Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

VISUALIZACIÓN PROGRESIVA DE DATASETS LIDAR DE GRAN ESCALA

Alumno

Juan Carlos Casas Rosa

Tutores

Carlos J. Ogayar Anguita
(Departamento de Informática)

Rafael Jesús Segura Sánchez
(Departamento de Informática)

Junio, 2020

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Carlos J. Ogayar Anguita y Don Rafael Jesús Segura Sánchez, tutores del Trabajo Fin de Grado titulado: '**Visualización progresiva de datasets LIDAR de gran escala**', que presenta Don Juan Carlos Casas Rosa, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2020

El alumno:

CASAS ROSA JUAN CARLOS - 26054785V

Firmado digitalmente por CASAS ROSA JUAN CARLOS - 26054785V
Fecha: 2020.06.03 14:06:28 +02'00'

Juan Carlos Casas Rosa

Los tutores:

SEGURA SANCHEZ RAFAEL JESUS - 26003853F

Firmado digitalmente por SEGURA SANCHEZ RAFAEL JESUS - 26003853F
Fecha: 2020.06.02 17:15:50 +02'00'

OGAYAR ANGUITA CARLOS JAVIER - 26039170L

Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26039170L, givenName=CARLOS JAVIER, sn=OGAYAR ANGUITA, cn=OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2020.06.02 17:50:39 +02'00'

Carlos J. Ogayar Anguita

Rafael Jesús Segura Sánchez

(Página intencionalmente en blanco)

Agradecimientos

Muchas gracias primeramente a mis directores del trabajo y lo que me dieron la oportunidad de realizarlo en el GGGJ; una experiencia muy enriquecedora. Muchas gracias, Carlos y Rafael, estos 8 meses de proyecto junto a vosotros me han encantado.

También me gustaría acordarme de mis amigos de “La Mafia”, todos mis compañeros de universidad y por supuesto de “Los Canteros”; gracias por soportarme diariamente y por compartir vuestro día a día conmigo.

Quiero agradecer también a mi familia su apoyo diario, especialmente en aquellos días en los que parecía que nada iba bien. Hoy día comprendo que es lo más normal del mundo que existan momentos así, especialmente en retos complicados. Los objetivos solo se cumplen a base de trabajo y constancia.

Finalmente, me gustaría acordarme de la persona más influyente de mi vida, mi abuelo Manuel, que, si bien me acompañó físicamente los 3 primeros años de este camino, estuvo conmigo de otra manera en este 4º curso. No te cansaste de repetírmelo una y otra vez, y ahora puedo decir: “Soy ingeniero, ¿no comprendes?”.

Muchas gracias de todo corazón a cada una de las personas que me han acompañado en este reto.

Sueño conseguido, quedan 99.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Juan Carlos Casas Rosa
Fecha de asignación	23/03/2020
Descripción corta	Este trabajo consiste en la realización de un estudio teórico-práctico sobre las técnicas a aplicar para conseguir la visualización progresiva de grandes conjuntos de datos procedentes de escaneado 3D LiDAR. Este tipo de escaneado a gran escala es muy utilizado en ámbitos como la Ingeniería Civil, la Arquitectura (BIM), las ciudades inteligentes, la gestión del territorio (topografía y cartografía) o la arqueología. Uno de los grandes problemas que se presentan con este tipo de datos es la incapacidad de la mayoría de los sistemas para almacenar todos los datos en memoria principal, así como el insuficiente rendimiento a la hora de visualizar y monitorizar conjuntos de datos a gran escala. Con este trabajo se plantea la elaboración de un prototipo que permita visualizar de forma progresiva y adaptativa grandes conjuntos de datos LiDAR, para lo cual será necesario realizar un estudio teórico-práctico de las técnicas disponibles en la actualidad para conseguir dicho objetivo.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

1 Especificación del trabajo 13

1.1 Introducción 13
1.2 Objetivos del trabajo 15
1.3 Antecedentes y estado del arte 15
1.3.1 Nubes de puntos 15
1.3.2 Tecnología LiDAR 16
1.3.3 Tratamiento masivo de datos 18
1.3.4 Formatos de fichero de almacenamiento 19
1.3.5 Plataformas y herramientas de visualización 21
1.3.6 Procesamiento en dos etapas y visualización progresiva 23
1.3.7 Técnicas representativas 26
1.3.8 Operaciones comunes realizadas a nubes de puntos LiDAR 31
1.4 Descripción de la situación de partida 33
1.4.1 Descripción del entorno actual 33
1.4.2 Resumen de las deficiencias y carencias identificadas 34
1.5 Requisitos iniciales 34
1.6 Alcance 35
1.7 Hipótesis y restricciones 35
1.8 Estudio de alternativas y viabilidad 35
1.9 Descripción de la solución propuesta 38
1.10 Material y métodos 39
1.11 Tecnologías utilizadas 40
1.12 Metodología de desarrollo de software 41
1.13 Estimación del tamaño y esfuerzo 41
1.14 Planificación temporal 41
1.15 Presupuesto 46

2 Desarrollo 49

2.1 Primera Iteración 49
2.1.1 Decisiones de diseño 49
2.1.2 Detalles de implementación 50
2.1.3 Pruebas realizadas 50
2.2 Segunda iteración 51
2.2.1 Decisiones de diseño 51
2.2.2 Detalles de implementación 51
2.2.3 Pruebas realizadas 53
2.3 Tercera iteración 53
2.3.1 Decisiones de diseño 53
2.3.2 Detalles de implementación 55
2.3.3 Pruebas realizadas 56
2.4 Cuarta iteración 56
2.4.1 Decisiones de diseño 56
2.4.2 Detalles de implementación 58
2.4.3 Pruebas realizadas 60
2.5 Quinta iteración 60
2.5.1 Decisiones de diseño 60
2.5.2 Detalles de implementación 61

2.5.3 Pruebas realizadas 63
2.6 Sexta iteración 64
2.6.1 Decisiones de diseño 64
2.6.2 Detalles de implementación 65
2.6.3 Pruebas realizadas 67
2.7 Séptima iteración 67
2.7.1 Decisiones de diseño 67
2.7.2 Detalles de implementación 70
2.7.3 Pruebas realizadas 72
2.8 Octava iteración 73
2.8.1 Decisiones de diseño 73
2.8.2 Detalles de implementación 75
2.8.3 Pruebas realizadas 75
2.9 Novena iteración 76
2.9.1 Decisiones de diseño 76
2.9.2 Detalles de implementación 79
2.9.3 Pruebas realizadas 80
2.10 Décima iteración 80
2.10.1 Decisiones de diseño 80
2.10.2 Detalles de implementación 83
2.10.3 Pruebas realizadas 83
2.11 Undécima iteración 84
2.11.1 Decisiones de diseño 84
2.11.2 Detalles de implementación 84
2.11.3 Pruebas realizadas 85
2.12 Pruebas finales 85

3 Experimentación, resultados y discusión 86

3.1 Experimentaciones y pruebas 86
3.2 Resultados y discusión 99

4 Conclusiones y trabajos futuros 102

5 Apéndices 104

5.1 Guía original del Trabajo Fin de Título 104

6 Definiciones y abreviaturas 107

7 Bibliografía 109

Índice de ilustraciones

Ilustración 1. Ejemplo de nube de puntos 16
Ilustración 2. Esquema conceptual de escaneo LiDAR....17
Ilustración 3. Esquema del procesamiento en dos etapas del visualizador web ViLMA 24
Ilustración 4. Subdivisión de nubes de puntos en la fase de procesamiento del artículo Supporting multi-resolution out of core rendering of massive LiDAR points clouds through non-redundant data structures....26
Ilustración 5. Representación de un árbol kd tridimensional 27
Ilustración 6. Ejemplo de visualización de una nube de puntos con el software Potree 28
Ilustración 7. Subdivisión de un modelo complejo en un octree....29
Ilustración 8. Nube de puntos esférica particionada en un Modifiable Nested Octree (MNO) 30
Ilustración 9. Web donde se encuentra la documentación de la biblioteca VTK....39
Ilustración 10. Planificación temporal original del proyecto....43
Ilustración 11. Pipeline de visualización de VTK....52
Ilustración 12. Estructura de datos espacial de grid disperso 54
Ilustración 13. Subdivisión de una nube de puntos en niveles de detalle....58
Ilustración 14. Algoritmo de subdivisión de una nube de puntos en niveles de detalle disjuntos 59
Ilustración 15. Estructura interna del Sparse Grid....61
Ilustración 16. Esquema de interacción entre los hilos visualizador y gestor de peticiones...69
Ilustración 17. Ejemplo de caja envolvente de un objeto tridimensional....74
Ilustración 18. Esquema del patrón de diseño Modelo-vista-controlador 76
Ilustración 19. Ejemplo del funcionamiento de la estructura de actores....78
Ilustración 20. Ejemplo de obtención del vecindario de un nodo....82
Ilustración 21. Tiempo usado en la realización de cada experimento 99
Ilustración 22. Tasa de refresco en función del presupuesto de puntos en cada experimento 100
Ilustración 23. Tasa de refresco en función del tamaño de celda 101
Ilustración 24. Presentación del TFG y conocimientos previos 104
Ilustración 25. Objetivos del TFG y metodología del mismo 105
Ilustración 26. Documentos y formatos de entrega del TFG 106

Índice de tablas

Tabla 1. Formato de fichero LAS 19
Tabla 2. Formato de los registros de los datos de los puntos 20
Tabla 3. Comparación entre formatos LAS y LAZ 21
Tabla 4. Comparación de lenguajes de programación considerados para el proyecto 36
Tabla 5. Comparación de estructuras de datos consideradas para el proyecto 37
Tabla 6. Comparación de herramientas de visualización consideradas para el proyecto 38
Tabla 7. Planificación temporal final del proyecto 45
Tabla 8. Gastos del material hardware del proyecto 47
Tabla 9. Gastos en personal del proyecto 48
Tabla 10. Gastos totales del proyecto 48
Tabla 11. Pseudocódigo de los hilos visualizador y gestor de peticiones 71
Tabla 12. Funcionalidades de los elementos del proyecto 73
Tabla 13. Parámetros usados en la experimentación 87
Tabla 14. Definición de los experimentos a realizar 89
Tabla 15. Resultados obtenidos del experimento 1 90
Tabla 16. Resultados obtenidos del experimento 2 90
Tabla 17. Resultados obtenidos del experimento 3 90
Tabla 18. Resultados obtenidos del experimento 4 90
Tabla 19. Resultados obtenidos del experimento 5 91
Tabla 20. Resultados obtenidos del experimento 6 91
Tabla 21. Resultados obtenidos del experimento 7 91
Tabla 22. Resultados obtenidos del experimento 8 91
Tabla 23. Resultados obtenidos del experimento 9 92
Tabla 24. Resultados obtenidos del experimento 10 92
Tabla 25. Resultados obtenidos del experimento 11 92
Tabla 26. Resultados obtenidos del experimento 12 92
Tabla 27. Resultados obtenidos del experimento 13 93
Tabla 28. Resultados obtenidos del experimento 14 93
Tabla 29. Resultados obtenidos del experimento 15 93
Tabla 30. Resultados obtenidos del experimento 16 93
Tabla 31. Resultados obtenidos del experimento 17 94
Tabla 32. Resultados obtenidos del experimento 18 94
Tabla 33. Resultados obtenidos del experimento 19 94
Tabla 34. Resultados obtenidos del experimento 20 94
Tabla 35. Resultados obtenidos del experimento 21 95
Tabla 36. Resultados obtenidos del experimento 22 95
Tabla 37. Resultados obtenidos del experimento 23 95
Tabla 38. Resultados obtenidos del experimento 24 95
Tabla 39. Resultados obtenidos del experimento 25 96
Tabla 40. Resultados obtenidos del experimento 26 96
Tabla 41. Resultados obtenidos del experimento 27 96
Tabla 42. Resultados obtenidos del experimento 28 96
Tabla 43. Resultados obtenidos del experimento 29 97
Tabla 44. Resultados obtenidos del experimento 30 97

Tabla 45. Resultados obtenidos del experimento 31 97
Tabla 46. Resultados obtenidos del experimento 32 97
Tabla 47. Resultados obtenidos del experimento 33 98
Tabla 48. Resultados obtenidos del experimento 34 98
Tabla 49. Resultados obtenidos del experimento 35 98
Tabla 50. Resultados obtenidos del experimento 36 98

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

En este trabajo de fin de grado (TFG), realizado primordialmente en el grupo de informática gráfica y geomática de la universidad de Jaén (GGGJ), hemos realizado **un estudio teórico-práctico relativo a las técnicas a aplicar para conseguir la visualización progresiva de grandes conjuntos de datos, en forma de nubes de puntos, procedentes de escaneos 3D LiDAR (Light Detection And Ranging)**.

Este campo de investigación es ampliamente estudiado, tanto en esta institución como en diferentes universidades o centros de investigación nacionales e internacionales; con diversas aplicaciones en todo tipo de ámbitos relacionados con la ciencia de computadores.

La naturaleza de esta técnica implica la **gestión de ingentes cantidades de información**, dificultando así la visualización de los puntos debido al límite de memoria principal de los equipos. A esta problemática, se le une además **el deficiente rendimiento** cuando nos disponemos a visualizar conjuntos de datos a gran escala.

De aquí en adelante, propondremos **una solución basada en un sistema de visualización progresiva, con diferentes fases de procesamiento y que es capaz de resolver la carga de datos bajo demanda a partir de un sistema productor-consumidor multitarea**. Las diferentes decisiones de diseño concebidas en este proyecto se fundamentan en las diferentes publicaciones, avances y estudios descritos en el epígrafe relativo al estado del arte de este proyecto y serán concebidas en el marco de una metodología de desarrollo ágil.

De acuerdo al índice previamente descrito, la estructura general de este documento es la siguiente:

- Este primer apartado estará centrado es describir la **base del proyecto**, realizando un análisis de la situación de partida y revisando los diferentes estudios en este campo de investigación relativos a los objetivos de este proyecto.
- La segunda parte concibe los **detalles de cada fase del desarrollo**, organizadas en dos partes; diferenciando así las decisiones de diseño tomadas de su correspondiente implementación. Todo lo referente al desarrollo de programación, como errores, problemas y sus soluciones, aparece reflejado en esta sección.
- El tercer apartado de este proyecto recoge los **resultados obtenidos** después de poner a prueba el prototipo desarrollado con diferentes conjuntos de datos y un análisis exhaustivo del rendimiento final, otorgando así una visión en perspectiva de los resultados del proyecto.
- En el cuarto epígrafe del documento se discuten y se ponen sobre la mesa las **conclusiones obtenidas después del desarrollo del TFG**; de tal modo se proponen posibles ampliaciones del mismo de cara a nuevos proyectos con el mismo enfoque o incluso con base fundamentada en el mismo software. También se describe la posibilidad de redactar un artículo científico relativo a este proyecto.
- En la quinta, sexta y séptima parte se describen finalmente diferentes **apéndices** necesarios para comprender los objetivos del proyecto, algunos términos del trabajo y finalmente la bibliografía consultada en este proyecto, con especial utilidad.

1.2 Objetivos del trabajo

De acuerdo al punto de partida definido anteriormente, los objetivos que abarcamos en este proyecto son los detallados a continuación:

I. **Realizar un estudio teórico-práctico** sobre métodos de visualización de nubes de puntos, así como estructuras de datos y algoritmos orientados al tratamiento masivo de puntos 3D.

II. **Diseñar e implementar un prototipo de aplicación** que permita el procesamiento de grandes conjuntos de datos LiDAR en tiempo real. Estudiar una posible estructura de datos multiresolución orientada a la carga progresiva de datos bajo demanda.

III. **Plantear un sistema productor-consumidor multitarea** para resolver la carga de datos bajo demanda del sistema de visualización de datos.

IV. Realizar una **fase de experimentación** que permita comprobar la calidad de los resultados, así como el rendimiento obtenido, mediante el uso de datasets reales de gran escala usados en la práctica.

V. **Redactar las conclusiones del estudio** destacando los aspectos más interesantes sobre las estructuras de datos y los métodos considerados, la implementación realizada y los resultados obtenidos.

1.3 Antecedentes y estado del arte

En este apartado se describirán conceptos determinantes para la comprensión de este trabajo y los elementos que han tenido relevancia en este estudio procedentes de la investigación de este campo de estudio.

1.3.1 Nubes de puntos

Una nube de puntos es un conjunto de vértices en un espacio tridimensional. Estos vértices se identifican mediante **ternas (X, Y, Z)** que determinan la posición de cada uno de esos puntos en el espacio y que en su conjunto representan la superficie externa de un objeto.

Ilustración 1. Ejemplo de nube de puntos

Estas nubes de puntos son generadas frecuentemente mediante escáneres tridimensionales; instrumentos que toman medidas en forma de puntos de los objetos o terrenos a estudiar y generan automáticamente una serie de ficheros de datos con el resultado del escaneo, en un formato determinado, y que será descrito en un posterior apartado de este epígrafe. En la ilustración 1 podemos ver un ejemplo de nube de puntos de una zona industrial.

1.3.2 Tecnología LiDAR

La tecnología **LiDAR** (acrónimo del inglés Light Detection And Ranging, es decir, detección de luz y distancia) es un **sistema láser que permite medir distancias desde el origen del láser hasta superficies u objetos a estudiar**. El tiempo de retardo entre la salida del láser del dispositivo de escaneo hasta su retorno permite calcular la distancia hacia una determinada parte de la superficie del elemento; lo que nos permite obtener un mapa tridimensional de alta resolución del mismo.

No solamente se analiza la distancia del escaneo, si bien se recogen diferentes medidas como el ángulo y la dirección con las que se realizó el escaneo, información geoposicional, intensidad de la devolución del pulso láser, momento temporal de la medición, color, etc; **si bien en este proyecto nosotros solamente consideraremos los atributos de posición y de color de cada punto**.

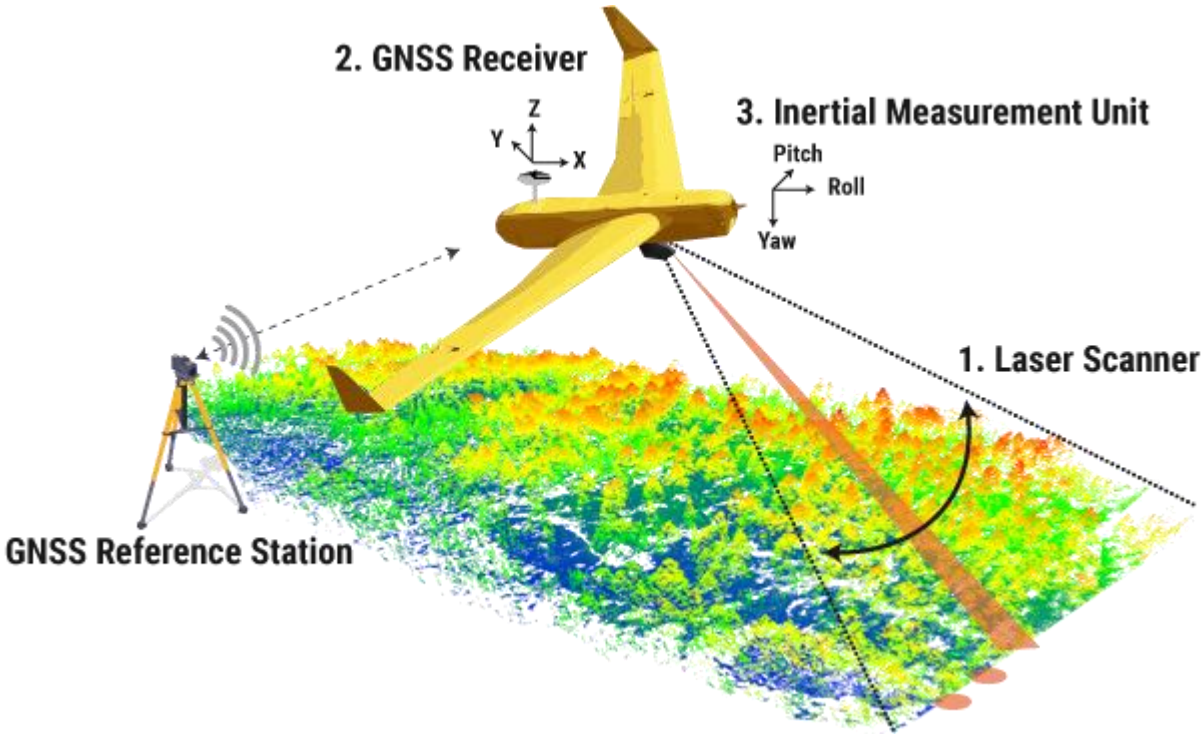

Ilustración 2. Esquema conceptual de escaneo LiDAR

Si bien el proceso de generación de datos LiDAR es arduo complejo (como se puede apreciar en la ilustración 2) y requiere un mayor estudio, en este trabajo nos centraremos en la parte posterior de procesamiento de los puntos y avances científicos.

La tecnología LiDAR proporcionar datos de alta resolución extremadamente útiles en forma de nubes de puntos que pueden ser aplicadas en un gran conjunto de campos como agricultura, arqueología, biología, geología o ciencias forestales. Aplicaciones de esta tecnología en el ámbito de ciencias de la tierra abarcan estudios sobre cambios en la costa (Sallenger et al. 1999), extracción de características geomorfológicas (Passalacqua et al. 2010), creación de detallados modelos de ciudades a gran escala (Lafarge and Mallet 2010), análisis de procesos de desplazamiento de terrenos (Ventura et al. 2011), volcanes (Kereszturi et al. 2012) y tectónicas activas (Arrowsmith and Zielke 2009, Brunori et al. 2013), entre otras.

El uso de esta tecnología ha experimentado un notable crecimiento en muchos sectores científicos recientemente con el crecimiento de vehículos aéreos no tripulados como drones.

1.3.3 Tratamiento masivo de datos

Hoy en día **la ingente cantidad de información espacial que es adquirida mediante la tecnología LiDAR supone un reto** al desarrollar aplicaciones enfocadas en el manejo de tal cantidad de datos.

