUnico").classList.add("selected");
            document.getElementById("buttonPuntoGrupo").classList.remove("selected");
            document.getElementById("buttonPuntoMedir").classList.remove("selected");
            document.getElementById("distanceContainer").classList.add("hidden");
            pointType.innerHTML = "Punto unico";
            isSinglePointSelect = true;
            isMeasuredSelected = false;
            changeSelectionType();
            break;
        case "group":
            document.getElementById("puntoGroupSelectContainer").classList.add("hidden");
            document.getElementById("puntoGroupSelect").classList.add("hidden");
            document.getElementById("singleGroupTitle").textContent = "Grupo de puntos seleccionados";
            document.getElementById("buttonPuntoUnico").classList.remove("selected");
            document.getElementById("buttonPuntoGrupo").classList.add("selected");
            document.getElementById("buttonPuntoMedir").classList.remove("selected");
            document.getElementById("distanceContainer").classList.add("hidden");
            pointType.innerHTML = "Grupos de puntos";
            isSinglePointSelect = false;
            isMeasuredSelected = false;
            changeSelectionType();
            break;
        case "measure":
            document.getElementById("puntoGroupSelectContainer").classList.add("hidden");
            document.getElementById("puntoGroupSelect").classList.add("hidden");
            document.getElementById("singleGroupTitle").textContent = "Medicion entre dos puntos";
            document.getElementById("buttonPuntoUnico").classList.remove("selected");
            document.getElementById("buttonPuntoGrupo").classList.remove("selected");
            document.getElementById("buttonPuntoMedir").classList.add("selected");
            document.getElementById("distanceContainer").classList.remove("hidden");
            let distanceHeader = document.getElementById("distanceValue")
            pointType.innerHTML = "Medicion entre dos puntos";
            distanceHeader.innerHTML = "Distancia entre puntos: ";
            isMeasuredSelected = true;
            changeSelectionType();
            break;
    }
}
```

Algoritmo 5.9: Bloque que realiza la lógica para la selección de punto y llama al método del componente Thre.js para realizar la lógica de visualización.

Por otro lado, se requirió controlar los *inputs* que realiza el usuario con el ratón, si se quiere mover la cámara, y con el teclado, si quiere seleccionar un punto específico.

Es por esto que se implementaron *listeners* que detectan si un usuario ha hecho clic, arrastrado y soltado el botón, y si lo ha hecho pulsando el botón *shift* o no.

```
  function shiftDown(event) {
    if (event.key === 'Shift') {
      isShiftDown = true;
```

```
    }
  }

  function shiftUp(event) {
    if (event.key === 'Shift') {
      isShiftDown = false;
    }
  }
```

Algoritmo 5.10: Bloque que controla si el boton shift está siendo pulsado.

Esa detección del botón *shift* combinada con la detección de arrastre del ratón pulsado, quedaría tal que así:

```
function onMouseMove(event) {
    if (!isShiftDown) return
    if (!helper.isDown) {
      helper.pointBottomRight.set(event.clientX, event.clientY);
      selectionBox.endPoint.set(
        (event.clientX / window.innerWidth) * 2 - 1,
        -(event.clientY / window.innerHeight) * 2 + 1,
        0.5
      );
    }
  }
```

Algoritmo 5.11: Bloque que controla si mientras el botón shift está pulsado se está arrastrando el ratón por una zona.

5.5.3. Interacción con el mapa de puntos

selección de puntos

distancia

perspectivas Con las comunicaciones entre componentes establecidas, por último, se implementó las diferentes funcionalidades de la aplicación que permiten interactuar con el mapa de puntos. En este apartado mostraremos las principales interacciones disponibles en la aplicación.

5.5.4. Selección de puntos

Ya se ha comentado en bloques anteriores las comunicaciones y botones del menú de selección de puntos. Ahora se explicará el funcionamiento del mismo.

- **Punto único**

La primera implementación de selección fue la selección de un **punto único**. Esta se realizó con el uso de **Raycasters**, clase de **Three.js** que permite la selección de objetos mediante input de ratón. Este se basa en lanzar un *rayo* emitido por la posición del ratón en la pantalla. Una vez lanzado el rayo, este puede intersectar con otros objetos, que son

recogidos por el mismo objeto. Este rayo se puede configurar para aumentar o disminuir su tamaño, haciendo una selección más amplia o más específica.

Tras lanzar el rayo, se obtiene el primer objeto que interceptó el rayo, este es, el objeto más cercano al rayo y más cercano a la cámara. Este primer objeto será el índice en el objeto de **Points** obtenido del mapa de puntos. Si buscamos el índice en el objeto, obtendremos nuestro punto seleccionado.