Podemos distinguir varias problemáticas cuando queremos desarrollar tales aplicaciones debido al gran volumen de información con la que se trabaja en este campo, en diferentes ámbitos:

1. En los servidores o centros de procesamientos de datos, la información LiDAR ocupa una gran cantidad de espacio ya que se almacenan datos procedentes de múltiples escaneos, que pueden llegar a incluso a varios billones de puntos por elemento, en el caso de superficies de terreno extensas y con gran nivel de detalle. En esta etapa se almacenan los datos tal y como provienen de los escaneos, crudos, por lo que la cantidad de recursos que se destinan al almacenamiento debe ser mucho mayor que en otros ámbitos; sin embargo, no precisaremos de potencia de visualización en esta fase.

2. En los clientes de sobremesa, serán los equipos en los que se desarrollan aplicaciones. A la hora de trabajar en el desarrollo de software con nubes de puntos se necesita en primera instancia capacidad de almacenamiento suficiente para albergar los datasets de escaneos LiDAR objetos de estudio y, también, potencia de visualización suficiente como para ofrecer una interacción con el programa transparente y fluida. Es importante, por lo tanto, encontrar un compromiso entre la memoria de estos equipos (tanto estática como volátil) y la potencia de cálculo.

3. En los clientes móviles, habitualmente con muchos menos recursos que los clientes de sobremesa, encontramos la problemática fundamental de sus recursos restringidos. Tanto la memoria como la potencia de los dispositivos móviles suele ser bastante menor que la de los PCs. Si bien no se desarrollan aplicaciones explícitamente sobre estos equipos, a la hora de trabajar en un software de nubes de puntos sobre este tipo de equipos se deben tener en cuenta las restricciones intrínsecas de memoria y de capacidad de procesamiento.

1.3.4 Formatos de fichero de almacenamiento

El escaneo por parte de los dispositivos que implementan la tecnología LiDAR generan un conjunto de datos y metadatos que deben ser almacenados en un formato de fichero específico, genérico y eficiente.

Para conseguir ese objetivo, y en busca de una estandarización internacional, la American Society for Photogrammetry and Remote Sensing (ASPRS) especificó el **formato LAS**, acrónimo de LASer, en **mayo de 2009**. Este formato de fichero es abierto, binario y diseñado específicamente para nubes de puntos.

La estructura de los ficheros LAS es la descrita en la Tabla 1.

Sección	Descripción
Cabecera pública	Describe el formato, número de puntos, extensión de la nube de puntos y otros datos genéricos
Extensión variable de registros (VLR)	Cualquier número de registros opcionales para proporcionar datos como el sistema de referencia espacial usado, metadatos, etc. El límite de datos es de 65.535 bytes.
Registros de puntos	Datos de cada punto de la nube: coordenadas, color, clasificación, ángulo de escaneo, intensidad, etc.
Registros de extensión variable de registros (EVLR)	Campo similar a la extensión variable de registros, pero después de los registros de los puntos y que permite almacenar muchos más datos que el VLR.

Tabla 1. Formato de fichero LAS

Cada uno de estos apartados descritos tienen su correspondiente descripción de sus componentes en la especificación del formato LAS. La descripción del campo más importante, el que contiene los datos sobre los puntos, aparece recogida tal y como viene en la documentación del estándar en la tabla 2.

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0, 1, 2)	3 bits	*
Number of Returns (given pulse)	3 bits (bits 3, 4, 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bit	*
Edge of Flight Line	1 bit (bit 7)	1 bit	*
Classification	unsigned char	1 byte	*
Scan Angle Rank (-90 to +90) – Left side	unsigned char	1 byte	*
User Data	unsigned char	1 byte	
Point Source ID	unsigned short	2 bytes	*
GPS Time	double	8 bytes	*
Red	unsigned short	2 bytes	*
Green	unsigned short	2 bytes	*
Blue	unsigned short	2 bytes	*

Tabla 2. Formato de los registros de los datos de los puntos

En la tabla 2 aparecen los diferentes componentes que se almacenan de cada medida, y en este proyecto **nosotros trabajaremos tanto con las ternas (X, Y, Z) como la de colores (Red, Green, Blue).**

El formato LAS se ha convertido en un estándar de facto en el almacenamiento y distribución de los puntos adquiridos mediante escaneos LiDAR. Debido a que la tecnología LiDAR escala, con el objetivo de proveer escaneos más densos y con mayor resolución, de tal manera los ficheros generados también aumentan su tamaño y su complejidad de lectura. A largo plazo esto puede ser una problemática especialmente en el caso de un futuro próximo en que los ficheros con billones de puntos estén a la orden del día. Para remediar esa problemática, Martin Isenburg publica en 2011 una alternativa, en su paper LASzip: lossled compression of LiDAR data describe **el formato LAZ, un formato de compresión sin pérdidas del LAS que mejora tanto en tamaño de ficheros generados como en velocidad de decodificación a su antecesor**. En la tabla 3 podemos ver una comparación entre ambos formatos, aplicándose a los mismos datasets y analizando la ratio de compresión.

LiDAR of Minnesota DNR by county			size [GB]		compression
name	# of files	# of points	LAS	LAZ	ratio
cottonwood	216	2,491,327,766	65.0	5.2	12.6
douglas	252	2,092,702,039	54.6	5.4	10.2
freeborn	260	1,713,544,294	44.7	3.7	12.1
houston	197	1,450,109,156	37.8	3.9	9.8
jackson	240	2,724,531,642	71.0	5.6	12.6
lincoln	195	2,200,533,847	57.4	4.5	12.8
martin	252	2,853,353,232	74.4	5.8	12.9
murray	252	2,872,608,269	74.9	5.8	12.9
pope	247	2,404,624,049	62.7	5.3	11.9
redwood	302	3,505,060,711	91.4	7.3	12.4
sibley	219	2,501,934,963	65.2	5.8	11.2
swift	282	2,931,687,204	76.4	5.8	13.2
total	**2,914**	**29,742,017,172**	**776**	**64**	**12.1**

Tabla 3. Comparación entre formatos LAS y LAZ

De acuerdo a la tabla 3, podemos ver como un fichero LAZ ocupa aproximadamente 12 veces menos que uno LAS. Además, en el artículo, se demuestra como la velocidad de decodificación de un fichero LAZ varía en el rango de 1 a 3 millones de puntos por segundo, una velocidad muy alta y adecuada para grandes conjuntos de datos.

En este proyecto se trabaja tanto con ficheros en formato LAS como en formato LAZ, aunque los ficheros generados por el propio software sean del último tipo mencionado debido a su mejor compresión y velocidad, descritas en este epígrafe.

1.3.5 Plataformas y herramientas de visualización

En el campo de estudio de las nubes de puntos se han propuesto trabajos sobre diferentes aspectos de las mismas con variadas herramientas y plataformas.

Para la realización de este trabajo se optó por la herramienta VTK (Visualization ToolKit), un sistema de software libre para la realización de gráficos 3D, procesamiento de imágenes y visualización de las mismas. VTK soporta los algoritmos de visualización más conocidos, permite la integración con la herramienta de construcción de interfaces de usuario Qt, es multiplataforma y ha sido usado con resultados satisfactorios por otros miembros del grupo de investigación en el que se realizó este trabajo.

Si revisamos la literatura científica relativa a las nubes de puntos no encontraremos un consenso sobre qué plataforma o herramientas usar. En los siguientes puntos se detallan implementaciones con diferentes plataformas y herramientas relativas a trabajos de visualización de nubes de puntos a gran escala.

- **WebGL**: Este estándar de renderización de gráficos dentro de cualquier navegador web es explotado por la mayoría de aplicaciones que precisan de visualizar nubes de puntos en un browser. Esta API, implementada en JavaScript, es usada por ejemplo en la implementación del framework GVLiDAR; un visualizador basado en la web e interactivo. Esta herramienta fue desarrollada por D. Deibe, M. Amor, R. Doallo, D. Miranda y M. Cordero; y es descrita en un paper de 2017: GVLiDAR: An Interactive Web-based Visualization Framework to Support Geospatial Measures on LiDAR Data. Los objetivos de esta plataforma se centran en conseguir un rendimiento acorde a la interacción en tiempo real, disponer de funcionalidades y en asegurar una disponibilidad completa para los datasets LiDAR.

- **Unity**: Uno de los más conocidos motores de videojuegos multiplataforma también tiene aplicaciones en el ámbito de visualización de nubes de puntos. Su explotación la podemos encontrar en la tesis de final de grado de S. Maximilian Fraiss de 2017, titulada: Rendering Large Point Clouds in Unity. Esta tesis contiene un visualizador de grandes nubes de puntos, varios millones o incluso billones, desarrollado en este motor y que no se pueden cargar completamente en memoria. Para ello, en este trabajo se describen estructuras de datos especiales y algoritmos que faciliten la visualización progresiva de las partes de la nube de puntos que son relevantes en función de la posición de la cámara. El resultado obtenido es un sistema con diferentes técnicas de renderización, eficiente y ampliamente parametrizable.

- **Unreal Engine**: Este motor de videojuegos se considera, a la par que Unity, uno de los más usados en el ámbito de desarrollo de videojuegos para ordenadores o dispositivos móviles. De acuerdo a la tesis de final de máster de Valentin Kraft de 2017 denominada Efficient rendering of

massive and Dynamic point cloud data in state-of-the-art graphics engines. En este trabajo el autor define una alternativa eficiente de implementación de renderización de nubes de puntos basadas en la GPU que es capaz de renderizar grandes nubes de puntos con gran calidad y en tiempo real. También se describe un sistema de paralelización masiva para el ordenamiento de nubes de puntos mediante shaders en tiempo de ejecución.

- **Cloud Compare**: Cabe reseñar la especial importancia que tiene en el ámbito de procesamiento de nubes de puntos el software CloudCompare. Este proyecto de código abierto contiene un programa de visualización de nubes de puntos y de operaciones con ellas multiplataforma y fácilmente escalable mediante plugins. Este software ha sido usado en la realización de este proyecto a la hora de hacer pruebas con posibles ficheros defectuosos y en la redacción de este documento para infografía ya que es muy sencillo de usar, es muy rápido, permite interacción y presenta una gran cantidad de opciones. Está desarrollado en C++, con la integración de la herramienta Qt y su licencia de uso es GPL.

1.3.6 Procesamiento en dos etapas y visualización progresiva

En este proyecto se divide en dos etapas claramente diferencias y con 2 objetivos totalmente distintos, sin entrar en mucho nivel de detalle, para conseguir un sistema de visualización progresiva se necesita diferenciar una etapa de preprocesamiento de los ficheros originales de una posterior fase de visualización dinámica de las nubes de puntos anteriormente divididas.

Esta idea, de procesamiento diferido y en tiempo de ejecución, recala primordialmente en el estudio de 2019: Supporting multi-resolution out of core rendering of massive LiDAR points clouds through non-redundant data structures, de D. Deibe, M. Amor y R. Doallo. En este trabajo, sus autores implementan un sistema de visualización progresiva de grandes nubes de puntos, mediante la implementación de diferentes estructuras de datos y en el entorno de una arquitectura cliente-servidor en la que visualizar los datos en un navegador.

Cabe remarcar el detalle de que estas estructuras y el consecuente procesamiento en dos etapas se implementan en un visualizador web implementado por los autores y denominado ViLMA. En la ilustración 3, extraída del artículo que acaba de ser mencionado, se explican estas dos fases de procesamiento y que en este proyecto también se realizan, con una adaptación a nuestras necesidades y objetivos.

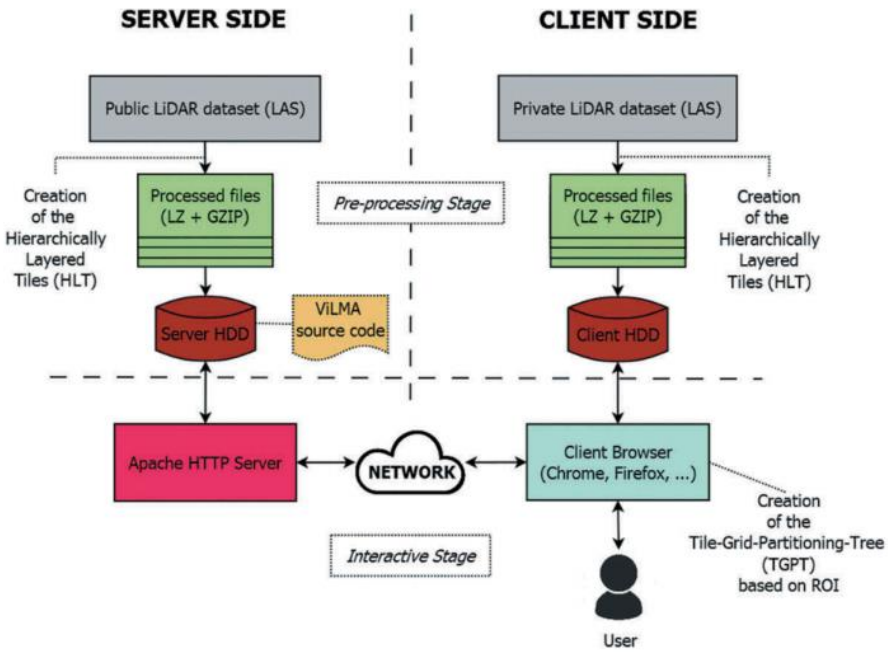

Ilustración 3. Esquema del procesamiento en dos etapas del visualizador web ViLMA

En el artículo que estamos mencionando, se disponen de dos datasets, público y privado, tanto en cliente como en servidor y se crea una estructura jerárquica de puntos en memoria masiva. Esta fase de preprocesamiento también será implementada en este proyecto, generando en disco una estructura de datos explícita agrupada en directorios. Posteriormente, ViLMA genera una estructura de datos dinámica en tiempo de ejecución en el lado del cliente que permite hacer peticiones concretas de nodos en regiones de interés (ROI: Region Of Interest) y así poder hacer posible la representación de las nubes de puntos adecuadas en función de posición del visor. **Este esquema de estructura de datos dinámica también es adaptado en este proyecto con el objetivo de incrementar la eficiencia del sistema, evitando duplicar peticiones y favoreciendo así al rendimiento de la aplicación**.

Con respecto a la visualización progresiva, **para asegurar un rendimiento adecuado para la interacción en tiempo real, es necesario subdividir una nube de puntos en diferentes componentes**. La subdivisión de nubes de puntos **no es trivial**, ya que podemos sobrecargar el almacenamiento si duplicamos los puntos. Para ello, la solución que nosotros proponemos, y que es interpretada también en este artículo, es que los componentes de una nube de puntos mayor sean todos **disjuntos**. Esto implica que una nube de puntos se subdivide en n componentes que no comparten ningún punto entre sí, pero donde la unión de todos los componentes conforma la nube de puntos original. **La unión de diferentes componentes nos permite emular diferentes niveles de detalle de la nube y aumentar la precisión**.

En este proyecto, la división se realiza en tantos ficheros como se indique, cada uno de los cuales contiene **un porcentaje de puntos de la nube de puntos original** como se indique, y donde los puntos se distribuyen en cada fichero de acuerdo a una función de distribución normal. En el artículo que estamos mencionando en este epígrafe, la subdivisión de los puntos en diferentes ficheros es explicada como aparece en la ilustración 4. En este caso, en disco se almacenan todos los puntos de forma disjunta de tal modo que la unión de ellos permite reproducir nuevos niveles de detalle en memoria principal, que junto a la nueva unión de demás ficheros que residían en disco permitan incrementar el nivel de detalle sin necesidad de duplicar información.

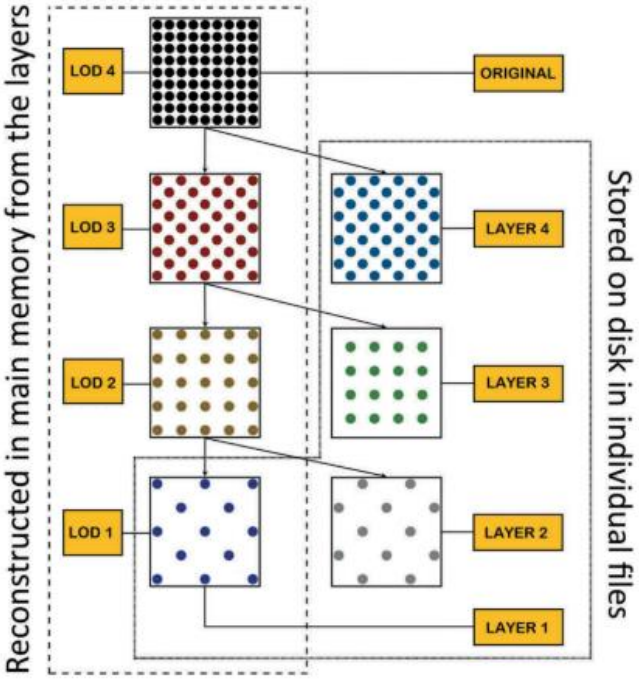

Ilustración 4. Subdivisión de nubes de puntos en la fase de procesamiento del artículo Supporting multi-resolution out of core rendering of massive LiDAR points clouds through non-redundant data structures

1.3.7 Técnicas representativas

En esta sección se mencionarán diferentes técnicas de gran impacto en el estado del arte relacionadas con los trabajos que anteriormente han sido funcionados y utilizados como recursos.

Dentro de cada trabajo, una serie de herramientas específicas han sido abordadas desde la perspectiva particular de cada proyecto y con el objetivo de lograr un resultado final novedoso, eficiente y de calidad. Las diferentes técnicas que podemos mencionar son las siguientes.

En la tesis de final de máster de Valentin Kraft de 2017 **denominada Efficient rendering of massive and Dynamic point cloud data in state-of-the-art graphics engines** se mencionan diferentes estructuras de datos jerárquicas basadas en árboles como base del planteamiento de este problema como kdtrees, mencionando el trabajo “Multidimensional binary search trees used for associative searching”. Communications of the ACM 18.9 (1975) de Jon Louis Bentley. y basadas en octrees como “Geometric modeling using octree encoding”. Computer Graphics and Image Processing 19.2 (1982) de Donald Meagher.

Después de considerar las diferentes estructuras y posibilidades, de acuerdo a la implementación concreta que estaba realizando con Unreal Engine, Kraft decidió que la estructura de datos espacial debería ser jerárquica y con estructura arbórea, optando por un **árbol KD o KD-tree**.

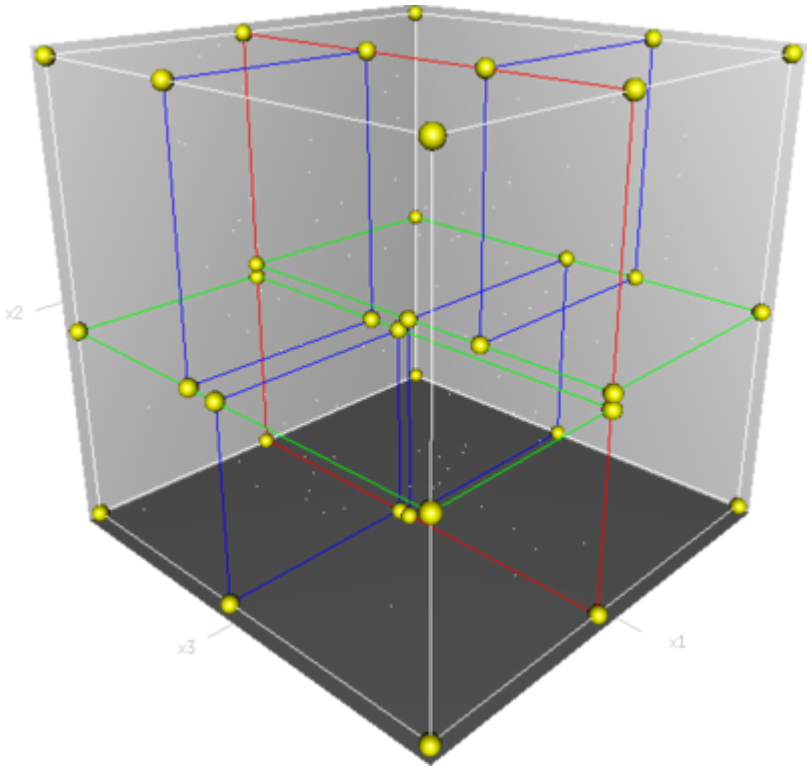

Ilustración 5. Representación de un árbol kd tridimensional

Un árbol kd, esquematizado en la ilustración 5, es una estructura de datos espacial que organiza los puntos en diferentes espacios de K dimensiones. La separación entre compartimentos es llevada a cabo mediante planos perpendiculares a alguno de los ejes del sistema de coordenadas. Todos los nodos de un árbol kd contienen algún punto, de tal manera que no se desaprovecha memoria almacenando nodos sin información.

Técnicamente, la letra k se refiere al número de dimensiones, por lo que la ilustración 5 podría ser llamada árbol 3d, aunque sin embargo se suele emplear la expresión “árbol kd tridimensional” ya que un árbol tridimensional es un término más amplio mientras que árbol ks se refiere al tipo concreto de árbol mencionado.

En la tesis de final de grado de S. Maximilian Fraiss de 2017, titulada**:** **Rendering Large Point Clouds in Unity**, el autor menciona en repetidas ocasiones que el aspecto de estructura de datos y formato de fichero almacenado es el del software **Potree**.

Potree es un sistema desarrollado y expuesto en la tesis de Markus Schuetz de 2015: **Potree – Rendering Large Point Clouds in Web Browsers**. Este software es un renderizador de nubes de puntos masivas de código abierto e implementado en WebGL. De acuerdo a la ilustración 6, esta herramienta nos permite realizar una gran cantidad de tareas, así como una cómoda interacción desde cualquier browser; y es por ello que se ha convertido en un estándar de facto de las nubes de puntos.

Ilustración 6. Ejemplo de visualización de una nube de puntos con el software Potree

Internamente, este software subdivide los puntos en una estructura arbórea especial, una **adaptación de un octree**. Los octrees son estructuras tridimensionales espacialmente que presentan una división recursiva de cada nodo interno en exactamente 8 hijos, particionando un espacio tridimensional de este modo. Por lo tanto, y en función de las zonas donde haya más congregación de puntos, esta estructura se extenderá recursivamente, permitiendo así que nuevos nodos hijos se encarguen de almacenar la información de las zonas más densas.

De acuerdo a la ilustración 7, podemos ver un ejemplo de un octree aplicado a un modelo tridimensional muy grande, donde las zonas simples no tienen gran cantidad de nodos mientras que las densas contienen gran cantidad de nodos más pequeños.

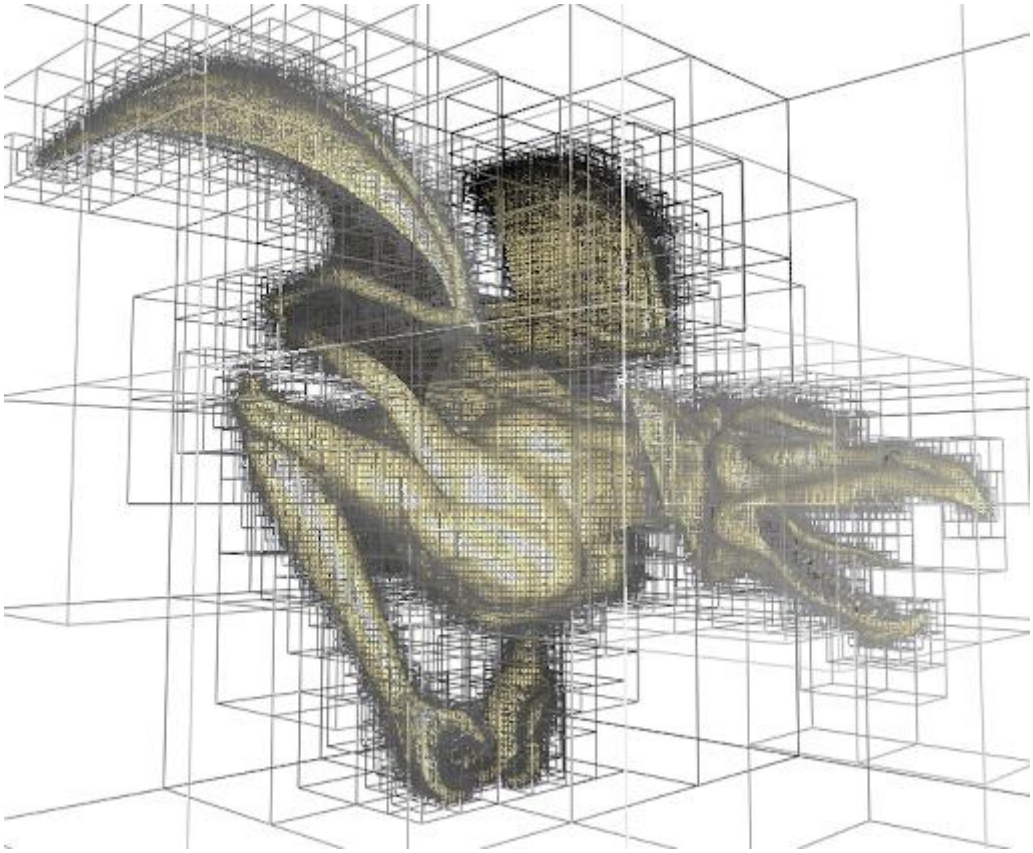

Ilustración 7. Subdivisión de un modelo complejo en un octree

Cabe mencionar que en el enfoque llevado a cabo en el paper **An out-of-core octree for massive point cloud processing** de K. Wenzel et al., de 2014 y desarrollado en el instituto de fotogrametría de la universidad de Stuttgart, donde los autores implementan, como el nombre del artículo indica, un **octree** como solución para almacenar nubes de puntos masivas.

No es el único enfoque llevado a cabo con octrees, a priori una estructura sencilla. En el paper de 2014**, Simple Octree Solution for Multi-resolution LiDAR Processing and Visualization** de Yu Fou et al. en el centro de investigación de Nokia, en Finlandia, los autores realizan una nueva aproximación a la visualización de nubes de puntos masivas mediante el planteamiento de un sistema altamente eficiente y que permita realizar operaciones sobre las nubes de puntos con distintas resoluciones y genérico, de tal modo que sea extrapolable a otros tipos de datos geográficos.

En la misma línea, y con la misma estructura, se puede mencionar el trabajo de 2007, **Processing and interactive editing of huge points clouds from 3D scanners** de Michael Wand et al. del instituto de informática Max-Planck de Saarbrucken, Alemania.

En este proyecto se vuelve a abordar el procesamiento y edición de nubes de puntos masivas, de un modo eficiente y apoyándose una vez más en la estructura espacial de **octree**. Sin embargo, en este trabajo, aun partiendo de la versión básica de octree, se le realizan una serie de cambios orientados a la adaptación a una versión dinámica del mismo que permita realizar una serie de operaciones comunes en las nubes de puntos.

Muy importante en este epígrafe resulta la estructura que incorpora el software Potree, una variación del propio octree, llamada **Modifiable Nested Octree (MNO)**. Una versión mucho más técnica y depurada cuyos detalles exceden los ámbitos de este proyecto y que darían mucho de qué hablar; simplemente debemos remarcar que una estructura de datos disjunta donde los nodos de niveles inferiores contienen subconjuntos de grandes volúmenes y que con cada nivel el tamaño de cada nodo encoge, aumentando así la densidad de puntos. De este modo, cada punto del Dataset original es asignado a exactamente un nodo del octree, biyectivamente. Por lo tanto, no se duplican ni se crean puntos nuevos, lo que permite realizar una serie de operaciones al usuario sobre puntos originales, no representativos de una zona, lo que impide obtener datos falseados. Un ejemplo de como funciona esta estructura por dentro aparece en la ilustración 8, obtenida de la tesis de Schuetz.

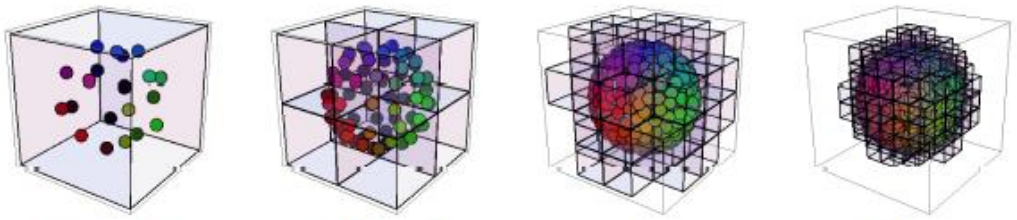

Figure 3.1: Low-level nodes (left) contain sparse models over large regions. Each level exponentially increases the number of points and details.

Ilustración 8. Nube de puntos esférica particionada en un Modifiable Nested Octree (MNO)

Sin embargo, no tenemos por qué recurrir al uso de estructuras tridimensionales obligatoriamente en el campo de las nubes de puntos. Un ejemplo lo podemos encontrar en el artículo de 2010, **Visualization of LIDAR datasets using point-based rendering technique** de Bostjan Kovac y Borut Zalik de la universidad de Maribor; donde abordan la visualización en tiempo real de datasets LiDAR, con un enfoque a más bajo nivel, utilizando shaders en GPU, con **quadtrees**. Los quadtrees son los equivalentes bidimensionales de los octrees, estructura tridimensional; donde en este caso las subdivisiones se producen de 4 en 4, siendo la filosofía de la estructura similar a la de los octrees, pero aplicada a un plano de menos dimensiones.

La estructura propuesta en este trabajo consiste en un grid disperso con 2 dimensiones espaciales con 1 extra de datos. Es decir, los puntos aparecen vinculados a nodos en el espacio euclídeo de acuerdo a sus posiciones en el eje X e Y; pero dentro de cada nodo distinguimos a su vez diferentes niveles de detalle disjuntos, conformados por puntos distribuidos uniformemente dentro del espacio que alberga la celda, que se agregan para generar un conjunto de datos más representativo conforme estamos más cerca de un nodo de la estructura. La implementación abordada en este proyecto se enmarcaría como una estructura bidimensional.

1.3.8 Operaciones comunes realizadas a nubes de puntos LiDAR

El concepto de nubes de puntos permite y simplifica una gran cantidad de tareas y funciones mediante la manipulación de las mismas, debido a su potencial intrínseco por albergar los datos de escáneres tridimensionales de zonas concretas. De una manera más simple, tenemos en una serie de puntos, toda una zona de gran relevancia para el estudio que estemos realizando.

Aunque en este trabajo, y en la gran mayoría de los mencionados en esta sección del estado del arte, se centren en la propia visualización, esta no es lo único que podemos hacer con las nubes de puntos. A continuación, y con la base del trabajo de 2008 **An Overview of Lidar Point Cloud Processing Software** de J.C. Fernández et al. del departamento de ingeniería civil y de costas de la universidad de Florida, se mencionarán las operaciones más comunes a este tipo de datos espaciales masivos estudiados en este proyecto, las nubes de puntos.

- **Selección de un único punto**. De esta manera se puede pivotar, rotar o aumentar la nube de puntos a partir de ese único punto.
- **Mediciones**. La capacidad de seleccionar puntos específicos de las nubes permite medir distancias entre puntos y obtener los ángulos que existen entre las líneas que conectan algún par de puntos.
- **Adaptación de primitivas**. Consiste en agrupar un grupo de puntos en una primitiva básica como un cilindro, cubo, esfera o plano de un modo similar a una bounding box, término que será explicado en la sección 2 de este trabajo y en la ilustración 16.
- **Segmentación**. Diferenciación y agrupación de puntos pertenecientes a un conjunto mayor de acuerdo a características no triviales. Por ejemplo, podemos tener una segmentación de acuerdo a los valores de intensidad de los mismos.
- **Clasificación**. Separación y agrupación de puntos de acuerdo a características intrínsecas o naturales. Poe ejemplo, podemos una tener una clasificación de puntos pertenecientes a zonas de vegetación, edificios o tierra.
- **Filtrado**. Eliminación de un conjunto de puntos de acuerdo a un esquema de segmentación o clasificación. Por ejemplo, podríamos considerar eliminar los puntos segmentados a un menor nivel de intensidad o clasificados como pertenecientes a una zona de vegetación.
- **Tranformaciones (traslación, rotado y escalado)**. Al igual que otros elementos tridimensionales, esta serie de primitivas también se le pueden aplicar a los puntos o a un conjunto de ellos.
- **Fusión**. Unión de dos nubes de puntos distintas en una sola con el objetivo de agrupar en una única nube final toda la zona abarcada por las originales.

1.4 Descripción de la situación de partida

Gran cantidad de implementaciones en diferentes entornos y con diversas tecnologías han dado buenos resultados para tareas específicas. Sin embargo, con este trabajo se pretende **seguir indagando en la vía de la visualización progresiva**.

Conforme las tecnologías avanzan, la cantidad de datos a tratar aumenta exponencialmente; provocando así la necesidad de diseñar soluciones que sean capaces de gestionar eficientemente tal información. Específicamente, en el ámbito de las nubes de puntos, son habituales en muchas ocasiones llegar a tener decenas de miles de millones de puntos relativas a un escaneo. La visualización y establecimiento de una calidad de interacción suficiente de tales cantidades de datos no son gestionables por cualquier software. De este modo, se plantea este proyecto en el que cada componente de mismo está estudiado precisamente para soportar grandes volúmenes de información.

1.4.1 Descripción del entorno actual

Como se acaba de mencionar, el ámbito de la visualización de nubes de puntos presenta una gran cantidad de trabajos y aproximaciones de muy diversas clases. Sin embargo, y dentro de un marco colaborativo como es el proyecto que lleva a cabo este grupo de investigación, se precisaba **una herramienta que pudiera, en tiempo real, trabajar con datos masivos de manera que la visualización, interacción y comunicación entre RAM y disco se naturalizase de tal manera que el retardo fuera mínimo**.

Con este proyecto, tenemos una herramienta que es capaz de **generar una estructura de datos parametrizable y desacoplada** (por lo que se puede sustituir fácilmente por otras) **a partir de datos masivos disjuntos y por otro lado visualizarla de una manera progresiva y adaptativa que impide la sobrecarga del sistema**. Esto acompañado de un **sistema de peticiones multihilo** permite plantear una aproximación acorde a las necesidades de un sistema de visualización de nubes de puntos masivas.

1.4.2 Resumen de las deficiencias y carencias identificadas

Los problemas que presentaba la visualización de nubes de puntos tradicional como se ha hecho hasta ahora con **enfoques monolíticos**, sin tener en cuenta la escalabilidad de estas técnicas, se van a poder solucionar con el paradigma de la visualización progresiva que sí considera esta problemática y la evita mediante el intercambio continuo de información entre los diversos hilos de ejecución y el control sobre qué nodos deben ser mostrados en pantalla y cuáles no**. El continuo incremento de la información a manejar obliga a desarrollar nuevas competencias dentro de los softwares visualizadores que tengan en cuenta tales circunstancias**.

1.5 Requisitos iniciales

Este proyecto, orientado a la visualización progresiva de nubes de puntos se adhiere a unos **objetivos que son detallados a continuación**:

- Debe poder procesar ficheros LiDAR con extensión .LAS y .LAZ indistantemente.
- Debe poder dividir el procesamiento en 2 etapas; una de preprocesamiento de los datos y otra de visualización
- Debe incorporar un sistema multihilo de peticiones de nodos y búsqueda de los mismos en estructuras de memoria.
- Debe incorporar el uso de una estructura de datos espacial que se vuelque en el sistema de ficheros.
- Debe incorporar un sistema de visualización progresiva de datos que proporcione una interacción fluida que permita desplazarse por la escena representada.
- Debe implementar alguno de los patrones de diseño más conocidos con el objetivo de escalar adecuadamente en futuras iteraciones del proyecto y se adhiera a los estándares del campo de investigación.
- Debe incluir una métrica que permita realizar peticiones en un formato preestablecido y eficiente.

1.6 Alcance

El entregable de este proyecto consiste en un **software ejecutable que nos permite procesar tanto los datos suministrados como realizar la visualización progresiva de los mismos**. Este software ejecutable, a su vez, también produce unos sub-entregables, ficheros LAZ ubicados en una estructura de carpetas en los que se trocea el *dataset* en diferentes conjuntos disjuntos por niveles de detalle y ubicación espacial a la par que un fichero de información de utilidad para su visualización. Estos ficheros, que conforman el último entregable, presentan un orden acorde a la ubicación dentro de la estructura de grid disperso volcada en disco.

1.7 Hipótesis y restricciones

El TFG se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.8 Estudio de alternativas y viabilidad

A la hora de emprender este proyecto se valoraron varias opciones respectivas a las diferentes componentes a tener en cuenta.

Lenguajes de programación	Características consideradas
C++	• Fácil integración de la vertiente gráfica • Capacidad de trabajar a bajo nivel • Experiencia • Existen trabajos previos con él en el grupo de investigación • Integración de herramientas LiDAR sencilla

Java	• Alto nivel de abstracción • Difícil integración de herramientas LiDAR • Vertiente gráfica no tan depurada como C++ • Menor experiencia que en C++
Python	• Aún más alto nivel de abstracción • Lentitud de ejecución • Vertiente gráfica no tan depurada como C++ • Menor experiencia que en C++

Tabla 4. Comparación de lenguajes de programación considerados para el proyecto

Es por ello, que en este proyecto finalmente se adoptó la decisión de usar el lenguaje de programación **C++**, siendo los principales determinantes para su elección la fácil integración con las bibliotecas LiDAR y su rendimiento con aplicaciones gráficas. En la tabla 4 aparecen recogidas los detalles considerados con respecto a los lenguajes de programación.

Estructura de datos espacial	**Características consideradas**
Grid tridimensional	• La EEDD espacial más sencilla • Es posible tener nodos vacíos • Memoria desperdiciada a causa de tener nodos sin puntos • Enfoque previamente estudiado en el grupo de investigación • 3 dimensiones espaciales
Sparse grid bidimensional	• Alta eficiencia en memoria al no permitir nodos vacíos

	• Enfoque propuesto en el grupo de investigación • Fácil comprensión • 2 dimensiones espaciales y 1 de datos • Esqueleto propuesto por el tutor del TFG
Octree	• EEDD jerárquica compleja • Implementada en otros proyectos • 3 dimensiones espaciales

Tabla 5. Comparación de estructuras de datos consideradas para el proyecto

Finalmente se optó por implementar un **grid disperso** debido a que los tutores de este TFG propusieron realizar la implementación del sistema de visualización desde esta perspectiva. En la tabla 5 se detalla una comparación entre las EEDD consideradas en este trabajo.

Herramienta de visualización	**Características consideradas**
OpenGL	• Características de muy bajo nivel • Gran esfuerzo hasta conseguir la primera visualización • Explorado anteriormente en el grupo de investigación
VTK	• Abstracción completa de gran cantidad de tareas • Primera visualización trivial • Explorado anteriormente en el grupo de investigación
Unity	• Potente motor de videojuegos

	• Explorado en el grupo de investigación para otros objetivos • Publicaciones científicas utilizan esta herramienta

Tabla 6. Comparación de herramientas de visualización consideradas para el proyecto

En este proyecto finalmente nos decantamos por la **herramienta VTK** debido a la fácil integración con la metodología de trabajo ágil llevada a cabo y su abstracción de componentes esenciales de toda visualización. En la tabla 6 figuran las herramientas de visualización consideradas en este proyecto.

1.9 Descripción de la solución propuesta

Esta solución software presentada se basa en varios conceptos con gran importancia en este proyecto como son, en primer lugar, el **uso de una estructura de datos espacial no jerárquica para crear un conjunto de ficheros** procesado para facilitar el segundo punto y clave; un **sistema de visualización progresiva de las nubes de puntos con interacción en tiempo real**. También podemos mencionar el tratamiento conjunto de ficheros. LAS o. LAZ, la adaptación a grandes conjuntos de datos y la parametrización del software que permita realizar un gran conjunto de pruebas y que lo dota de una **gran flexibilidad y escalabilidad**.

El uso de **la progresividad permite el escalado y la gestión de nubes de puntos masivas**; el sistema multihilo sincronizado consigue que la tarea de realizar y recibir peticiones se realiza eficientemente de acuerdo a las necesidades del proyecto y la amplia parametrización permite probar diversas configuraciones hasta encontrar la idónea para un Dataset determinado.

1.10 Material y métodos

Los medios empleados para realizar este proyecto han sido principalmente la **documentación relacionada con la herramienta VTK, también sobre la librería STL del lenguaje C++ y las reuniones periódicas** con ambos tutores del trabajo con el objetivo de discutir decisiones de diseño, plantear refactorizaciones y decidir el siguiente paso.

Con respecto a los Datasets utilizados y propuestos, este software se mostrado robusto al poder trabajar con variados ficheros .LAS y .LAZ sin ningún problema e independientemente de su tamaño. Ejemplos de experimentaciones con conjunto de datos masivos se pueden encontrar en el epígrafe 3, experimentación, resultados y discusión, de esta documentación.

De todos los medios empleados, el que ha cobrado especial importancia y que merece la pena remarcar es la documentación de la biblioteca VTK, que podemos ver en la ilustración 9.

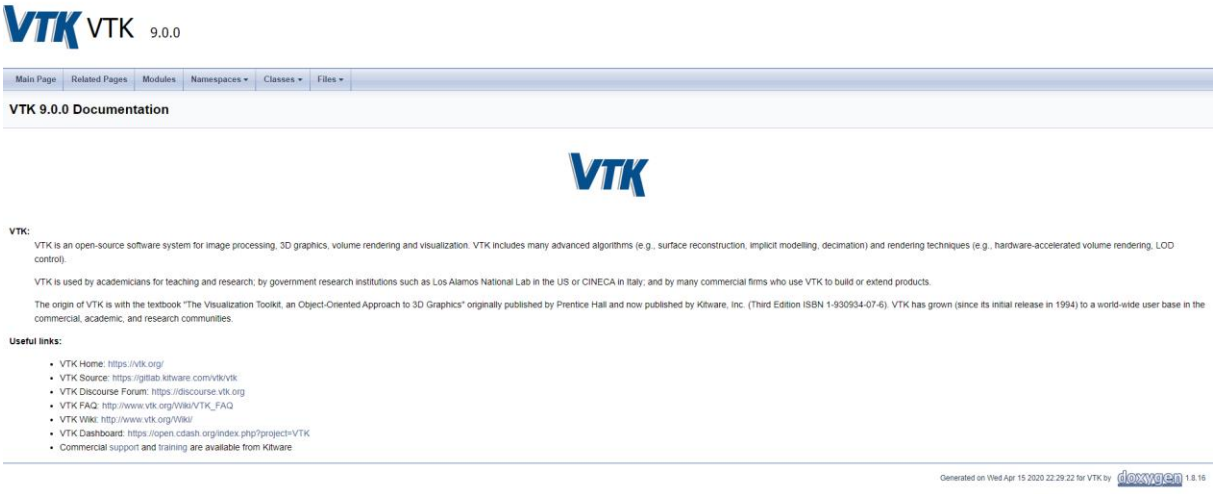

Ilustración 9. Web donde se encuentra la documentación de la biblioteca VTK

1.11 Tecnologías utilizadas

A la hora de emprender este trabajo, se barajaron diferentes alternativas en cuanto a las herramientas a usar, como se ha comentado anteriormente, y si bien es cierto que el desarrollo de este proyecto ha sido iterativo y cambiante, el objetivo primordial estaba definido desde primera hora.

Finalmente, de acuerdo a consideraciones y consultas con personas experimentadas en este ámbito dentro del grupo de investigación en el que se realizó este proyecto **llegamos a la conclusión de que las tecnologías que usaríamos serían las siguientes**:

- **Lenguaje de programación C++** debido a su fácil integración con librerías para procesar datos LiDAR, experiencia previa y fácil adaptación en temas de programación de aplicaciones gráficas.
- La elección de C++ implicaba directamente por su potencial el uso del entorno de programación **Visual Studio 2017** debido a su potencial de funcionalidades, adaptación a proyectos de envergadura y herramientas de depuración.
- **Entorno de visualización VTK** debido a su versatilidad, anteriores resultados satisfactorios en otros proyectos relacionados con el grupo, recomendación por parte del tutor y por su masiva documentación y comunidad.
- **Librería LASlibz**, que comprende a su vez más librerías, proporcionada por el tutor de este trabajo para poder así trabajar con el formato de fichero LAS y LAZ indistintamente.
- **Librería GLM (OpenGL Mathematics)** para poder almacenar y acceder a los datos de los puntos de una manera cómoda, intuitiva y eficiente.
- **Sistema de control de versiones Git** debido a mi experiencia previa en su uso y gozar de amplia popularidad y soporte.