```
const coordinates = new THREE.Vector2(
  (event.clientX / renderer.domElement.clientWidth) * 2 - 1,
  -((event.clientY / renderer.domElement.clientHeight) * 2 - 1)
);
raycaster.setFromCamera(coordinates, camera);

const intesercts = raycaster.intersectObjects(scene.children, true);
if (intesercts.length > 0) {

  try {

    if (intesercts[0].object.isPoints) {
      selectedPoints = [];
      scene.add(sphereClicked);
      const index = intesercts[0].index;
      const position = intesercts[0].object.geometry.attributes.position.array;
```

Algoritmo 5.12: Como se crea y obtiene el Raycaster de Three.js y como se obtiene el punto del mapa.

Para indicar que un punto ha sido seleccionado, se añade a la escena una esfera de color amarillo situada en el punto seleccionado.

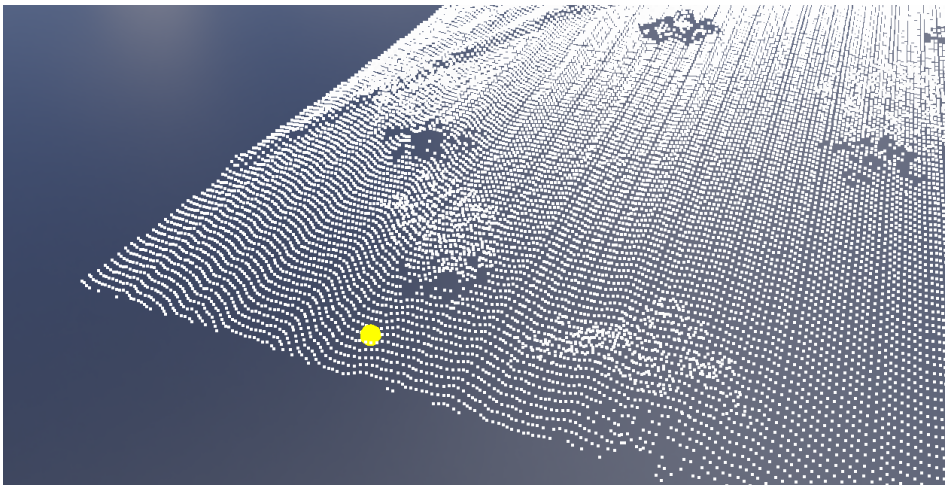

Ilustración 5.14: Punto único seleccionado en el mapa de puntos.

Una vez seleccionado un punto, podemos personalizarlo cambiando los atributos del mismo mediante la interfaz de usuario.

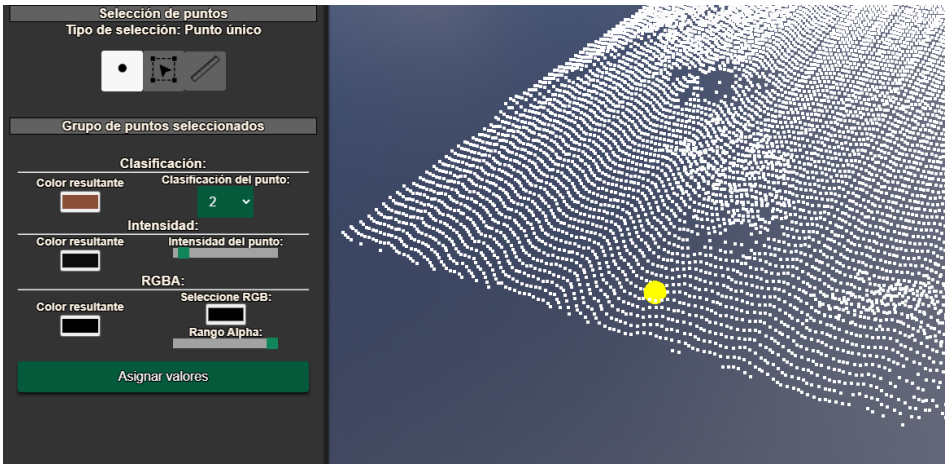

Ilustración 5.15: Configuración disponible en la interfaz de usuario para la selección de un punto.

- **Medir distancia entre puntos.**

Para implementar esta funcionalidad se aprovechó la implementación de selección única.

Primero se hace una selección igual a la selección única, pero este se guarda en un array que contendrá los dos puntos con el que medir la distancia. Cuando se seleccione otro punto diferente, este se guardará en el array que almacena los puntos y se creará un **Objeto 3D** representando una línea para su mejor visualización.

Luego, **Three.Js** ofrece un método que calcula la distancia entre dos vectores de tamaño tres llamado **distanceTo** [13]. Con esto, y teniendo en cuenta que en la posición de los puntos, se obtienen las posiciones de cada punto, se transforman en dos arrays con sus posiciones XYZ, y se obtiene el resultado del método. Este resultado se entiende que están en medida de metros.

Una vez obtenido, se muestra el resultado en la interfaz de usuario.