El desarrollo de este proyecto parte sin ninguna base más allá que las herramientas mencionadas anteriormente y la ayuda de los tutores, centrándose en una metodología de desarrollo incremental en la que una versión ejecutable siempre estaba presente pero sujeta a cambios y refactorizaciones continuas

1.12 Metodología de desarrollo de software

La metodología llevada a cabo en este proyecto ha sido **puramente ágil**. En todas las diferentes etapas de este proyecto, incluso desde las más tempranas, una versión ejecutable estaba disponible. Estas diferentes versiones ejecutables siempre estaban sujetas a cambios y refactorizaciones continuas, de acuerdo a la filosofía de este tipo de desarrollo.

Después de cada avance, previamente consensuado, se definieron nuevos objetivos a cumplir y este proceso recursivo se extendió a lo largo de todo el desarrollo del trabajo.

1.13 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFG, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.14 Planificación temporal

En este epígrafe se detallará en qué se invirtió el tiempo de diseño y desarrollo. Se debe tener en cuenta que al seguir este proyecto una metodología de desarrollo ágil, con continuas refactorizaciones y cambios; la influencia de este tipo de técnicas también repercute en la planificación del progreso del software ya que antiguos componentes deben ser actualizados continuamente.

Debemos remarcar el detalle de la pandemia ocurrida durante el transcurso del curso académico ocasiónó un cambio entre la cronología

planteada y la finalmente seguida. En la ilustración 10 se puede ver el diagrama de Gantt referente a la planificación primeramente definida.

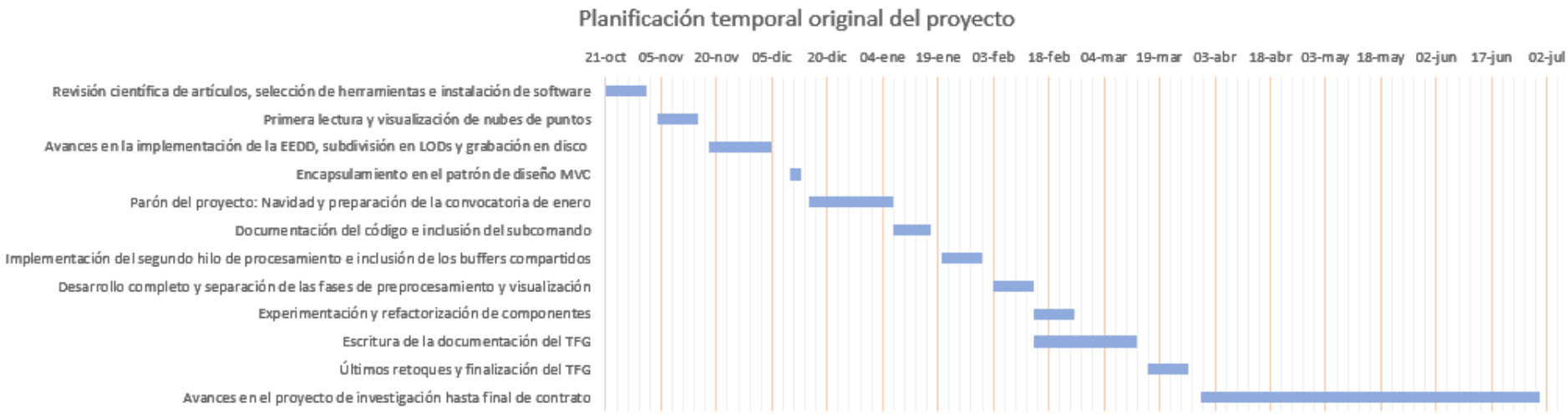

Ilustración 10. Planificación temporal original del proyecto

De acuerdo a los cambios por la situación, el orden de tareas realizadas de acuerdo a la programación primeramente planteda, es el siguiente de la tabla 7.

Tarea	Fecha de inicio	Fecha de finalización
Revisión científica de artículos, selección de herramientas e instalación de software	21/10/2019	1/11/2019
Primera lectura de ficheros .LAS y .LAZ.	4/11/2019	8/11/2019
Primera visualización de nubes de puntos	11/11/2019	15/11/2019
Creación de la clase Cloud y de la estructura de datos del grid disperso. Carga de datos en ambas instancias	18/11/2019	22/11/2019
Desarrollo en mayor profundidad y depuración de las clases Cloud y Sparse Grid. Implementación de Smart Pointers. Primera versión estática de subdivisión de una nube de puntos en niveles de detalle	25/11/2019	29/11/2019
Modificación para permitir la subdivisión en niveles de detalle dinámicos. Inclusión del grabado de los datos generados en una jerarquía de carpetas en disco. Mejora de la visualización	2/12/2019	5/12/2019
Inclusión de la clase Dataset. Mejora de la interacción e inclusión del subcomando periódico	10/12/2019	13/12/2019
Documentación en formato Doxygen de todo el código. Recopilación de aspectos mejorables. Planteamiento de las siguientes iteraciones	7/1/2020	10/1/2020
Primera versión de visualización progresiva en un único hilo y definición de los parámetros iniciales	13/1/2020	17/1/2020
Mejora de la documentación. Mayor encapsulación y limpieza del código. Refactorización de la subrutina periódica.	20/1/2020	24/1/2020

Planteamiento del hilo que atiende peticiones e inclusión de secciones críticas		
Finalización del segundo hilo que atiende peticiones. Primera definición de buffers compartidos. Lectura de los datos preprocesados residentes en disco en una fase de visualización completada	27/1/2020	31/1/2020
Mejoras en la visualización progresiva. Creación de fichero de log que ayuda a la visualización. Mayor encapsulamiento en el paradigma Modelo-Vista-Controlador.	3/2/2020	7/2/2020
Corrección de bugs de menor importancia. Colocación de la cámara al comenzar la ejecución. Desacoplamiento total entre vista y controlador	10/2/2020	14/2/2020
Avance en la documentación y escritura de este documento	24/2/2020	27/2/2020
Finalización de una versión estable mediante la encapsulación total del modelo y buenos resultados de rendimiento.	9/3/2020	13/3/2020
Recuperación del material del laboratorio e instalación en nuevo equipo. Inclusión de la pausa y reanudación del intercambio entre buffers. Realización de experimentos y finalización de la memoria.	1/4/2020	31/5/2020

Tabla 7. Planificación temporal final del proyecto

De acuerdo a la tabla 7, las diferentes tareas realizadas aparecen recogidas de acuerdo a unas fechas de inicio y fin marcadas. El desarrollo de este proyecto ha sido ininterrumpido salvo en tres momentos del mismo:

1. La navidad de 2019-2020 no presenta desarrollo alguno por motivo del transcurso del periodo de exámenes del primer cuatrimestre.

2. La semana del 17 al 21 de febrero no se realizó ningún avance significativo de mencionar (solo leves retoques) debido a una gripe del autor de este trabajo.

3. El lapso de 2 semanas comprendido entre el 13 de marzo y el 1 de abril de 2020 se debió al parón por el COVID-19 que afectó a los laboratorios de investigación de todas las universidades españolas. Después de ese periodo, se avanzó en la redacción de este documento a la par que coincidió con el periodo de exámenes y entrega de trabajos del segundo cuatrimestre. El martes 19 de mayo finalmente se pudo volver a visitar el laboratorio para recoger el material necesario para poder finalizar el proyecto en casa.

1.15 Presupuesto

Con respecto a los gastos vinculados al proyecto, podemos hacer una división tanto en componentes hardware como vinculados al personal. Los gastos que puedan ocasionar el uso de determinado software no se incluyen debido a la existencia de numerosas licencias proporcionadas por la universidad tanto para Windows, Word, y Visual Studio. El resto de elementos software usados, tales como bibliotecas, datasets o artículos científicos son de uso libre o han sido proporcionados por los tutores del proyecto, pertenecientes al grupo de investigación GGGJ.

Con respecto a los recursos hardware usados, se incluirán las características del equipo en el que se realizó la experimentación, que difiere en el que se realizó el diseño por la situación originada por el COVID-19.

Recurso	Precio	Tiempo de uso en el proyecto	Amortización mensual del recurso
Intel Core i4-4440 @ 3.10 GHz	182 €	8 meses	22.75 € / mes
AMD Radeon R7 265 Dual-X de 2 GB GDDR5	163 €	8 meses	20.38 € / mes

SSD Crucial MX500 de 250 GB	52.02 €	8 meses	6.51 € / mes
8GB de memoria RAM Kingston HyperX Fury Blue	44 €	8 meses	5.5 € / mes
Fuente de alimentación CoolBox Poweline Black 600 W	33.16 €	8 meses	4.15 € / mes
Placa base Gigabyte GA-H81M-S1	46 €	8 meses	5.75 € / mes
Caja L-Link Q7 USB 3.0	34 €	8 meses	4.25 € / mes
Monitor principal BenQ GW2480E 23.8'' LED IPS FullHD	109 €	8 meses	13.63 € / mes
Monitor secundario BenQ VL2040AZ 19.5'' LED	78.95 €	8 meses	9.87 € / mes
Teclado mecánico marca LESHP	21 €	8 meses	2.63 € / mes
Ratón SteelSeries Rival 100 4000 DPI	40 €	8 meses	5 € / mes
Auriculares Corsair HS60 Gaming Surround	95 €	8 meses	11.88 € / mes
Total	**898.13 €**	**8 meses**	**112.266 € / mes**

Tabla 8. Gastos del material hardware del proyecto

De acuerdo a la tabla 8 podemos ver el material hardware usado en el proyecto junto a su precio actual consultado en la web, además de su amortización durante los 8 meses de duración del desarrollo.

En otro orden de gastos se encuentran los derivados del personal, los cuales podemos categorizar de acuerdo a la cuantía que otorgan las becas de colaboración del Ministerio de Educación y Ciencia, mediante la cual se realizó este proyecto.

Recurso	Dotación económica	Duración del proyecto	Amortización mensual del sueldo
1 becario de colaboración del Ministerio de Educación y Ciencia	2000 €	8 meses	250 € / mes

Tabla 9. Gastos en personal del proyecto

En la tabla 9 aparecen reflejados los gastos económicos derivados en personal del proyecto. Y a continuación, en la tabla 10, podemos ver el gasto total del proyecto.

Elementos del presupuesto	Coste asociado
Material hardware	898.13 €
Personal	2000 €
Total	**2898.13 €**

Tabla 10. Gastos totales del proyecto

También debemos tener en cuenta los gastos indirectos derivados del proyecto como uso de instalaciones, desplazamientos o facturas de servicios básicos. Para ello

debemos incluir un sobrecoste del proyecto del 10% sobre la estimación inicial, 289.81€.

Después incluir estos gastos secundarios, el **montante total** del presupuesto asociado al proyecto asciende hasta los **3187.94 €**

2 DESARROLLO

En este epígrafe se detallarán todas las iteraciones del proyecto, para cada una de las cuales se describirán las decisiones de diseño tomadas y los correspondientes detalles de implementación a destacar.

2.1 Primera Iteración

2.1.1 Decisiones de diseño

Dentro de esta primera iteración y en el marco del inicio del proyecto, el primer objetivo, básico y primordial, era definir la manera en las que almacenásemos los puntos y la manera de leerlos. Nos planteamos dentro del equipo dos enfoques, uno en el que se almacenasen los puntos como propio objeto y otro, más sofisticado y eficiente, que era utilizar buffers con sus atributos.

Otra decisión que tuvimos que tomar, aunque parece muy sencilla, era la de qué parámetros almacenar de cada medida del escaneo. Dentro de la tecnología que rodea a los datos LiDAR, no solamente se almacena información espacial, sino que cada punto tiene asociado también una gran variedad de metadatos como el color, ángulo y dirección del escaneo, posición del GPS, intensidad, etc.

Tras estudiar las posibles alternativas, y debido a la rápida escalabilidad que nos permiten tener este sistema de buffers, optamos por almacenar:

- Información espacial tridimensional (**coordenadas X, Y, Z**) referente a la posición relativa en la escena respecto al primer punto leído. Del primer punto que leamos, almacenaremos su posición y lo consideraremos el centro de la nube. A partir de ahí, a cada punto que coloquemos en nuestros buffers le restaremos las coordenadas suyas a las definidas por el punto central para que de tal manera las posiciones sean correlativas al centro.

$$(Coordenadas\ finales) = (Coordenadas\ originales) - (Coordenadas\ centro)$$

- Información del color (**coordenadas R, G, B**) referente a cada punto escaneado. Estos valores tendrán un rango comprendido entre 0 y 255, siguiendo el modelo de color RGB previamente mencionado. Es importante saber que un único índice, y aunque manejemos dos buffers distintos, representa al mismo punto; es decir, ambos buffers tienen el mismo tamaño y, con un índice, podremos acceder a los valores de posición y de color del mismo punto y con complejidad O(1).

2.1.2 Detalles de implementación

Tras definir los objetivos iniciales de esta primera iteración, lo primero que se hizo fue crear y poner a punto las librerías a utilizar, el control de versiones y realizar las primeras pruebas con el objetivo de verificar que todo estaba correctamente acoplado y así evitar posteriores problemas y pérdidas de tiempo.

Después de terminar estas tareas pre-proyecto, se definió la clase **Reader,** que encapsula la lectura de los puntos y que se encargará de abrir los ficheros que se encontrasen en una carpeta cuya ruta sea introducida por el usuario.

Las estructuras de datos y el tipo de objeto usado para poder almacenar los puntos son vectores de STL del tipo glm::vec3, este tipo permite guardar ternas de valores numéricos, idóneos para nuestro diseño.

2.1.3 Pruebas realizadas

Las pruebas realizadas consistieron en una **lectura de los archivos presentes en un directorio** cuya ruta aparece indicada por parámetro de tal modo que después de la lectura del mismo la instancia Reader contuviera todos los puntos de todos los ficheros. En esta etapa no se incluyó ninguna tarea relacionada con estructuras de datos ni con sistemas de visualización; algo a trabajar en las sucesivas iteraciones y el objetivo principal buscado se basaba en depurar el sistema de lectura de datos.

2.2 Segunda iteración

2.2.1 Decisiones de diseño

En esta segunda etapa en la que todos los puntos estaban ya traídos y leídos de disco, era **poder visualizarlos**, después de estudiar la manera en que nuestro software de visualización VTK puede renderizar una serie de puntos contenidos en las respectivas estructuras de datos de visualización que necesita.

2.2.2 Detalles de implementación

La ampliación de código llevada a cabo consistió en volcar los buffers leídos de los puntos en dos nuevos vectores de STL. Una vez con todo preparado en el ámbito de una función, el siguiente paso era crear las estructuras y objetos necesarios para determinar el proceso de visualización de VTK.

Conviene mencionar el hecho de que VTK trabaja todos sus objetos con Smart Pointers, por lo que cuando se menciona un tipo de dato relativo a VTK, éste será un vtkSmartPointer y se creará con la plantilla de este tipo.

En esta primera versión, primitiva, de visualización, primero se crea una estructura de puntos (vtkPoints) y otra de colores, que se almacenará en un array de caracteres unsigned (vtkUnsignedCharArray); justo después se procede a volcar los datos de los vectores de STL en las estructuras correspondientes con los métodos InsertNextPoint(X, Y, Z) e InsertNextTupleValue(R, G, B), respectivamente.

Al tener estos dos objetos ya creados, se crea el vtkPolyData y se le asignan los puntos y los colores con las órdenes SetPoints(vtkPoints) y GetPointData()->SetScalars(vtkUnsignedCharArray). Seguidamente, se instancia un objeto del tipo vtkVertexGlyphFilter que en versiones posteriores fue reemplazado y cuya funcionalidad era albergar un vtkPolyData para posteriormente volcarlo en otro; un enfoque que posteriormente se cambió ya que no era eficiente ni se correspondía a lo que buscábamos.

Después de tener el vtkPolyData creado y con los dos buffers dentro, se procede a crear un vtkPolyDataMapper y asignarle el vtkPolyData con el método SetInputData(vtkPolyData). La funcionalidad de este último objeto es, de acuerdo a la documentación de VTK, mapear un vtkPolyData a las primitivas gráficas, es decir, constituye un paso intermedio dentro del pipeline de visualización.

Una vez terminado el paso anterior, se instancia un vtkActor, estructura que representa un objeto (tanto su geometría como sus propiedades en la escena renderizada), se vincula al mapper con la sentencia SetMapper(vtkMapper) y se establece el tamaño de los puntos a un tamaño fijo parametrizable que en este problema será fijo dentro del actor con el método SetPointSize(int) ya que, como se comenta en epígrafes anteriores, no consideramos el tamaño de las medidas en nuestro problema.

Una vez se tiene encapsulado en el actor los puntos obtenidos del escáner LiDAR, se pasa a crear 3 objetos primordiales en la visualización y base de la gran mayoría de visualizaciones en VTK:

- **vtkRenderer**: Objeto que se encarga de abstraer el proceso de renderizado de objetos. Este proceso de renderizado se encarga de convertir la geometría, luz y la vista de la cámara en la imagen que vemos en pantalla.
- **vtkRenderWindow**: Objeto que crea la ventana que permite a los renderers dibujar. Se encarga de abstraer el comportamiento de una ventana de renderización. Esta ventana constituye una interfaz gráfica de usuario donde los renderers dibujan sus imágenes.
- **VtkRenderWindowInteractor**: Objeto que provee de un mecanismo para el control de la interacción de eventos temporales, de ratón y de teclado. También se encarga del control de los frames que vemos por pantalla.

La manera de vincular estas 3 estructuras básicas de visualización es añadiendo el vtkRenderer al vtkRenderWindow con la función AddRenderer(vtkRenderer) y vinculando sucesivamente esta vtkRenderWindow dentro del vtkRenderWindowInteractor con el método SetRenderWindow(vtkRenderWindow), como aparece en la ilustración 11.

Ilustración 11. Pipeline de visualización de VTK

Finalmente se debe añadir el actor de antes al vtkRenderer con la sentencia AddActor(vtkActor) y llamar a la función Render del objeto vtkRenderWindow para comunicar al vtkRender que empiece a renderizar y al método Start() del vtkRenderWindowInteractor que se hará con el control del hilo de interacción y se encarga de gestionar los diferentes eventos previamente mencionados.

2.2.3 Pruebas realizadas

Las pruebas realizadas consistieron en visualizar una nube de puntos procedente de los buffers que ya habíamos sido capaces de leer en la primera iteración.

La primera versión de visualización presentaba muchos problemas como que los puntos no tenían apariencia redonda, sino que eran cuadrados, la interacción dejaba mucho que desear y el rendimiento era muy pobre.

Sin embargo, **sacamos varios aspectos positivos** de esta fase como que el uso de la estructura de PolyData funcionaba, y que la definición del canvas y el posicionamiento de la cámara en el espacio eran dos problemas solucionados automáticamente por VTK.

El **siguiente paso** sería **avanzar en el ámbito de la encapsulación y de la estructura de datos, a la par que mejorar la calidad de la visualización**.

2.3 Tercera iteración

2.3.1 Decisiones de diseño

Una vez terminada la etapa anterior y en la que comprobamos que todo el proceso de lectura y renderización estaba asentado, a expensas de una mejora, pasamos a considerar la siguiente parte del proyecto, consistente en una **encapsulación de los buffers de posiciones y de colores** dentro de una nueva clase, la **clase Cloud**. Esta nube también almacena el centro de la misma y representa a **cada subconjunto de puntos que decidamos almacenar de una manera conjunta**. Esta clase tendrá una importancia primordial a lo largo del desarrollo del trabajo debido a que la estructura de datos espacial que definiremos está orientada a almacenar nubes.

Una vez definimos esta clase, pasamos a definir la estructura de datos espacial base de este proyecto, un **grid disperso**. En la ilustración 12 podemos ver una representación gráfica simplificada de la misma.

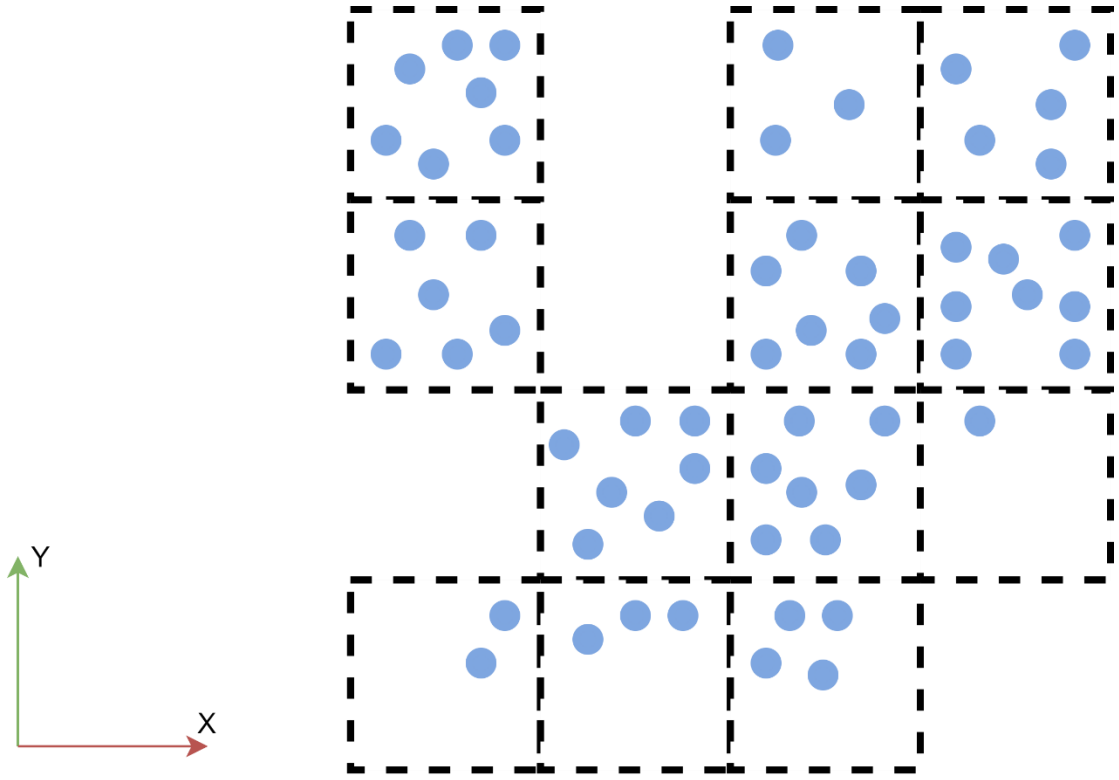

Ilustración 12. Estructura de datos espacial de grid disperso

En este contenedor, esquematizado en la ilustración 11, la **clave** consistirá en un **par de coordenadas** enteras (X, Y) que identifican a un nodo de la estructura a partir de su posición X e Y en el plano. Los valores que almacenaremos en este contenedor serán nubes de puntos como hemos descrito en este mismo apartado. Para poder obtener la clave relativa a cada nube de puntos, se ha utilizado **una función hash que permite asignar a cada punto con el que trabajemos, el nodo apropiado**; es decir, para cada punto calcularemos su valor hash (una operación matemática con sus coordenadas (X, Y)) y lo insertaremos en la nube correspondiente en función de las coordenadas (X, Y) que tenga.

Finalmente nos marcamos un objetivo adicional en la práctica de **poder definir los puntos de una forma redondeada**, y de esta manera evitar la visualización de los puntos como cubos.

2.3.2 Detalles de implementación

El paso más importante en esta etapa es la definición de la estructura de grid disperso, que estará representado por la estructura de datos de STL, **unordered_map**. Este mapa desordenado es un contenedor asociativo que almacena elementos formados por la combinación de un valor clave y un valor mapeado por el mismo identificador unívoco, lo que permite un rápido acceso a elementos individuales a partir de su clave.

En esta estructura, las claves serán instancias del tipo glm::ivec2, que representan un par de coordenadas (X, Y) del plano y el valor que se almacenará dentro serán nubes, es decir, instancias a su vez de la clase que definimos en la iteración anterior y que definen a un subconjunto de puntos con posiciones correlativas. Es necesario definir una función de hashing debido a que trabajamos con un tipo de clave importado de la biblioteca externa GLM; esta simple operación matemática multiplica la coordenada X e Y por los números primos 18397 y 2048,3 respectivamente y suma sus valores en un número entero final que servirá para identificar al nodo de la estructura en el que irá destinado el punto.

En esta primera versión del grid disperso no consideramos aún el tamaño de cada nodo por lo que se generarán un gran conjunto de celdas al tener una función de dispersión tan amplia. Esta problemática se tendrá en cuenta en sucesivas iteraciones.

El último objetivo definido era importante abordarlo ya que era inadmisible el tener un sistema de visualización progresiva de nubes de puntos con forma cúbica, sin embargo, había muchas alternativas y se optó por la que un compañero del grupo tenía implementada en un software totalmente distinto, pero con la misma librería VTK. Al actor le asignaba la propiedad SetRenderPointsAsSpheres(); de esta manera la renderización tendría en cuenta que los puntos asociados al polydata deberían ser esféricos. Esta aproximación fue adaptada a nuestras necesidades, teniendo que realizar una serie de pequeños cambios a nivel de código como almacenar los vértices a la hora de insertar los puntos y realizando la asignación correspondiente al polydata. La estructura usada para guardar los vértices fue del tipo vtkCellArray, objetos usados para representar conectividad entre celdas y que almacena topologías del dataset en cuestión como una tabla de conectividad explícita que lista los identificadores de los puntos que componen cada celda. Para ello almacenamos el id de cada punto al

insertarlo, que será del tipo vtkIdType, y lo insertamos en una celda de tamaño 1 con la función InsertNextCell del vtkCellArray.

Con respecto al resto del proyecto, no se realizaron cambios mayores a parte de la integración del contenedor en las clases correspondientes añadiendo métodos que facilitasen su creación.

2.3.3 Pruebas realizadas

El testeo de esta iteración consistió el **volcar los puntos en instancias de la clase Cloud y SparseGrid, ya que la estructura de datos, ya planteada en esta etapa está orientada a almacenar nubes de puntos como valor vinculado a la clave**; dada por el par de coordenadas espaciales X e Y. De este modo **se depuraron errores de programación** de tal modo que se pudiera avanzar sobre una base asentada robusta.

Con respecto a la solución de errores de etapas anteriores, se mejoró la visualización mediante la renderización de puntos esféricos en vez de cuadrados; además los puntos ya se procesan en la EEDD espacial, aunque ésta **requiere una mayor depuración de acuerdo al número de nodos generado**; un aspecto tenido en cuenta en la siguiente iteración.

2.4 Cuarta iteración

2.4.1 Decisiones de diseño

Como mencioné en la iteración anterior, nuestra función de dispersión nos devolvía una gran cantidad de valores, mapeando así un número ingente de nodos en nuestra estructura de datos. Si no atajamos este problema en el instante siguiente de definirlo, esta sobrecarga de nodos afectaría negativamente tanto al rendimiento de nuestro sistema en memoria volátil como persistente.

Es por ello que el primer objetivo que nos marcamos tanto tutores del proyecto como el autor del mismo fue **reducir la dispersión producida al mapear puntos y nubes**.

La alternativa que consideramos fue un enfoque sencillo y propenso a experimentos como definir un **tamaño de celda parametrizable**. Por lo tanto, a la hora de obtener la clave que identifica a una nube en el grid disperso, dividiremos

tanto la coordenada X obtenida como la Y por este tamaño de celda. De esta manera **acotaremos el rango de nodos que podremos tener**, disminuyendo el número de nubes en la estructura, el número de ficheros en disco y, además, permite realizar una serie de pruebas con diversos tamaños de celda con el objetivo de encontrar el que mejor se amolda a un determinado dataset.

Otro punto importante que abordamos en esta cuarta etapa de desarrollo contempla la **división en niveles de detalle de una nube de puntos**. En un enfoque clásico, cuando visualizamos un escaneo LIDAR, estamos renderizando todos los puntos que el/los fichero/s contienen. Sin embargo, si desde nuestro dispositivo, una determinada región en la que tengamos una gran concentración de puntos ocupa una pequeña porción de la pantalla, estaremos desperdiciando recursos valiosos de nuestro sistema al mandar procesar una cantidad de puntos mayor a la que podremos visualizar y perjudicando así el rendimiento del mismo.

Con este apunte negativo presente en algunos visualizadores, y con el objetivo de ir definiendo los pilares de la visualización progresiva posterior, era la hora de abordar la **multiresolución**. Para ello, **dividiremos una nube en diferentes componentes disjuntas y que se puedan tratar independientemente**, de tal manera que, si tenemos una gran cantidad de medidas en una zona de la pantalla que por sus dimensiones no puede mostrarlas completas, solamente visualizar una porción de la misma.

Básicamente, el objetivo de definir unos diferentes niveles de detalle (Levels of detail) es **rebajar el consumo de recursos del dispositivo que se encargue de la visualización para poder así asegurar un alto rendimiento de la aplicación y una interacción del usuario fluida, a costa de la resolución**.

La división de un conjunto de puntos en diferentes subconjuntos disjuntos no es trivial debido a que la definición de las medidas realizadas posee un orden y si solamente nos limitamos a escoger los primeros n puntos, solamente abarcaremos una porción del espacio; obviando así otras zonas. Por lo tanto, **la aproximación que debemos realizar debe consistir en elegir puntos uniformemente distribuidos por la nube** para así asegurar que abarcamos toda su extensión y evitar que con niveles de detalle bajos lo que mostramos por pantalla diste mucho del escaneo completo. En la ilustración 13 podemos ver un ejemplo básico y visual de esta tarea.

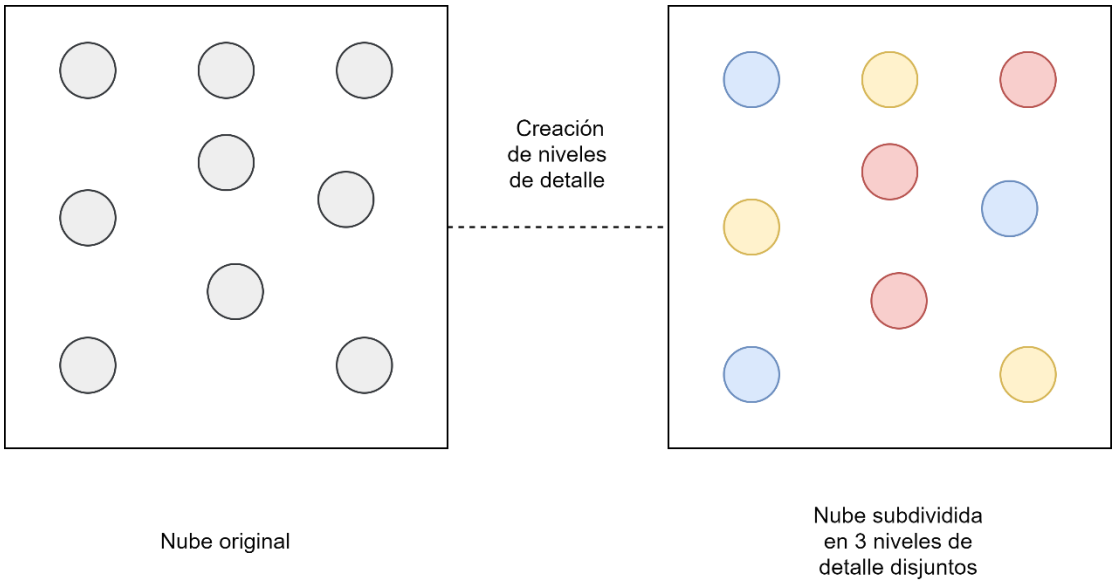

Ilustración 13. Subdivisión de una nube de puntos en niveles de detalle

Con esta figura podemos ver gráficamente en qué consiste la creación de niveles de detalle: en una **subdivisión aleatoria de los puntos originales, pertenecientes a una única nube, en nuevas nubes**.

2.4.2 Detalles de implementación

Con respecto a la implementación de esta nueva iteración, el proceso de reducir el mapeado de la función de dispersión del grid disperso fue trivial al solamente tener que aplicar la división de las coordenadas X e Y de los puntos por el tamaño de la celda, el cual es parametrizable y por defecto es de 100.

El objetivo posteriormente definido de la **subdivisión en niveles de detalle** fue más complejo ya que **debemos considerar la parametrización de los mismos por parte del usuario**. Además, debemos considerar la posibilidad de que en determinados conjuntos de datos unos niveles de detalle nos proporcionen mejor rendimiento que otras pruebas. El proceso de realizar la subdivisión aparece detallado en el siguiente diagrama.

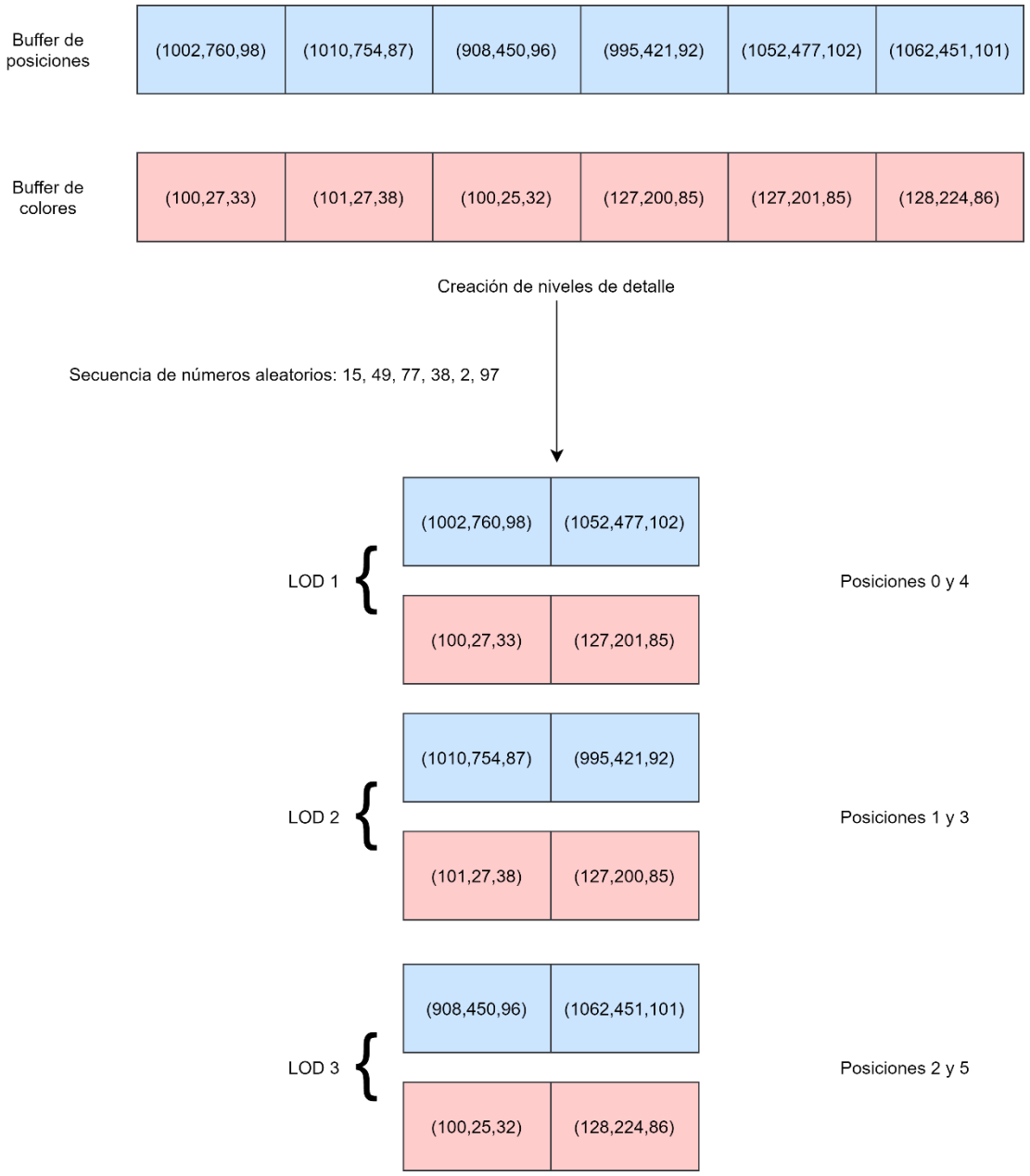

Ilustración 14. Algoritmo de subdivisión de una nube de puntos en niveles de detalle disjuntos

En la ilustración 14 se describe el funcionamiento del algoritmo gráficamente. El proceso consiste en generar un número aleatorio del 1 al 100 por cada punto y en función del tamaño que tiene cada nivel de detalle deseado, mapear ese punto a un determinado nivel de detalle (aunque en el ejemplo he planteado 3 niveles de detalle similares al 33.3%, normalmente en la práctica los niveles de detalle son incrementales). Para generar estos números pseudoaleatorios, tendremos que partir de una distribución uniforme que genere valores en el rango [0,100) y en función de su valor, insertar el punto en el nivel de detalle correspondiente. En la figura, valores

del 0 al 33, se insertarían en el LOD 1, del 33 al 66 en el LOD 2 y del 66 al 99 en el LOD 3.

2.4.3 Pruebas realizadas

Esta iteración resultaba crítica en el desarrollo del sistema de visualización progresiva. Una vez testeado y probado el rendimiento con diferentes tamaños de celda (algo recogido en el epígrafe de experimentación de este proyecto), **el objetivo principal consistía en asegurar que cada una de las subdivisiones de las nubes de puntos originales contenían elementos disjuntos, tuviesen un número de puntos acorde a lo preestablecido y que la unión de los LODs conformase la nube de puntos original**.

Sin embargo, una **parametrización de estos niveles de detalle era un requisito obligatorio**; y por ello fue abordado en la iteración sucesiva.

2.5 Quinta iteración

2.5.1 Decisiones de diseño

En esta nueva iteración, **nos marcamos el objetivo de poder aplicar la creación de LODs dinámicos a todas las subnubes del Sparse Grid**. Este cambio, al tener la encapsulación de esta operación en la clase Cloud, es trivial ya que solo precisa de iterar por la estructura y aplicar la operación por los nodos. **Otro aspecto** que consideramos implementar en esta etapa fue el **grabar los nodos en disco de acuerdo al orden establecido por los nodos**. Cada clave identifica unívocamente mediante un par X, Y a una nube, la cual tiene a su vez diferentes subnubes que conforman los niveles de detalle incrementales y disjuntos. La estructura de la EEDD espacial planteada en este proyecto y sobre la cual aplicar el almacenamiento la podemos ver en la ilustración 15.

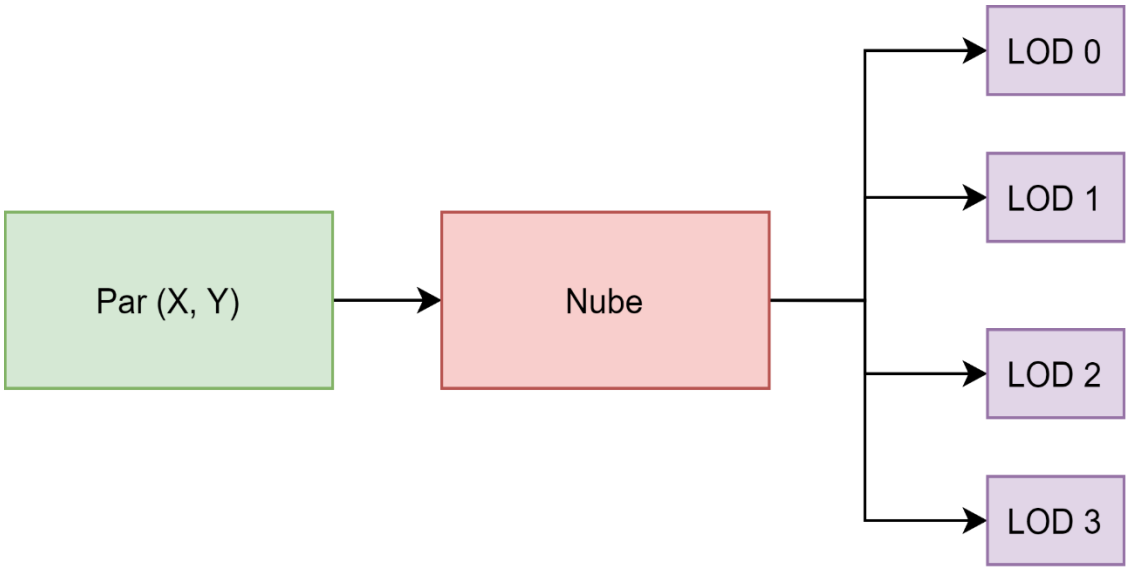

Ilustración 15. Estructura interna del Sparse Grid

Será la ilustración 15 la que nos facilitará la organización en memoria persistente. En la ruta definida en la que decidamos guardar todos los ficheros creados, crearemos tantas carpetas como nodos tengamos cuyo nombre será **X(valor X de la clave)Y(valor Y de la clave)**. Dentro de estas carpetas, almacenaremos los niveles de detalle, empezando a enumerarlos desde LOD0 hasta LODN, en orden incremental de tamaño donde los LODs con menos puntos tendrán un número más bajo y que por tanto son los que únicamente se deberían ver desde más lejos, dejando los ficheros más densos para cuando el nodo esté más cerca de la cámara y requiramos mayor detalle.

El último aspecto que consideramos necesario abordar fue una mejora de la interacción del software. La interacción que nos provee por defecto VTK deja mucho que desear: controles nada intuitivos y complicadas combinaciones de teclas. Para facilitar la visualización se definió el objetivo de estudiar las diferentes herramientas y maneras de gestionar la interacción por VTK e implementar la adecuada a nuestras necesidades de acuerdo a los dispositivos que manejaremos: teclado y ratón.

2.5.2 Detalles de implementación

El primer objetivo marcado fue sencillo de implementar debido a la encapsulación que habíamos hecho previamente de la función que se encarga de crear los LODs. Para ello lo que teníamos que hacer era iterar por todos los nodos del

sparse grid, llamando a la función que se encargaba de hacer la creación de los niveles de detalle (reordenación de los buffers). Sin embargo, en función de cada dataset y de los parámetros configurables es posible que no merezca la pena dividir en diferentes ficheros nodos muy pequeños. Para ello, **se define un nuevo atributo parametrizable, el tamaño mínimo de nube para aplicar esta transformación o no**. Si esto no ocurría, no malgastamos recursos en pocos puntos, rebajando la carga de trabajo del software, que, si bien en la fase de creación y división en niveles de detalle es muy baja, conviene tratar eficientemente desde las primeras etapas del proyecto.

Es importante remarcar que como nuestro software se separa en dos partes diferenciadas, una de tratamiento de los escaneos y otra de visualización de los mismos, tenemos que poder mantener los datos en almacenamiento masivo para que no se borren de ejecución en ejecución ya que la estructura no se almacena en memoria volátil después de la ejecución del programa.

Para ello, dentro de la clase nube se define el método storeLODs, que recibe un path destino y que se encarga de, a partir de los LODs que hemos calculado en el paso previo y que almacenamos como atributo, determinar los nombres de ficheros LAZ que crearemos, con nombre incremental (ruta absoluta + (LOD0.laz, LOD1.laz, etc.)) para ir insertando en los mismos los valores que ya hemos calculado y que tenemos en memoria RAM. Este método se encarga de definir una cabecera genérica de fichero LAZ con información similar y estándar para todos los ficheros e ir insertando determinados objetos LASPoint que vayamos creando en el fichero que le corresponde. Tendremos un puntero del tipo LASWriter que es el que se encarga de apuntar al fichero correcto almacenado en un array del mismo tipo. También tendremos un array del TDA LASwriteopener, necesario para crear y abrir cada uno de los ficheros definidos. Una vez tenemos definido todo lo relativo a la manipulación de ficheros, tendremos que iterar por cada punto, escribiéndolo en el fichero correcto.

Una vez podemos escribir una nube en disco, con sus diferentes niveles de detalle, y como he explicado antes, después de iterar por todos los nodos y crear los correspondientes LODs, se crean las carpetas correspondiente en la ruta suministrada con el nombre detallado en la sección anterior y, en el interior de ella, almacenar los ficheros.