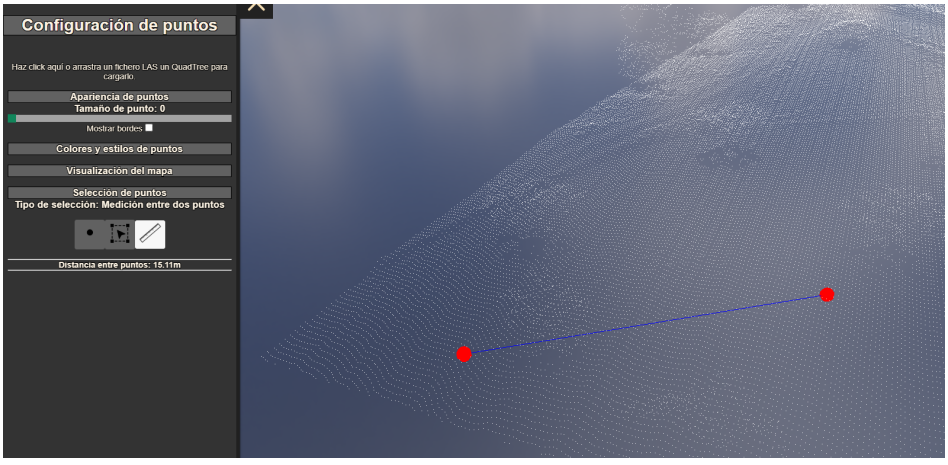

Ilustración 5.16: Medición de la distancia entre dos puntos y su resultado.

- **Selección en grupo**

Para la selección en grupo de puntos, se hizo uso de dos herramientas aportadas por **Three.Js**

- SelectionBox: Clase que realiza la selección de puntos en una escena.
- SelectionHelper: Clase que realiza ayuda visual para la selección de puntos.

Estas clases fueron fundamentales para conseguir esta funcionalidad. Para ello, se crearon los eventos que controlan los clics del ratón necesarios para su funcionamiento.

- Evento **mouseDown**

Cuando se realiza un clic en combinación a mantener presionada la tecla shift, empieza la selección de puntos, y aparece el **SelectionHelper** para ayudar visualmente al usuario a saber qué puntos está seleccionando.

```
helper.element.classList.add("selecting")
        controls.enabled = false;
        helper.pointTopLeft.set(event.clientX, event.clientY);
        selectionBox.startPoint.set(
          (event.clientX / window.innerWidth) * 2 - 1,
          -(event.clientY / window.innerHeight) * 2 + 1,
          0.5
```

Algoritmo 5.13: Evento que inicia la selección en grupo de puntos.

- Evento **mouseMove**

Mientras el usuario mantenga ambas teclas presionadas, este puede mover el ratón para indicar el área a seleccionar.

```
function onMouseMove(event) {
    if (!isShiftDown) return
    if (!helper.isDown) {
      helper.pointBottomRight.set(event.clientX, event.clientY);
      selectionBox.endPoint.set(
        (event.clientX / window.innerWidth) * 2 - 1,
        -(event.clientY / window.innerHeight) * 2 + 1,
        0.5
      );
    }
  }
```

Algoritmo 5.14: Evento que expande el área de selección en grupo.

- Evento **mouseUp**

Una vez soltado el botón del ratón, se entenderá que el usuario ha terminado de indicar el área a seleccionar y se guardarán los objetos **Points** de los puntos seleccionados en un de objetos **Points**. Si el mapa visualizado es un **Quadtree**, se obtendrán varios objetos **Points**, en caso contrario será un único objeto **Points**.

```
  function onMouseUp(event) {
    if (!isShiftDown || !helper.element) return
    if (!helper.isDown) {
```

```
      controls.enabled = true;
      selectedPoints = selectPoints(jsonQuadTree);
      helper.isDown = false;
      helper.element.classList.remove("selecting")
      pointGroupSelectedFunction(selectedPoints);
    }
    document.removeEventListener('mousemove', onMouseMove);
    document.removeEventListener('mouseup', onMouseUp);
}
```

Algoritmo 5.15: Evento que termina el área de la selección.