Una vez este requisito, importante para poder segmentar en 2 partes diferentes y acopladas nuestro proyecto (creación y visualización), estuvo solucionado, se pasó a trabajar con detalles menos importantes, pero aun así necesarios.

En este caso, con la interacción de **VTK**. **En esta herramienta, la interacción se aplica mediante objetos que heredan de la clase vtkInteractorStyle, una clase que implementa la mayoría de controles y que define a su vez una interfaz orientada a eventos para apoyar al vtkRenderWindowInteractor**. Este último elemento, muy importante en la visualización y que nos permite la interacción, implementa diferentes sabores para controlar los periféricos de control de nuestro equipo trabajando conjuntamente con el vtkInteractorStyle.

Después de revisar la documentación y estudiar las diferentes alternativas, la que mejor se amoldaba a nuestro software y que es más intuitiva, tanto para los experimentos, como en versiones futuras y variaciones de este proyecto fue acoplar una instancia del tipo vtkInteractorStyleTrackballCamera a nuestro vtkRenderWindowInteractor. Este objeto permite al usuario manipular interactivamente la escena haciendo zoom, girándose, moviéndose, … con toda la soltura necesaria que necesita la visualización.

Para insertarlo, debemos crear una instancia del mismo y asignarlo al interactor de la ventana con la orden SetInteractorStyle, que recibe un parámetro, el estilo que hayamos definido, que en nuestro caso es el objeto recientemente creado.

2.5.3 Pruebas realizadas

Esta iteración marcaba 3 objetivos bastante importantes: la parametrización de los niveles de detalle, el grabado en disco y la mejora de la interacción. El testeo del primer hito marcado consistía en realizar una serie de pruebas mediante la validación de diferentes niveles de detalle que, o bien debían funcionar adecuadamente, o bien causar un error al estar mal indicados. La segunda parte de las pruebas se centró en estudiar la manera que usaba la biblioteca LASlibz para guardar las nubes en formato .LAZ, ocupando menos espacio que en formato .LAS. Una vez se decidió la manera de almacenar los datos en ficheros disjuntos, se trató de conseguir una robustez de tal manera que ningún tipo de nodo lanzase alguna excepción o error incontrolado. Sin embargo no debemos olvidar que el desacoplamiento de la EEDD residente en memoria a los datos de disco no estaba terminado y, hasta dentro de unas iteraciones, las peticiones se realizan a una

instancia de SparseGrid en vez de a disco, el objetivo final. El último aspecto a probar se basaba en mejorar la experiencia que proporcionaba el hilo gestor de la visualización y de la interacción. Las pruebas de esta parte consistieron en probar una serie de estilos de interacción de VTK y la elección del adecuado para moverse, aumentar o disminuir el zoom o rotar la escena.

Cabe remarcar que **en este momento solo hay incorporado al sistema un hilo encargado de la visualización e interacción**; y la progresividad, apoyada principalmente en un mecanismo que periódicamente realizase comprobaciones de nuevos nodos a mostrar y esconder, de acuerdo a lo establecido en la propuesta de este trabajo, no estaba en proceso de desarrollo aún. **Este elemento**, base del sistema, y el cual proporciona la base sobre la que cimentar todo el software que estamos desarrollando, **se plantea en las siguientes iteraciones inmediatas del proyecto**. Además, la encapsulación de las funcionalidades de los diferentes componentes, a pesar de haberse depurado en esta iteración, no estaba lo suficientemente abstraída, siendo este elemento también objetivo de mejora en las siguientes iteraciones.

2.6 Sexta iteración

2.6.1 Decisiones de diseño

Tras cumplir las tareas previamente marcadas, tuvimos una nueva reunión en la que discutir cuáles serían los siguientes pasos. En la plataforma actual que estamos usando para mostrar las nubes de puntos, VTK**, el hilo que se encarga de crear los puntos y de mostrarlos, permanece en un bucle permanente cuando le indicamos que atienda a las interacciones**. De esta manera, nosotros no podemos acometer el trabajo posterior de poder escalar el sistema hacia la progresividad. Por lo tanto, y tras una investigación por mi parte de las diferentes maneras de solventar esta problemática y tras discutirlo, definimos el objetivo de **implementar un subcomando periódico**.

Este subcomando será **la manera en la que podremos tomar, temporalmente y a intervalos regulares de tiempo, el control del hilo de la visualización e interacción para poder modificar las estructuras y atender a diferentes eventos posteriores**. De esta manera, esta herramienta nos servirá como

marco para mostrar y esconder diferentes nubes en tiempo real, en función de la interacción.

Llegados a esta etapa del proyecto, era importante mantener un código claro y conciso debido al rápido escalado que se prevé y a la implementación de nuevas funcionalidades. Por lo tanto, para la siguiente interacción se propuso **documentar todo el código descrito con el estándar Doxygen, y mantener este tipo de comentarios a lo largo de todo el proyecto conforme se describan nuevas funcionalidades**.

Otro objetivo que nos marcamos fue una **refactorización de la clase nube en otra que solamente representara lo necesario para visualizarla, CloudView**. Anteriormente esta clase almacenaba todos los parámetros relacionados con la nube además de la respectiva parte de visualización; sin embargo, **con la creación de otra clase que representara a la vista de las nubes desacoplaríamos estos 2 conceptos distintos y así el proyecto escalaría cómoda y desacopladamente**.

2.6.2 Detalles de implementación

Después de revisar la documentación de la herramienta de visualización que estamos usando en este proyecto, llegamos a la conclusión de que la solución que mejor se adaptaba a nuestras necesidades era una **implementación de la clase vtkCommand. Esta es la superclase necesaria para la implementación del patrón de diseño observador. En este patrón de diseño, cualquier instancia puede ser observada en cada evento que realice. En nuestro caso, los eventos que realizará el objeto relativo a la vista serán de tiempo a un intervalo regular prefijado**.

Esta clase vtkCommand necesitará de ser heredada por la que nosotros especifiquemos, que **en nuestro proyecto aparece denominada como CommandSubClass** y está implementada en el header file de CloudView (clase que también definimos en esta etapa) ya que es una clase inline y como está ligada a la vista, decidimos colocarla ahí a pesar de que podemos encapsularla en un nuevo fichero cuando sea necesario. El subcomando que acabamos de definir implementa, además de otros métodos no útiles, **la función Execute, que será la que se ejecutará justo después de cada evento**. Será en este segmento de código donde

tendremos que especificar el comportamiento que nosotros queramos implementar en función del estado de determinadas variables relativas a la visualización. Estas variables que nosotros le pasemos deberán aparecer explícitas en una nueva función que definamos y que invoquemos justo después de la creación del objeto. En esta iteración en la que empezamos a abordar este nuevo paradigma para gestionar en tiempo real los cambios de nuestro software solamente le pasamos el objeto vtkRenderer de la vista, mediante el cual podremos acceder a la cámara y ver en tiempo real la nueva posición de la misma. Para poder añadir el evento temporal regular, debemos invocar al método Initialize del vtkRenderWindowInteractor para que se prepare para recibir eventos, crear el propio timer en el mismo interactor con la función CreateRepeatingTimer (el tiempo se pasa por cabecera en ms), y finalmente pasarle el subcomando mediante el método AddObserver, especificando que es un evento de tiempo y la instancia del objeto.

El siguiente objetivo en la lista **era realizar una refactorización de clase Cloud para separar los conceptos de visualización de los intrínsecos a la nube como son los buffers, el centro de la misma o los niveles de detalle**. Para ello, el cambio fue trivial: creé una nueva clase CloudView y me llevé los atributos y funciones que realizaban tareas de visualización a la nueva clase. Después desacoplé las llamadas a los métodos de visualización de la anterior versión de la clase nube tanto en la clase principal como en las relativas para que trabajaran con la instancia de CloudView que acababa de crear.

El último objetivo que nos marcamos en la reunión de esta sexta iteración era realizar una **documentación de todo el código que llevábamos hasta el momento**. **Para ello, utilizaríamos el estándar Doxygen**. Esta herramienta se ha erigido como un estándar de facto para generar documentación de C++, pero también incluye soporte para otros lenguajes como C, C#, Java, Python o VHDL, entre otros. Para ello, utilizamos etiquetas marcadas con el signo @ en comentarios que insertaríamos en los header de cada clase. La lista de etiquetas que usaré será la siguiente:

- @brief: Breve resumen de lo que hace la función
- @param: Descripción del parámetro que le pasamos al método
- @return: Valor y breve descripción de lo que devuelve la función, en el caso de que no sea void.

El proceso de documentación del código se alargó durante varios días y a partir de esta iteración se comentó cada nueva funcionalidad sobre la marcha para así mantener el estilo de programación consistente y uniforme; a la par que evitar en el futuro tener que revisar funciones con un ámbito ligado al momento temporal de la iteración en la que fueron descritas.

2.6.3 Pruebas realizadas

La primera prueba realizada en esta iteración consistió en comprobar que el subcomando periódico realizaba la funcionalidad que nosotros buscábamos sin interferir en demás elementos. Además, se realizaron varios tests buscando el valor adecuado para establecer el tiempo de repetición de este comando. Con un tiempo demasiado pequeño o demasiado grande no seríamos capaces de realizar eficientemente o en tiempo real las tareas designadas en esta parte.

La segunda parte de testeo de la aplicación consistió en corroborar que la separación entre modelo y vista, de las clases Cloud y CloudView, estaba bien hecha y no había errores de ejecución inadvertidos al compilar; esto se realizó mediante el procesamiento de diferentes datasets de diferente tamaño y usando parámetros distintos.

En esta etapa del proyecto se presenta el momento de la integración del sistema productor-consumidor, de una manera estable, en parte conseguido a las continuas refactorizaciones y pruebas a las que estamos sometiendo cada iteración. Será en la séptima iteración donde se abordará este aspecto fundamental, piedra angular del resto de funcionalidad a implementar en el proyecto.

2.7 Séptima iteración

2.7.1 Decisiones de diseño

El desarrollo del proyecto hasta este punto incorpora la funcionalidad necesaria para almacenar y visualizar nubes de puntos, como en cualquier software que implemente un enfoque clásico para mostrar los puntos además del marco que nos permitirá evolucionar hasta la progresividad que buscamos. Por lo tanto, en esta etapa en la que nos encontramos, discutimos que deberíamos **implementar un 2º hilo de ejecución que se encargase de traer desde disco los ficheros adecuados para**

así evitar que el hilo principal invierta tiempo en esta tarea, perjudicando gravemente a la interactividad. **Este enfoque con una segunda hebra nos facilitaría el objetivo de implementar un visualizador progresivo ya que podríamos gestionar la tarea de procesar los puntos necesarios en un instante desde otra parte diferente, agilizando así la ejecución y permitiendo en tiempo real la interacción y evitando una visualización no fluida**.

El planteamiento concurrente de resolución de este problema **conlleva la problemática de tener que considerar la comunicación entre las diferentes ejecuciones mediante el uso de buffers compartidos demás de un control efectivo del acceso a secciones críticas del código en las que solamente una hebra a la vez puede ejecutar un determinado segmento de código**. Para solventar esta problemática del acceso seguro, usaremos el concepto de los semáforos de exclusión mutua que vimos en la asignatura de tercer curso, Sistemas Concurrentes y Distribuidos. Esta herramienta, que acoplaremos dentro de esta funcionalidad, nos permitirá asegurar que solamente uno de los dos hilos accede en un instante a un buffer compartido, evitando así conflictos y la falta de consistencia. Para ello, el esquema general a seguir será el descrito en la figura 16.

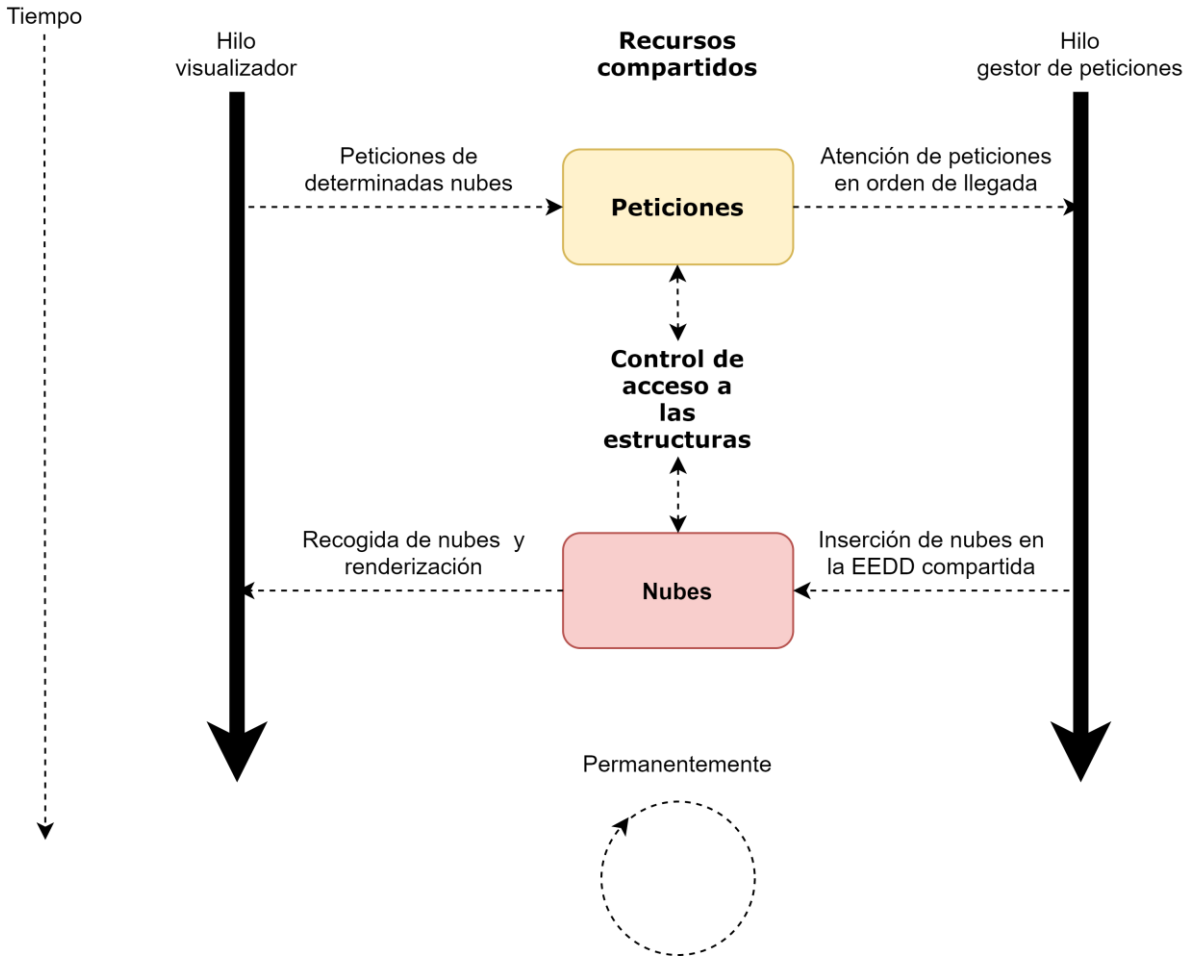

Ilustración 16. Esquema de interacción entre los hilos visualizador y gestor de peticiones

Como podemos ver en el esquema de la ilustración 16, tenemos un bucle que se repite durante toda la ejecución en el que **el hilo visualizador realiza una serie de peticiones** en función del estado de la cámara en la escena, y recoge las nubes que estaban almacenadas en disco. Por su parte, **el hilo gestor de peticiones espera a tener una serie de peticiones de ficheros que se encuentran en disco para poder procesarlos** y pasárselos al buffer compartido en el formato que el hilo visualizador necesita.

Una vez definimos el objetivo principal a explotar en esta iteración**, también nos marcamos una mejora del subcomando que integramos en la sexta iteración** y que realizaba una función trivial de mostrar las coordenadas de la cámara. Ahora **esta rutina temporal debe poder realizar peticiones de ciertos nodos asociados a nubes del grid disperso a la vez que recoger los puntos procesados por el hilo**

que trabaja en disco. De esta manera, acoplamos el hilo adecuadamente con el proyecto y le damos provecho desde la primera etapa en la que lo definimos.

2.7.2 Detalles de implementación

La ejecución de los objetivos previamente definidos constituía un fuerte avance del proyecto, por lo que **esta iteración cobra una importancia máxima hasta la fecha**. Por ello, la elección de las estructuras de datos acordes a nuestras necesidades y la correcta adaptación del segundo hilo planteado, que se encargará de traer unos determinados puntos desde el sistema de almacenamiento masivo, son dos conceptos que deben ser ampliamente estudiados con el objetivo de definirlos e integrarlos de una manera correcta.

Sin embargo, debemos tener en cuenta que no estamos en fases finales del proyecto y que todas nuestras clases son meramente prototipos; prueba de ello es que todavía no está realizada la separación entre la parte de creación de ficheros y la visualización de los mismos: ahora mismo todos los puntos están en un objeto SparseGrid que se encuentra en memoria principal, y el segundo hilo, que en el futuro hará consultas a disco, ahora mismo solo hace peticiones a ese objeto. Este aspecto será solventado en sucesivas iteraciones ya que está localizado y debido a la fuerte encapsulación y continuas refactorizaciones del proyecto, no precisará de grandes cambios.

Volviendo a la lógica relativa al segundo hilo, primero se planteó la manera de simplemente crearlo y ejecutar una función del mismo. En esta universidad, la programación concurrente se afronta desde el lenguaje de programación Java, por lo que para este proyecto se tuvo que revisar la documentación sobre la manera de hacerlo en C++. La opción elegida fue por crear una instancia de clase Thread, la cual se debe crear junto a la llamada a un método de una instancia de una clase diferente, en este caso y en pseudocódigo (suponiendo que estamos en un ámbito dentro de la clase A que constituye la clase principal):

```
Clase B* Instancia de clase B = new Clase B()
std::thread Hilo Clase B = Instancia de clase B -> EjecutaFuncionalidad()
```

Otro aspecto a tener en cuenta es el **acceso a los buffers compartidos**. Si no cuidamos este detalle nuestro software tendrá errores de ejecución que no podremos

detectar trivialmente debido a condiciones de carrera. Una condición de carrera es un problema en programación donde el estado del sistema depende de una secuencia de eventos que se ejecutan en orden arbitrario sobre el mismo recurso compartido. Para solventar esta problemática **se usan los semáforos de exclusión mutua** o, más comúnmente denominados, mutex. Mutex (acrónimo de **Mut**ual **Ex**clusion) es la herramienta más simple para la sincronización de hilos y es ampliamente usada para proteger el acceso a secciones críticas y por lo tanto prevenir condiciones de carrera. Para asegurar el correcto funcionamiento, **cada hilo que quiera acceder a una sección crítica (en nuestro caso, el acceso a un recurso compartido) debe poseer el mutex. Una vez el hilo abandona la sección crítica, suelta el mutex**. **Todos los hilos que quieran acceder a una sección crítica y no tengan el mutex disponible, esperan hasta que se cumpla esta condición antes de proseguir**. En C++ los mutex se implementan mediante instancias de la clase std::mutex. Los hilos adquieren el control con la función lock y lo sueltan invocando al método unlock. El pseudocódigo que planteamos respecto a los 2 hilos, que accederán a los recursos compartidos de una manera adecuada es el siguiente:

Hilo visualizador	Hilo gestor de peticiones
¿Necesito nuevas nubes? : Añado la petición al buffer de peticiones ¿Hay nuevas nubes listas? : Recojo la nube del buffer de nubes, la añado al pipeline de visualización y borro la nube del buffer de nubes Visualizo todas las nubes que almaceno y vuelvo a repetir el proceso	¿Hay peticiones de nubes? : Las traigo de disco y borro la petición ¿La he traído de disco? : La añado al buffer de nubes Vuelvo a repetir el proceso

Tabla 11. Pseudocódigo de los hilos visualizador y gestor de peticiones

Como hemos visto en la tabla 11, debemos cubrir correctamente el acceso a los 2 recursos compartidos con los que estamos trabajando, **adquiriendo el mutex antes de consultar el buffer y soltándolo justo después de su uso**.

Los recursos compartidos si se han definido teóricamente, pero sin embargo a la hora de llevarlos a la práctica, necesitan una correcta representación. Para ello, usaremos 2 vectores de STL y lo que variará serán los tipos de esos vectores-plantilla. En las primeras versiones de estas estructuras, consideré que las peticiones eran

rutas a los ficheros, pero de acuerdo al enfoque Modelo-Vista-Controlador que en sucesivas iteraciones iremos depurando, no se adaptaba. Por lo que el tipo de estas peticiones serían tuplas de 3 valores: (Coordenada X del nodo, Coordenada Y del nodo, Nivel de detalle), y en fases aún más posteriores, **(Coordenada X del nodo, Coordenada Y del nodo, Número de puntos)**. De esta manera, podemos identificar en función de lo que precise el hilo visualizador, los ficheros necesarios.

Con respecto al nodo que almacena la EDD, el tipo de dato abstracto estaba claro, en este caso usaríamos **smart pointers de nubes** (para utilizar memoria dinámica) de las que obtendríamos los puntos para volcarlos en la estructura de datos que volcamos en el pipeline de VTK (el PolyData).

Finalmente, en esta iteración se afrontó la **mejora de la subrutina periódica para que hiciese la tarea definida tanto en la ilustración 15 como en el pseudocódigo de la tabla 11**. Para ello es necesario pasarle los parámetros necesarios en una nueva función para que tenga acceso a ellos e implementar la funcionalidad descrita anteriormente, **prestando especial atención al acceso a secciones críticas y modificando correctamente los recursos compartidos**. En resumidas cuentas, el trabajo que debe de hacer la subrutina consiste en **determinar a cada instante la posición del observador, computar la distancia con respecto a los nodos de la estructura y mostrar los puntos que necesitemos a la par que ocultar los sobrantes**.

2.7.3 Pruebas realizadas

En esta iteración, en la que se solventan errores descritos en las pruebas llevadas a cabo en anteriores etapas, se incluyen elementos muy importantes del proyecto y también **propensos a presentar fallos como es la concurrencia**. La ejecución de varios hilos que comparten estructuras de memoria tiene que estar sustentada en un análisis completo de las posibles problemáticas, como es el caso de este proyecto. Las pruebas por tanto, consistieron primeramente en realizar una serie de ejecuciones con diferentes conjuntos de datos y parámetros y revisar continuamente el estado de los hilos y uso de memoria; con el fin de detectar posibles problemas que pudieran escalar conforme creciese el proyecto.

Aún en esta etapa no estaban desacopladas las 2 fases de preprocesamiento y visualización de este proyecto ni tampoco desaparecen las

nubes cuando no están en nuestro rango de visión, elementos a trabajar en la siguiente iteración.

2.8 Octava iteración

2.8.1 Decisiones de diseño

Como se comenta en la etapa anterior, a pesar de plantear el segundo hilo desde la perspectiva de traer los datos de disco, depositarlos en el buffer correspondiente y finalmente que la vista visualice esos puntos**; la implementación al final de la etapa anterior solo contemplaba que el segundo hilo hiciese las consultas al objeto SparseGrid que se encontraba en memoria, y no a la estructura implícita que ya almacenamos en memoria persistente**. Por lo tanto, el siguiente paso era **desacoplar completamente esta instancia**, que, si bien es necesaria para crear los ficheros, no es necesaria para su visualización. Esta refactorización se encamina al objetivo de tener **2 funcionalidades distintas en nuestro software: una relativa al tratamiento de los puntos y otra a la visualización de los mismos**.

Tratamiento	Visualización
➢ Lectura de ficheros iniciales ➢ Creación del Sparse Grid ➢ División en niveles de detalle ➢ Almacenamiento de los nodos en disco	➢ Visualización fluida ➢ Interacción en tiempo real

Tabla 12. Funcionalidades de los elementos del proyecto

Como podemos ver en la tabla 12, es importante separar adecuadamente las diferentes tareas de tal modo que un cambio en alguna de ellas no afecte a las demás. Este diseño favorecerá tanto a futuras iteraciones, como a remodelaciones del mismo o a nuevas personas que se pudieran sumar al proyecto, ya que se pueden insertar módulos nuevos con gran facilidad y modificar los presentes sin que el cambio afecte en gran medida al resto de componentes del software.

A la hora de realizar la visualización de las nubes, **se necesita información relevante a la estructura de datos espacial que no aparece descrita en los ficheros**.

Para ello, y después de estudiar las alternativas, se estimó oportuno crear un **fichero .csv que nos permitiese guardar esa información al crear el conjunto de datos a visualizar, para una vez terminada la 1ª etapa, poder leer esos componentes intrínsecos a la estructura y que a la vez son necesarios para poder realizar la visualización**. Lo que almacenamos en este fichero .csv son el **número de niveles de detalle, el tamaño de las celdas y el centro de la caja envolvente**. Este último parámetro no tiene ningún uso en la versión final del proyecto.

Como he mencionado varios nuevos aspectos, haré una breve reseña de los dos últimos. Un fichero de extensión csv (del inglés comma-separated values) es un tipo de documento en el que los valores dentro de una fila se separan por comas (o punto y coma a veces, aunque en este caso se usa la forma anterior) y las columnas por saltos de línea. Con estas sencillas normas podemos escribir ficheros con una estructura concreta que posteriormente podremos leer sin mayor dificultad que conocer el orden en el que se escribió el documento.

Por otro lado, el termino bounding box es un mecanismo para describir un área particular de un mapa mediante el almacenamiento de las longitudes y latitudes mínima y máxima. En este caso, al estar trabajando con un entorno tridimensional, nosotros trabajamos con la X, Y, Z mínima y máxima. En la siguiente figura 17 podemos ver un ejemplo de una caja envolvente en 3 dimensiones aplicada a un cilindro.

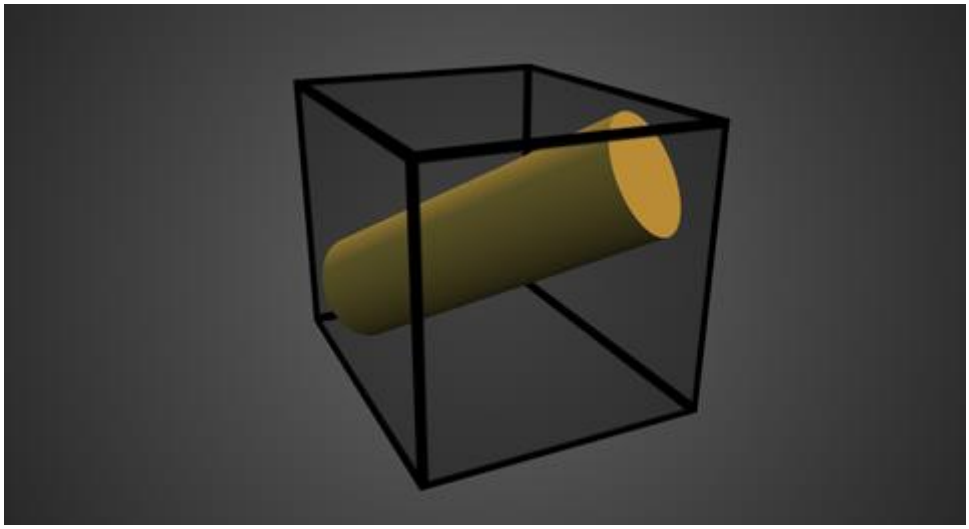

Ilustración 17. Ejemplo de caja envolvente de un objeto tridimensional

Para obtener el centro de la caja envolvente se realiza la siguiente operación.

$$(X, Y, Z\ centrales) = (\frac{X\min + X\ max}{2} + \frac{Y\min + Y\ max}{2} + \frac{Z\min + Z\ max}{2})$$

2.8.2 Detalles de implementación

Debido a la fuerte encapsulación que hemos tenido en cuenta desde las primeras fases del desarrollo, la **separación de las dos funcionalidades** resulta trivial. **Es necesario definir un parámetro** que se puede definir en el propio código fuente o que se introducir como parámetro en la llamada al programa que permita elegir qué funcionalidad elegir. **En función de este parámetro, se realizará un tratamiento de los ficheros que se encuentran en una ruta para prepararlos para la visualización progresiva, o se realizará directamente esta visualización**. Es importante una clara separación y orden del código en esta parte para evitar errores cuya depuración resulte problemática.

Con respecto al segundo objetivo marcado, cuando queremos visualizar un dataset se necesitan datos adicionales que antes ya teníamos en memoria principal pero que al almacenar en disco no se presentan en los puntos. Para ello, **se hace uso del fichero de extensión csv generado en la fase preprocesamiento**. Esta breve iteración está centrada en lograr el **desacoplamiento entre el grid de memoria para pasarlo a disco**, para así conseguir el objetivo de hacer las peticiones de nodos adecuadamente al sistema de ficheros y no a datos previamente cargados en memoria principal. El enfoque anterior es inviable en datasets de gran escala, los cuales buscamos visualizar en este trabajo.

2.8.3 Pruebas realizadas

Esta corta iteración consistente en separar las 2 fases de las que consta nuestro proyecto tuvo una fase experimentación muy corta debida a su naturaleza. Los tests realizados se basaron simplemente en comprobar que después de la abstracción de los componentes, el funcionamiento fuese similar al de la iteración anterior. Debemos recordar que todavía persisten los mismos problemas de la iteración anterior, los cuales motivarán parcialmente las últimas 3 iteraciones.

2.9 Novena iteración

2.9.1 Decisiones de diseño

Después de una fase de desarrollo rápida con varias iteraciones que provocaron varias refactorizaciones en el proyecto y el desacoplamiento de funcionalidades, la ingente cantidad de código desordenado y típica de solventar errores sin prestar atención al diseño, la estructura general de clases distaba mucho de seguir cualquier patrón de diseño estándar. Por ello, y con el objetivo de escalar el proyecto en versiones futuras, en la reunión periódica que tuvimos decidimos acordar una **nueva refactorización más encaminada a orientar el código fuente al paradigma Modelo-vista-controlador**.

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software que tiene la función de separar los datos, de la presentación y de la lógica de la aplicación, respectivamente. Se persigue dividir todos los componentes de una aplicación en las tres categorías previamente definidas de tal manera que el intercambio de información entre las diferentes partes siga un esquema general. Este patrón es ampliamente usado en gran cantidad de aplicaciones y de todos los ámbitos. Dentro del ámbito de esta titulación, es visto en la asignatura de tercer curso Desarrollo de software para dispositivos móviles.

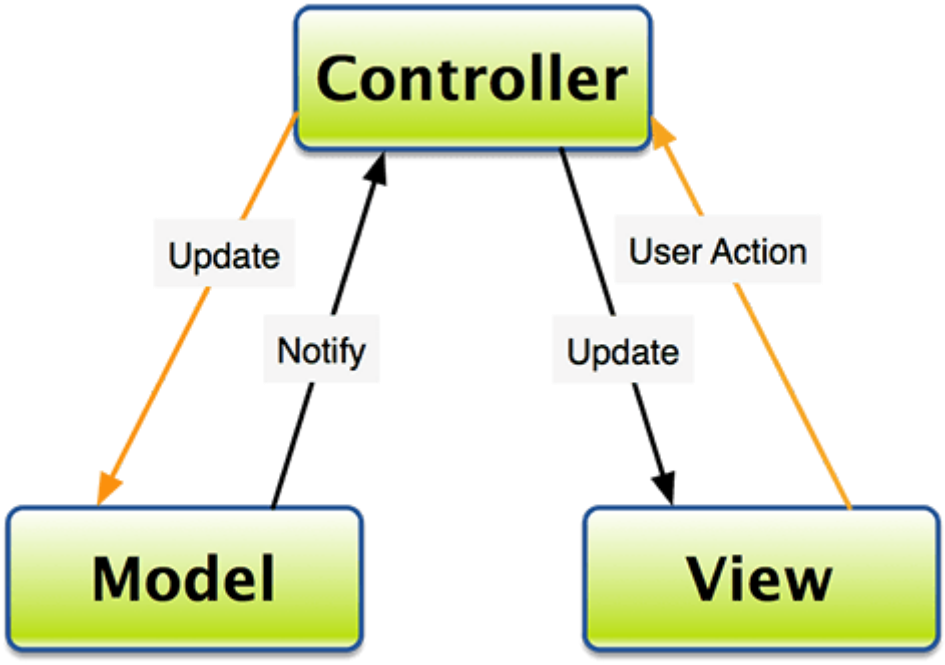

Ilustración 18. Esquema del patrón de diseño Modelo-vista-controlador

De acuerdo a la ilustración 18, los diferentes componentes de este patrón de arquitectura software se encargan de una parte de la funcionalidad muy concreta y se comunican entre sí de una manera predefinida.

- **Modelo**: Se encarga de notificar cambios en el mismo al controlador. Es la parte en la que se almacenan los datos.
- **Vista**: Se encarga de notificar las interacciones de los usuarios al controlador. Es la parte que se encarga de mostrar los datos y de proveer la interacción.
- **Controlador**: Se encarga de gestionar el intercambio de información entre la vista y el modelo. Es la parte lógica de la aplicación y el centro de comunicaciones entre los componentes.

Para implementar este patrón de diseño es necesario definir qué clases serán de cada componente. Para ello, **se decidió crear las clases DatasetView, donde se encapsulase todo el segmento de código relativo a la interacción del usuario y parte final de la visualización, y CloudView, que representa a cada nodo de la nube**. Esta sería el elemento vista de esta arquitectura. También se optó por crear la **clase Dataset, elemento que serviría de intermediario entre vista y modelo, es decir, como controlador**. El **modelo, por lo tanto, y de acuerdo a nuestro esquema, sería la clase relativa al segundo hilo que atiende peticiones acudiendo a disco, la clase PetitionsThread; pudiendo considerar también la clase Cloud como modelo**.

Terminado todo lo relativo a la implementación de este patrón de diseño tan conocido, en la misma reunión decidimos varias estrategias para afrontar una progresividad sostenible en el tiempo. **Hasta el momento, todas las nubes cargadas desde disco a memoria RAM, permanecían en pantalla a pesar de que nos deplazásemos hacia zonas lejanas o incluso, si nos acercábamos a la nube, y esta nos cargaba más puntos, al alejarnos estos no desaparecerían**.

Era muy necesaria una solución a esa problemática ya que conforme nos deplazemos a lo largo de la nube o hagamos una serie de interacciones continuas en el tiempo, el rendimiento del sistema se deterioraría enormemente. Para ello, llegamos

a la conclusión de que era necesario **encapsular a cada uno de los subnodos** que cargamos dentro de una celda en una **nueva estructura de actores**, que son los elementos que visualizamos. Si ponemos el ejemplo de que visualizamos una única celda, con 5 niveles de detalle, si estamos lejos de ella solo veremos el nivel de detalle 0, y conforme nos vayamos acercando iremos realizando las peticiones de nodos pertinentes en el buffer correspondiente y se cargarán los niveles de detalle 1, 2 y sucesivos, los cuales, al ser disjuntos, complementan la celda pero son puntos independientes. De esta manera, a la hora de alejarnos de la nube, no trataremos al nodo en sí, sino a cada uno de sus componentes, pudiendo nosotros quitar de la visualización los actores que representan a los niveles de detalle 4, 3, 2, 1 y 0 en este orden.

De acuerdo a la ilustración 19, en la que se presenta un ejemplo del funcionamiento de esta estructura de actores, se puede explicar brevemente qué es lo que hace. En el primer cuadrado, a una distancia lejana, solo se carga el LOD0, si nos acercamos (segundo cuadrado) se añade otro nivel de detalle al ya existente y al alejarnos el nuevo nivel de detalle desaparece, dejando solamente el original.

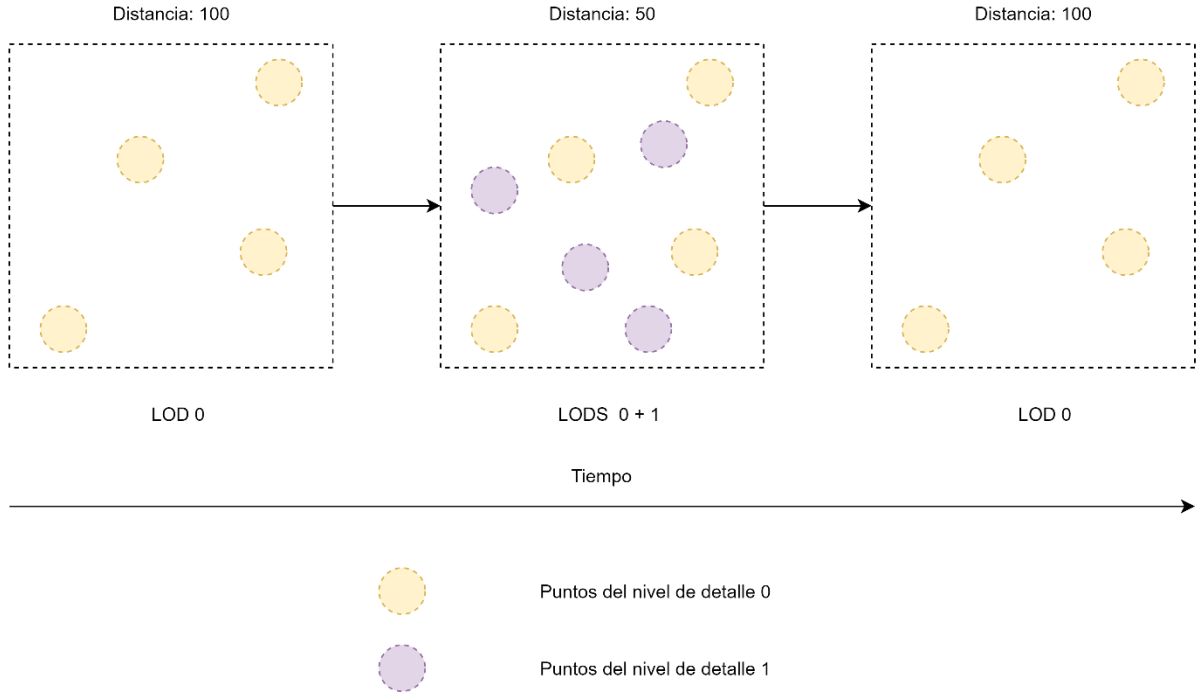

Ilustración 19. Ejemplo del funcionamiento de la estructura de actores

Los detalles de implementación y acoplación en el código de esta estructura serán descritos en el siguiente epígrafe al igual que los respectivos detalles de implementación del patrón MVC.

2.9.2 Detalles de implementación

En esta iteración debemos **refactorizar por enésima vez** nuestro código, en este caso **orientado a adaptar nuestra implementación al patrón de diseño Modelo-Vista-Controlador** comentado anteriormente.

De acuerdo a esta arquitectura, debemos dividir los componentes que tenemos en función de su finalidad. El modelo será el elemento que trabaja a más bajo nivel, el que se encarga de traer los datos crudos desde disco, cuando el controlador se lo pida. Para ello, la clase del segundo hilo de peticiones, presentada unas iteraciones atrás, será la encargada.

Primeramente, **en la clase PetitionsThread se define un índice espacial que se encarga de recorrer el directorio en el que se encuentran los nodos del sparse grid en el inicio de la ejecución**. **El índice espacial consiste en una lista que almacena que nodos existen y cuáles no, para que, a la hora de tratar peticiones de la vista, no se desperdicien recursos atendiendo peticiones de nodos inexistentes**. Posteriormente, en esta clase, la cual recibe las peticiones en forma de terna que será explicada en la próxima iteración, se encargará de acceder a los datos que residen de disco y devolverlos en el buffer correspondiente al controlador, que, a su vez, lo comunicará a la vista.

Seguidamente, la clase Dataset ejerce de controlador en este proyecto**. La funcionalidad de la misma consiste en comunicar modelo con vista mediante el intercambio correcto de información, para ello, comparte punteros a las estructuras de datos de intercambio de nodos y peticiones entre vista y modelo**. De esta manera, puede estar en medio de la comunicación gestionándola de tal manera que el modelo y la vista no están conectados directamente.

Finalmente, la clase DatasetView es la encargada de funcionar como vista. **Esta clase concentra toda la visualización, y por lo tanto todo el segmento de código de VTK está incluido en esta sección**. También recoge la interacción mediante teclado y ratón, explicada en iteraciones anteriores. **Esta clase alberga la estructura de actores como un mapa en el que la clave es el par de coordenadas**

X, Y del nodo del objeto CloudView. **Los objetos CloudView son la abstracción necesaria para la visualización de las nubes de puntos**.

La estructura de actores consiste en un grid disperso de objetos CloudView. Estos objetos albergan como atributos los determinados puntos correspondientes a cada nodo. **Serán las mismas instancias las que se visualizan y que se van actualizando en función de la posición del observador**. La manera en la que se actualizan los actores de cada celda será debatida en el siguiente epígrafe.

2.9.3 Pruebas realizadas

Al igual que todas las pruebas relativas a la refactorización de segmentos de código, esta nueva encapsulación implicó repetir el mismo procedimiento llevado a cabo en estos casos: una comprobación de que los resultados de salida obtenidos en la anterior iteración eran similares a los obtenidos en esta. No se realizaron más pruebas al no incluirse funcionalidad nueva, algo que si ocurrió en las dos últimas iteraciones.

2.10 Décima iteración

2.10.1 Decisiones de diseño

En esta a priori última iteración importante del proyecto se llega con un bagaje de hitos alcanzados importante, sin embargo, **hay algunos aspectos recogidos a lo largo de las pruebas realizadas que deben ser finalmente tenidos en cuenta para depurar el proyecto y así lograr el objetivo de conseguir un software visualizador de nubes de puntos masivas, progresivo y eficiente**.

Con respecto al enfoque principal de este proyecto, la progresividad, se nos planteaba diferentes alternativas a la hora de definir una métrica que permitiese determinar qué conjunto de puntos se cargaría y cuál no. Tras estudiar las diferentes posibilidades, acordamos implementar **una variación de la función de distribución normal que, en función de la distancia del observador a la celda, determinase qué conjunto de puntos debe presentar un nodo en la pantalla**. A continuación,

aparece detalladamente el desarrollo de la métrica usada en la fase de visualización de este proyecto.

$$Partiendo\ de\ la\ función\ de\ distribución\ normal \rightarrow f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$Y\ usando\ una\ varianza\ \ \sigma^2 = \frac{1}{2\pi}\ obtenemos:$$

$$Función\ de\ distribución\ implementada \rightarrow\ \boldsymbol{\varphi(x) =\ e^{-\pi x^2}}$$

Debemos tener en cuenta que en esta función de distribución para el rango de x en el cual obtenemos valores mayores de 0 es muy pequeño. Entonces lo que hacemos es **comprimir el rango de distancias con las que queremos trabajar al intervalo [0,1], normalizándolas** mediante una división de la distancia actual al nodo entre una distancia máxima de visualización predefinida en el código. Será este valor normalizado el que reciba la función de distribución. De tal modo, si esta distancia máxima fuera de 1000, una celda que está a una distancia 500 del observador obtendría un valor x = 500 / 1000 = 0.5, obteniendo una probabilidad en nuestra función de distribución $\varphi(x)$ de 0.455. Sin embargo **si una celda está a una distancia de 20, el valor de X será de 0.02 y la probabilidad será de 0.998**. **En el otro extremo, si la celda está a 4000, x = 4 y la probabilidad ≈ 0**.

De acuerdo a la métrica definida en este mismo epígrafe; **el funcionamiento de la progresividad usará este mismo enfoque**. Primeramente, **se obtiene una lista de los nodos cercanos a la posición del observador en la escena**. Esta lista depende de la distancia del observador a los nodos, por lo **que conforme más cerca estemos de ellos, menos vecinos del nodo principal tendremos que recorrer y conforme más lejos, más vecindario deberá ser tenido en cuenta**. Un ejemplo de cómo se obtiene esta lista, en función de la distancia de la cámara, lo podemos ver en

la ilustración 20, donde el nodo azul es el apuntado por la cámara, y los coloreados, el vecindario a tener en cuenta.