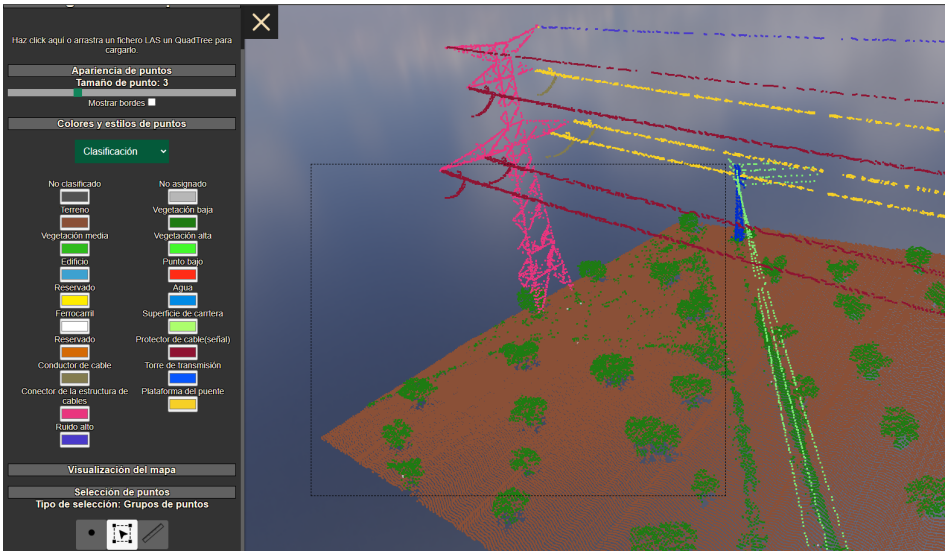

Ilustración 5.17: Selección en grupo de puntos.

Una vez obtenidos los objetos **Points**, se recorrerá el array de posiciones del objeto, comprobando si alguno de los puntos guardados se sitúan dentro del área indicada. Si es así, se guarda el punto con todos sus atributos en un array, en caso contrario no se hace nada con ese punto y se sigue recorriendo el array.

Una vez terminado este proceso, obtendremos todos los puntos con sus atributos en un array, los cuales podemos modificar como un grupo único, de igual forma que con la selección única de puntos.

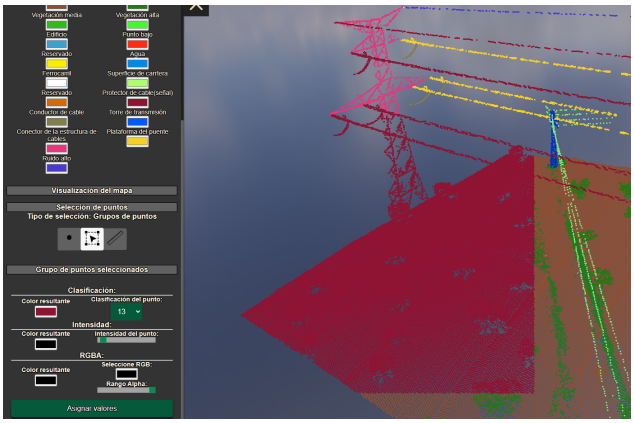

Ilustración 5.18: Cambio de atributos de un grupo de puntos seleccionado.

Capítulo 6

Conclusiones y trabajo futuro

Durante el proceso de desarrollo de este proyecto, se notaron los grandes vacíos de conocimiento en gran parte de las herramientas utilizadas en este proyecto. Que causó grandes retrasos y desmotivación por seguir avanzando, además de errores en parte de la estructura y del uso de algunas de las herramientas utilizadas, pero con gran esfuerzo y con la ayuda del tutor Agustín, se continuó progresando a pasos cortos.

Incluso con las dificultades aquí mostradas, este proyecto puede servir de buena base para seguir expandiendo el proyecto en futuras actualizaciones. Un ejemplo de ello es la implementación de un **Quadtree** que de aún más rendimiento, el cual estaba en proceso, pero que por falta de tiempo no ha sido posible implementar adecuadamente.

Otros apartados a mejorar sería hacer uso de las posibilidades de comunicación entre componentes de React usando esta misma librería. El previo desconocimiento en esta área provocó dio como resultado una implementación un poco más engorrosa pero eficaz.

En todo el proceso del proyecto se han aprendido nuevas herramientas, librerías y lenguajes que solo se habían comentado durante la titulación de Ingeniería Informática, y por fin utilizar dichas herramientas en un entorno real ha sido una de las principales motivaciones de este proyecto, desde el inicio hasta el final. Hay que recalcar que los resultados obtenidos son más que suficientes para lo que se tenía en mente para este proyecto, ya que la idea principal era la de un visor que permitiese visualizar un fichero LAS de extensiones considerables. Esta idea fue evolucionando hasta obtener lo que en esta memoria presentamos.

Bibliografía

[1] Desktop, A. (2021a). Lidar point classification. https://desktop.arcgis.com/en/arcmap/latest/manage-data/las-dataset/lidar-point-classification.htm.

[2] Desktop, A. (2021b). What is lidar intensity data? https://desktop.arcgis.com/en/arcmap/latest/manage-data/las-dataset/what-is-intensity-data-.htm.

[3] Geeksforgeeks (2024). React components. https://www.geeksforgeeks.org/reactjs-components/.

[4] OpenGl. Fragment shader. https://www.khronos.org/opengl/wiki/Fragment_Shader.

[5] OpenGl. Vertex shader. https://www.khronos.org/opengl/wiki/Vertex_Shader.

[6] Schütz, M. (2023). Potree. *https://github.com/potree/potree*.

[7] Schütz, M., Ohrhallinger, S., and Wimmer, M. (2020). Fast out-of-core octree generation for massive point clouds. *https://www.cg.tuwien.ac.at/research/publications/2020/SCHUETZ-2020-MPC/*.

[8] SimonDev (2020). "3d world generation: #3 (quadtree & lod)". https://www.youtube.com/watch?v=YO_A5w_fxRQ.

[Team] Team, G. D. A. dat.gui. https://github.com/dataarts/dat.gui.

[10] Three.Js. Bufferattribute. https://threejs.org/docs/#api/en/core/BufferAttribute.

[11] Three.Js. Buffered geometry. https://threejs.org/docs/index.html#api/en/core/BufferGeometry.

[12] Three.Js. Material. https://threejs.org/docs/index.html#api/en/materials/Material.

[13] Three.Js. Three.vector3. https://threejs.org/docs/#api/en/math/Vector3.distanceTo.

[14] Veesus. Arena4d. https://www.veesus.com/arena4d/.

[15] Veesus. Vpc converter. https://www.veesus.com/vpc-creator/.

[16] Veesus. Vpc viewer. https://www.veesus.com/vpc-viewer/.

[17] Verma, U. and Butler, H. (2014). plasio. *https://github.com/verma/plasio*.

[18] Vivo, P. G. (2015a). The book of shaders. https://thebookofshaders.com/02/.

[19] Vivo, P. G. (2015b). The book of shaders. https://thebookofshaders.com/glossary/?search=smoothstep.

Glosario

ASPRS Sociedad Estadounidense de Fotogrametría y Detección Remota. Sociedad americana dedicada a la fotogrametría y teledetección. 13

Clean Code Filosofía de desarrollo de software que se centra en facilitar la escritura y la lectura del código de una aplicación software.. 15

LiDAR Abreviación de Light Detection and Ranging. Tecnología de medición de terreno mediante rayos disparados desde un dispositivo.. II, III, 1, 2

Octree Estructura de datos que representa un área descompuesta en regiones de ocho cuadrantes iguales, que as su vez están divididas en otras ocho regiones iguales.. 7–10, 20

Out-of-core Algoritmos que utilizan el espacio en disco para guardar información en vez de utilizar la memoria principal, evitando problemas de excesos de memoria.. 10, 22

Quadtree Estructura de datos que representa un área descompuesta en regiones de cuatro cuadrantes iguales, que a su vez están divididas en otras cuatro regiones iguales hasta llegar a un límite impuesto. . 1, 7–9, 17, 18, 20–23

React Librería de JavaScript open source diseñado especialmente para aplicaciones en una única página, alabando a la reactividad de las páginas. 7

Three.js Biblioteca escrita en JavaScript que permite el renderizado de escenas y figuras en 3D en un entorno Web.. 2, 3, 7, 11, 15, 18, 19, 29

Capítulo 7

Anexos

7.1. Manual de aplicación web

Para este manual, puede hacer referencia a la figura 5.13. Aquí se explicará el uso de esta aplicación web.

7.1.1. Iniciar el proyecto

Para iniciar el proyecto, baje el proyecto de GitHub con este link. Luego con un entorno IDE como visual studio code, realice el comando **npm install**, esto le instalará todas las dependencias necesarias para ejecutar el programa.

Luego, realice un **npm start** y se desplegará su aplicación en la dirección http://localhost:3000/.

7.1.2. Importar archivos

Para importar archivos de formato .LAS, haga clic en la zona superior de la interfaz. Donde se abrirá un explorador de ficheros donde seleccionar el archivo necesario. También puede arrastrar el fichero a la parte superior de la interfaz.

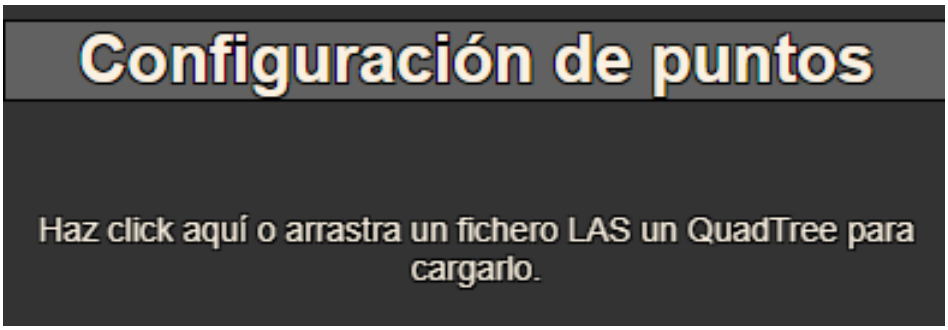

Ilustración 7.1: Zona para arrastrar o elegir el importar un archivo en la interfaz de usuario.

Para introducir las carpetas convertidas en **Quadtree** arrastre la carpeta contenedora de las carpetas con las regiones a la parte superior de la interfaz.

7.1.3. Selección de puntos

Para la selección de puntos en el proyecto, debe seleccionar el botón correspondiente y luego mantener pulsado el botón **shift** y hacer clic en un botón. Esto sirve también para la medición de puntos. Primero seleccione uno de los puntos y luego seleccione otro diferente.

Para la selección en grupo, mantenga pulsado el botón **shift** y haga clic y arrastre el área que quiera seleccionar. Todos los puntos dentro del cuadrado mostrado, serán seleccionados.

7.1.4. Ajustar intensidad de puntos

La intensidad de los puntos puede ser diferente según el tipo de sensor que se haya utilizado, es por ello que se implementó un *slider* para adaptarse a cualquier tipo de sensor.

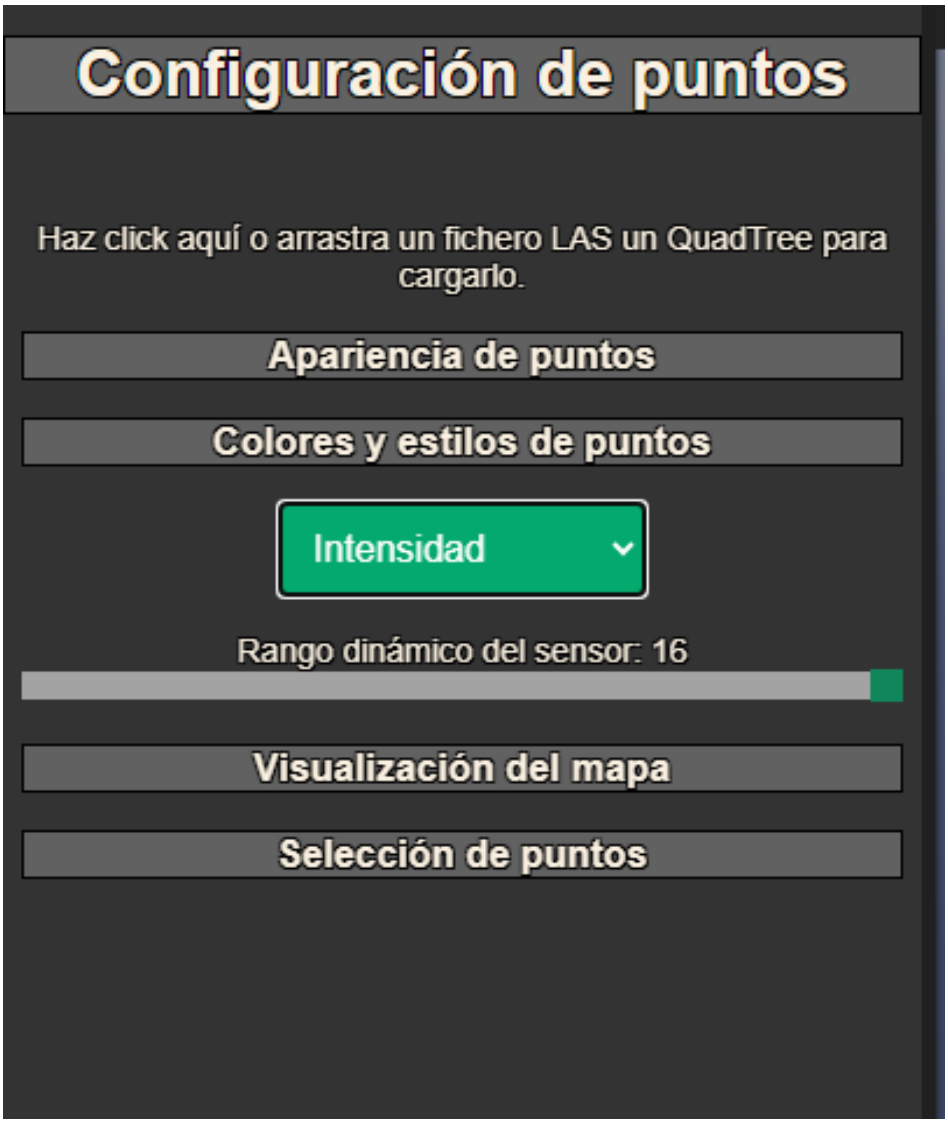

Ilustración 7.2: Modificar el rango del sensor para ajustarse a la intensidad.

Para ello, arrastre el *slider* y seleccione el valor que mejor se ajuste a sus necesidades.

7.1.5. Modificación de colores de clasificación

Para modificar el color de un tipo de clasificación estandarizado, seleccione el estilo de visualización de Clasificación.

Ilustración 7.3: Modificación de colores de clasificación en la interfaz de usuario.

Con esto, podrá ver todos los colores estandarizados y los colores representativos. Si usted da clic al color, podrá editar el color de la clasificación.

7.1.6. Modificación de límites de altura

Para modificar el color de los puntos según la altura, seleccione el estilo de visualización de **Rangos de altura**.

Así podrá ver el color primario y el secundario, estos siendo para valores más bajos y valores más altos. Si quiere modificar estos colores, haga clic en el color y modifique el color.

También se puede modificar los límites de los rangos de altura mediante el uso de los *sliders* respectivos. Si hace que el límite negativo sea más grande que el límite positivo, provocará que los colores se inviertan.

7.1.7. Distancia de renderizado

Para aumentar el límite que indica si la cámara está lo suficientemente cerca como para entrar dentro del **Quadtree** simplemente aumente el valor a través del *slider* correspondiente.

7.1.8. Cambios de color de fondo

Para cambiar el color de fondo de la escena, seleccione uno de los cuatro botones disponibles. Si selecciona el botón de skybox, se cargará un fondo de pantalla representativo de un cielo con nubes.

7.1.9. Perspectivas de cámaras

Esta fila de botones muestra las diferentes perspectivas que se puede asignar a la cámara, siendo estos en orden

- Visión lateral izquierda
- Visión lateral derecha
- Visión vista desde arriba
- Visión vista desde abajo
- Visión vista frontal
- Visión vista trasera

7.2. Manual de aplicación generadora de Quadtree

Este manual de usuario tiene como objetivo guiar al lector sobre cómo hacer uso de esta aplicación generadora de Quadtree. Para iniciar el proyecto, baje el proyecto de GitHub con

este link.

Primero de todo, asegúrese que tiene instalado Python en su máquina. Si no es así, por favor diríjase a este link e instale Python.

7.2.1. Preparación del programa

A continuación, abra un IDE capaz de correr scripts de Python, o bien, inicie una consola de comandos en la carpeta donde quiera generar el Quadtree.

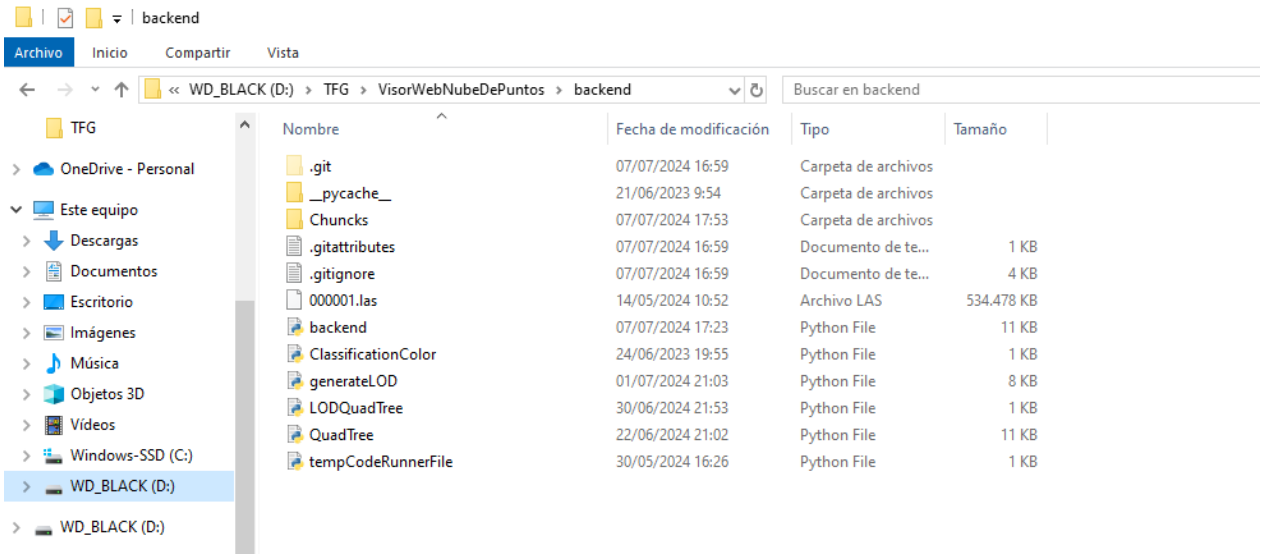

Ilustración 7.4: Primer paso para iniciar la línea de comando

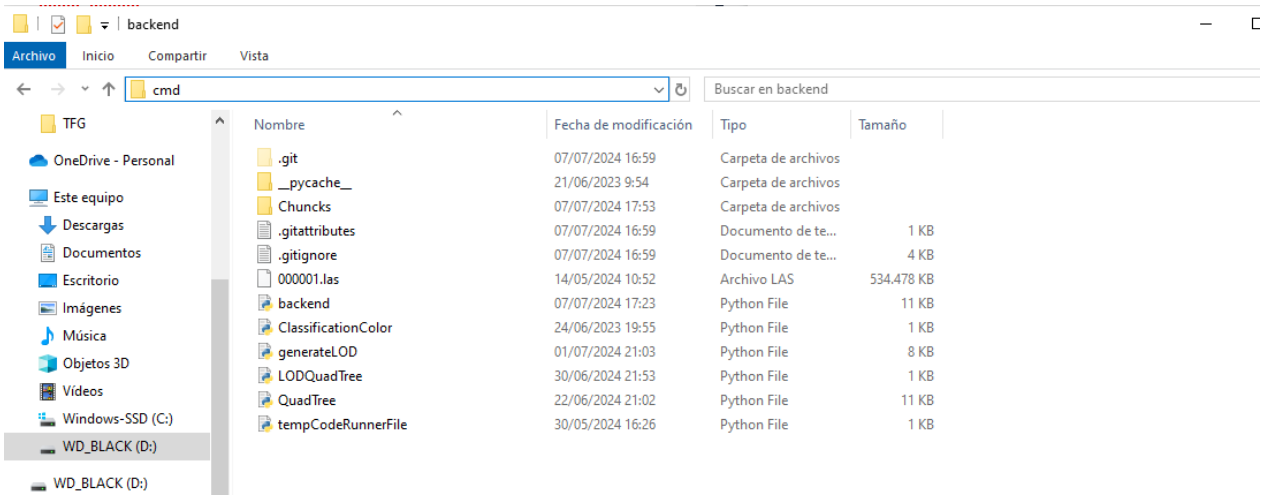

Ilustración 7.5: Segundo paso para iniciar la línea de comando

Ilustración 7.6: Tercer paso para iniciar la línea de comando

Una vez hecho eso, ejecute el script de la siguiente manera:

Python DirecciónDelProyectoDescargado/**backend/backend.py** DirecciónDelArchivoLas/NombreArchivoLas**.las** ./NombreDeLaCarpetaObjetivo

Un ejemplo de ejecución sería el encontrado en la imagen 7.7, en donde me sitúo dentro de la carpeta del proyecto en el que se encuentra el script, y además he añadido el archivo .LAS. Por último, decidí guardar el Quadtree dentro de la carpeta Chuncks/Inicial

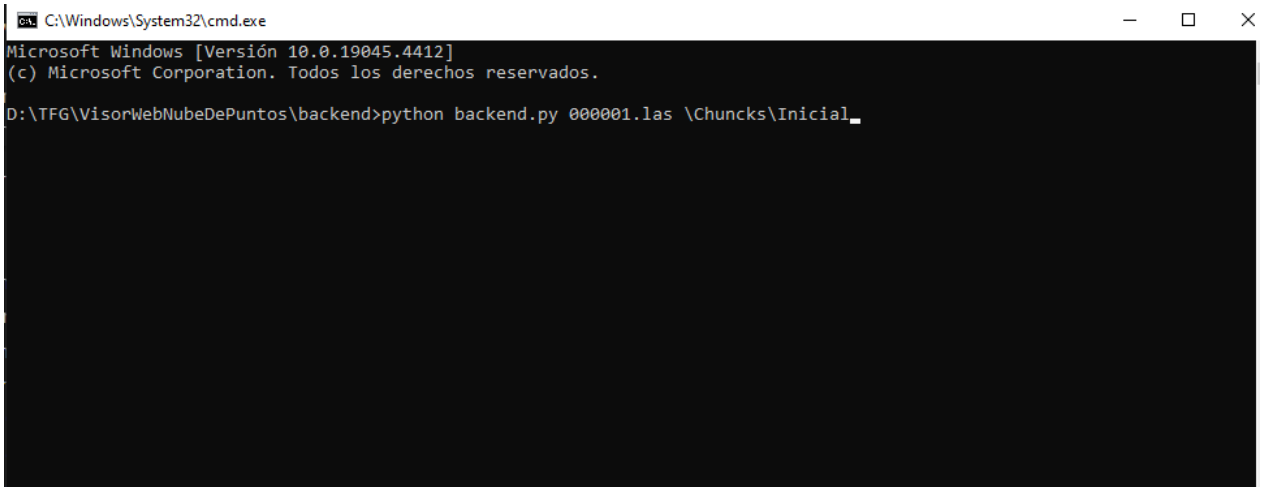

Ilustración 7.7: Ejemplo de inicio de script para la aplicación de Python

El único matiz que debe de tener en cuenta es que el último parámetro se refiere a una carpeta que se encuentre en la raíz donde está usando la consola. Por favor, indique una carpeta tal como aparece en el ejemplo. Si la carpeta no existe a la hora de lanzar el programa, Python se encargará de crearla.