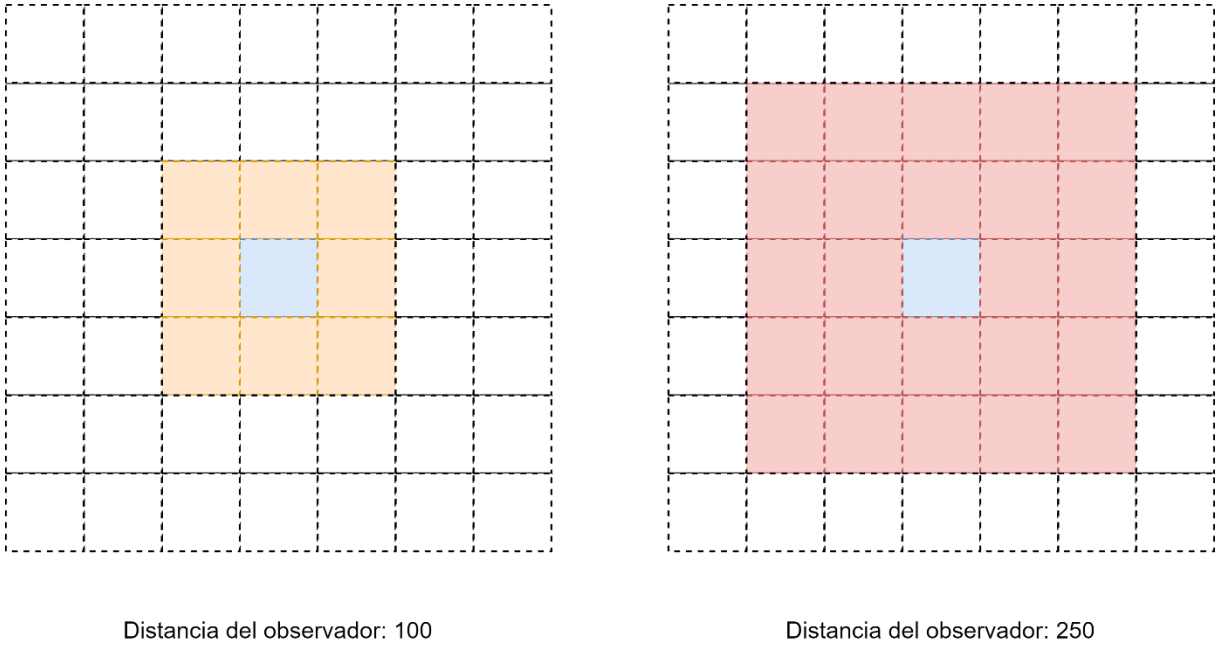

Ilustración 20. Ejemplo de obtención del vecindario de un nodo

Una vez definido los nodos que vamos a visualizar, debemos tener en cuenta que **no todos ellos necesitan del mismo número de puntos**, y para ello, hacemos **uso de la métrica** previamente definida. **A cada número determinado de frames, recorremos el vecindario actual y calculamos el presupuesto de puntos que debe tener cada nodo considerado**. **El cálculo de este presupuesto de puntos se obtiene multiplicando el valor obtenido de la función de distribución, acorde a la distancia al nodo, por un límite de puntos, un parámetro que indica una cota superior al número de puntos que puede tener un nodo.** Este presupuesto de puntos (porcentaje asignado por la función φ(x) multiplicado por un presupuesto de puntos predefinidos) será clave para realizar una petición o no. **Solo si el presupuesto de puntos es mayor que el que había antes, se realiza la petición**. **Aparte, para ocultar los nodos menos visibles desde la posición actual, se itera por todos los nodos y se comprueba a intervalos regulares de tiempo si están fuera del vecindario del nodo actual**, en un sentido opuesto al de la ilustración 19 pero con la misma base.

Se plantea en este proyecto la posibilidad, aparte de esconder los nodos, de borrarlos de memoria RAM y volverlos a traer posteriormente desde disco; sin embargo, **este enfoque de borrarlos no ha sido abordado**, ya que normalmente al visualizar una nube de puntos se suele volver a zonas exploradas, y si solo se esconden y no se borran, se gana rendimiento. El motivo principal de no introducir este cambio, evolución natural del proyecto, es que **el rendimiento no se ve afectado aun cuando tenemos muchos datos cargados siempre y cuando los nodos no estén siendo visualizados**; aunque en el caso de que se quiera exportar a otra plataforma no sería mala idea plantear un sistema que eliminase de memoria a los nodos **de acuerdo a unas reglas distintas** a de ocultarlos.

2.10.2 Detalles de implementación

Si bien las decisiones de diseño son complejas en comparación a las anteriores iteraciones, la implementación de tales objetivos, en gran parte debido a la continua refactorización del proyecto, resultó trivial.

Cabe remarcar que la implementación de la funcionalidad de calcular el vecindario y el cálculo del presupuesto de puntos por nodos en función de la distancia del observador al nodo recaía sobre la vista, al ser esta la encargada de realizar las peticiones. Un detalle de la implementación es que la operación de calcular la distancia al cuadrado no se realiza con la función std::pow, sino con una multiplicación de ella misma por ella misma debido a que **la eficiencia de esta última instrucción es mayor que la de usar la función de calcular potencia con un número pequeño como es 2**. Otro detalle importante es que **el radio que determina el tamaño del vecindario sigue intervalos fijos de distancias prefijados para obtener valores** y que han sido obtenidos empíricamente después de realizar diferentes ejecuciones.

2.10.3 Pruebas realizadas

Las pruebas relativas a esta iteración consistieron en **demostrar que la métrica que se plantea funciona adecuadamente**, devolviendo más puntos a los nodos más cercanos a la cámara y menos a sus vecinos. También las pruebas se encargaron de corroborar que **el número de frames parametrizado para hacer peticiones de nodos y esconder a los que están en pantalla pero no en zonas de interés fuese el adecuado**. En esta penúltima iteración **el rendimiento no estaba lo**

suficientemente exprimido ni en la lectura de ficheros ni en la visualización como para terminar el desarrollo, por lo que, con el objetivo de solventar esta última problemática, se decidió realizar una **última iteración**.

2.11 Undécima iteración

2.11.1 Decisiones de diseño

Después del parón obligatorio debido a la crisis originada por el COVID-19, el proyecto tuvo un parón en su desarrollo software de aproximadamente un mes. A pesar de no poder continuar en el laboratorio, se pudo continuar con el trabajo de manera remota.

Esta última fase de desarrollo, con todo el software integrado y testeado, se dedicó a **mejorar dos elementos problemáticos**.

El primero de ellos consistía en la lectura de ficheros, que era altamente ineficiente y ocasionante de errores por falta de memoria RAM.

El segundo se basaba en un rendimiento deficiente en la visualización debido al constante intercambio de información entre bufferes y un cuello de botella presente por el pipeline de VTK que no podía paralelizarse.

Una vez se arreglaron estos elementos se procedió a la experimentación.

2.11.2 Detalles de implementación

El error en la lectura de los ficheros se debía a un volcado de la instancia de la clase Reader a un objeto Cloud y de ahí al SparseGrid; pero **ese paso intermedio era innecesario**. Para solucionar ese error, **la EEDD se rellena ahora secuencialmente de tal modo que copias innecesarias no son creadas entre medias**. Los cambios a nivel de código solo implicaron cambios en los parámetros recibidos por las funciones y el error se detectó debido a que el segundo ordenador en el que se realizó el desarrollo disponía de menos memoria RAM.

El segundo problema que se solucionó consistía en un **muy bajo rendimiento por el cuello de botella inherente al pipeline de VTK al procesar puntos**.

Para mejorar eso, se planteó el **uso de un evento que permitiese parar el intercambio de datos entre el hilo visualizador y el atendedor de peticiones**. Este evento se activaría con la tecla espacio. Mediante el uso de las subclases que se encargan de controlar la interacción se añadió un parámetro que permitiese activar o desactivar la realización de peticiones y la recogida de los puntos procedentes de disco.

Otro cambio de menor importancia implementado y que mejora el rendimiento levemente consiste en renderizar las peticiones ya recogidas procedentes de disco una a una en frames consecutivos en vez de todas a la vez.

2.11.3 Pruebas realizadas

Las pruebas relativas a la última iteración fueron simples de realizar debido a la localización de los errores y al conocimiento de que algún error habría si no se presentaba ningún cambio con respecto al resultado de la iteración anterior. Al ser esta la última etapa del proyecto se llega con una experiencia previa sobre el mismo que ayuda a identificar problemas y resolverlos de manera rápida. **El conjunto de tests realizados consistieron en comprobar la correcta integración de la lectura secuencial y del evento que paraliza o reanuda el intercambio entre buffers**.

2.12 Pruebas finales

Al seguir una metodología de desarrollo ágil en este proyecto, cada iteración tenía marcado uno o varios objetivos que debían ser testeados. Cada uno de estos objetivos, por regla general, era complementario a los anteriores y no constituía un objetivo general final del proyecto por sí mismo, por lo que la realización de pruebas en esa etapa concreta consistía en un simple testeo que verificase el acoplamiento correcto en el software conjunto.

Por ello, **todas las pruebas que verifican la eficacia, eficiencia y robustez del sistema en su versión final se recogen en el siguiente punto y sucesivos**; que se centran en realizar una serie de experimentos reales para poner a prueba el sistema desarrollado y ser capaces de redactar unas conclusiones.

3 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

En este epígrafe se detallarán los resultados obtenidos de este software, de acuerdo a su rendimiento después de experimentar con varios parámetros del programa que permitirán finalmente abrir un debate y discutir sobre mejoras en futuras iteraciones del proyecto.

En las 2 siguientes secciones se obtendrán unas conclusiones en base al desempeño de este software con un conjunto de datos de prueba, un dataset usado en la práctica por el grupo de investigación en el que se realizó este proyecto; un dataset mobile mapping de gran resolución, que abarca ubicaciones características del centro de Madrid (la Puerta de Alcalá, el Ayuntamiento, la Fuente de Cibeles, el Edificio Metrópolis…). Este dataset está compuesto por 7 ficheros disjuntos que aúnan entre sí 200 millones de puntos, ocupando en disco más de 6GB.

Este conjunto de datos fue elegido debido a sus características volumétricas, prácticas y visuales. De todos modos, **este no ha sido el único dataset estudiado y experimentado durante las iteraciones del proyecto con el objetivo de evaluar la robustez y adaptación del software que enmarca este informe**.

3.1 Experimentaciones y pruebas

Las experimentaciones que se llevarán a cabo en esta sección se centrarán en evaluar, con respecto al dataset previamente planteado, el rendimiento en cada una de las 2 fases (preprocesamiento y visualización) del sistema planteado en este trabajo.

En la fase de preprocesamiento se medirá el tiempo que tarda el equipo en generar la estructura de ficheros procesados necesaria para poder visualizar las nubes de puntos en tiempo real; y en la fase de visualización se medirá la tasa de refresco y el rendimiento del procesamiento de las peticiones. Esta última medida es subjetiva y presenta un problema en la biblioteca de visualización que estamos usando. VTK no permite una paralelización de un proceso necesario para poder visualizar un conjunto de nubes (iterar por dos bucles que recorren posiciones y colores), esto ocasiona que el hilo de visualización experimente un retardo en

determinadas ocasiones, provocando que la interacción se pare durante un instante, para después continuar.

Al ser este software un prototipo, sirve para validar la idea de un sistema progresivo y verificar su funcionamiento con conjuntos de datos a gran escala; sin embargo, para futuras versiones del proyecto, se plantea desacoplar la parte de VTK (de una manera sencilla, de acuerdo a la constante encapsulación del código) y utilizar OpenGL. Esta biblioteca de más bajo nivel nos permite trabajar directamente con los buffers de datos en GPU; algo necesario en un sistema perfectamente optimizado y que nos ofrecería una eficiencia superior a la actual. Sin embargo, VTK nos ha permitido abstraernos de gran cantidad de tareas necesarias en la visualización y también explora la posibilidad de que, a mayor alto nivel de abstracción, menor eficiencia a bajo nivel nos encontramos.

Los parámetros usados para realizar las experimentaciones serán los indicados en la tabla 13:

Niveles de detalle	3	4	5
Tamaños de celda	25	50	100
Presupuesto de puntos	100.000	250.000	500.000

Tabla 13. Parámetros usados en la experimentación

Para cada una de estas subdivisiones en niveles de detalle, aplicaremos los 3 tamaños de celda y los 3 presupuestos de puntos. Este último parámetro solo influye en la visualización, por lo que tendremos 3*3 = 9 pruebas de preprocesamiento, con medidas de tiempos, y 3 tablas de visualización vinculadas a cada una de las anteriores, haciendo un total de 3*9 = 27 pruebas de visualización; por lo que redactaremos nuestras conclusiones en base a 9 pruebas de procesamiento +27 pruebas de visualización = 36 pruebas; repartidas en 9 tandas de 4 ejecuciones cada una. Se denomina tanda a la realización de un nuevo preprocesamiento con sus 3 visualizaciones correspondientes.

En este software aparecen también parametrizados diversos valores como el tamaño del vecindario o frecuencia de peticiones a disco que se mantienen constantes para toda la experimentación.

Estas 36 pruebas son las que se recogen, detalladamente de acuerdo a cada parámetro usado en la tabla 13, a continuación. Aparecen experimentos de diferentes colores de acuerdo a la naturaleza del mismo.

Índice de experimento	Tipo de experimento	Niveles de detalle	Tamaños de celda	Presupuesto de puntos
1	Preprocesamiento	3	25	*
2	Visualización	3	25	100.000
3	Visualización	3	25	250.000
4	Visualización	3	25	500.000
5	Preprocesamiento	3	50	*
6	Visualización	3	50	100.000
7	Visualización	3	50	250.000
8	Visualización	3	50	500.000
9	Preprocesamiento	3	100	*
10	Visualización	3	100	100.000
11	Visualización	3	100	250.000
12	Visualización	3	100	500.000
13	Preprocesamiento	4	25	*
14	Visualización	4	25	100.000
15	Visualización	4	25	250.000
16	Visualización	4	25	500.000
17	Preprocesamiento	4	50	*
18	Visualización	4	50	100.000
19	Visualización	4	50	250.000
20	Visualización	4	50	500.000
21	Preprocesamiento	4	100	*

22	Visualización	4	100	100.000
23	Visualización	4	100	250.000
24	Visualización	4	100	500.000
25	Preprocesamiento	5	25	*
26	Visualización	5	25	100.000
27	Visualización	5	25	250.000
28	Visualización	5	25	500.000
29	Preprocesamiento	5	50	*
30	Visualización	5	50	100.000
31	Visualización	5	50	250.000
32	Visualización	5	50	500.000
33	Preprocesamiento	5	100	*
34	Visualización	5	100	100.000
35	Visualización	5	100	250.000
36	Visualización	5	100	500.000

Tabla 14. Definición de los experimentos a realizar

**: Significa que en la fase de preprocesamiento ese parámetro no influye; ya que solo es relevante en la visualización*

Estos experimentos se han realizado en un ordenador menos potente que con el que se desarrolló. Las características más importantes de las que dispone el equipo de experimentación una tarjeta gráfica AMD Radeon R7 265 Dual-X de 2 GB GDDR5; 8GB de memoria RAM Kingston HyperX DDR3, un procesador Intel Core i4-4440 @ 3.10 GHz y un SSD Crucial MX500 de 250 GB de almacenamiento. Este equipo tiene unas propiedades inferiores a los ordenadores que nos podemos encontrar actualmente en el mercado y es por ello que debemos tener en cuenta este factor en la redacción de los resultados, a priori, prometedores y a la altura de este primer prototipo que sirve para validar la idea del uso de una estructura de datos dispersa como solución para la visualización de conjuntos de datos LiDAR masivos.

A continuación, se adjuntan los resultados obtenidos en la fase de experimentación, de acuerdo a las pruebas definidas en la tabla 14.

Preprocesamiento	
Niveles de detalle	3
Tamaño de celda	25
Tiempo de preprocesamiento	9 minutos 50 segundos

Tabla 15. Resultados obtenidos del experimento 1

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	47
Rendimiento del procesamiento de peticiones	Muy alto

Tabla 16. Resultados obtenidos del experimento 2

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	28
Rendimiento del procesamiento de peticiones	Alto-muy alto

Tabla 17. Resultados obtenidos del experimento 3

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	23
Rendimiento del procesamiento de peticiones	Medio

Tabla 18. Resultados obtenidos del experimento 4

De esta manera terminamos la primera tanda de ejecuciones.

Preprocesamiento	
Niveles de detalle	3
Tamaño de celda	50
Tiempo de preprocesamiento	9 minutos 30 segundos

Tabla 19. Resultados obtenidos del experimento 5

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	62
Rendimiento del procesamiento de peticiones	Medio-alto

Tabla 20. Resultados obtenidos del experimento 6

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	50
Rendimiento del procesamiento de peticiones	Media

Tabla 21. Resultados obtenidos del experimento 7

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	43
Rendimiento del procesamiento de peticiones	Medio-bajo

Tabla 22. Resultados obtenidos del experimento 8

Estas son las 4 tablas obtenidas después de la segunda tanda de ejecuciones.

Preprocesamiento	
Niveles de detalle	3
Tamaño de celda	100
Tiempo de preprocesamiento	9 minutos 38 segundos

Tabla 23. Resultados obtenidos del experimento 9

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	20
Rendimiento del procesamiento de peticiones	Bajo

Tabla 24. Resultados obtenidos del experimento 10

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	12
Rendimiento del procesamiento de peticiones	Bajo-muy bajo

Tabla 25. Resultados obtenidos del experimento 11

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	9
Rendimiento del procesamiento de peticiones	Muy bajo

Tabla 26. Resultados obtenidos del experimento 12

Después de realizar la tercera tanda de ejecuciones, los resultados obtenidos son los presentados.

Preprocesamiento	
Niveles de detalle	4
Tamaño de celda	25
Tiempo de preprocesamiento	9 minutos 54 segundos

Tabla 27. Resultados obtenidos del experimento 13

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	19
Rendimiento del procesamiento de peticiones	Muy alto

Tabla 28. Resultados obtenidos del experimento 14

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	18
Rendimiento del procesamiento de peticiones	Alto

Tabla 29. Resultados obtenidos del experimento 15

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	15
Rendimiento del procesamiento de peticiones	Medio

Tabla 30. Resultados obtenidos del experimento 16

Al finalizar la cuarta tanda de ejecuciones podemos ver los resultados obtenidos en las tablas 27 a 30.

Preprocesamiento	
Niveles de detalle	4
Tamaño de celda	50
Tiempo de preprocesamiento	9 minutos 24 segundos

Tabla 31. Resultados obtenidos del experimento 17

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	62
Rendimiento del procesamiento de peticiones	Medio- alto

Tabla 32. Resultados obtenidos del experimento 18

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	58
Rendimiento del procesamiento de peticiones	Medio-alto

Tabla 33. Resultados obtenidos del experimento 19

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	55
Rendimiento del procesamiento de peticiones	Medio

Tabla 34. Resultados obtenidos del experimento 20

La quinta tanda de ejecuciones recoge estos 4 experimentos presentados en esta página, realizados de acuerdo a los parámetros establecidos en la tabla 14.

Preprocesamiento	
Niveles de detalle	4
Tamaño de celda	100
Tiempo de preprocesamiento	9 minutos 31 segundos

Tabla 35. Resultados obtenidos del experimento 21

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	38
Rendimiento del procesamiento de peticiones	Medio-bajo

Tabla 36. Resultados obtenidos del experimento 22

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	22
Rendimiento del procesamiento de peticiones	Bajo

Tabla 37. Resultados obtenidos del experimento 23

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	14
Rendimiento del procesamiento de peticiones	Bajo-muy bajo

Tabla 38. Resultados obtenidos del experimento 24

En la tanda de pruebas Nº6, que recogen las tablas 35-38 se realizan los experimentos con un preprocesamiento del dataset con 4 niveles de detalle y un tamaño de celda de 100.

Preprocesamiento	
Niveles de detalle	5
Tamaño de celda	25
Tiempo de preprocesamiento	9 minutos 32 segundos

Tabla 39. Resultados obtenidos del experimento 25

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	32
Rendimiento del procesamiento de peticiones	Muy alto

Tabla 40. Resultados obtenidos del experimento 26

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	22
Rendimiento del procesamiento de peticiones	Alto-muy alto

Tabla 41. Resultados obtenidos del experimento 27

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	17
Rendimiento del procesamiento de peticiones	Medio-alto

Tabla 42. Resultados obtenidos del experimento 28

En esta séptima tanda de experimentos podemos ver los resultados obtenidos desde el experimento 25 al 28, ambos inclusive.

Preprocesamiento	
Niveles de detalle	5
Tamaño de celda	50
Tiempo de preprocesamiento	9 minutos 2 segundos

Tabla 43. Resultados obtenidos del experimento 29

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	62
Rendimiento del procesamiento de peticiones	Medio-alto

Tabla 44. Resultados obtenidos del experimento 30

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	55
Rendimiento del procesamiento de peticiones	Medio-alto

Tabla 45. Resultados obtenidos del experimento 31

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	50
Rendimiento del procesamiento de peticiones	Medio

Tabla 46. Resultados obtenidos del experimento 32

En esta página se adjuntan los resultados de los experimentos 29 al 32, pertenecientes a la octava tanda

Preprocesamiento	
Niveles de detalle	5
Tamaño de celda	100
Tiempo de preprocesamiento	9 minutos

Tabla 47. Resultados obtenidos del experimento 33

Visualización	
Presupuesto de puntos	100.000
Tasa de refresco media (fps)	25
Rendimiento del procesamiento de peticiones	Medio-bajo

Tabla 48. Resultados obtenidos del experimento 34

Visualización	
Presupuesto de puntos	250.000
Tasa de refresco media (fps)	15
Rendimiento del procesamiento de peticiones	Bajo

Tabla 49. Resultados obtenidos del experimento 35

Visualización	
Presupuesto de puntos	500.000
Tasa de refresco media (fps)	12
Rendimiento del procesamiento de peticiones	Bajo

Tabla 50. Resultados obtenidos del experimento 36

Con los experimentos 33 a 36, pertenecientes a la novena tanda, se finaliza el epígrafe de experimentaciones y pruebas.

El análisis y conclusiones serán temas a comentar en el siguiente punto; resultados y discusión.

3.2 Resultados y discusión

De acuerdo a los resultados experimentales obtenidos después de la realización de cada uno de los 36 experimentos predefinidos en la tabla 14; podemos realizar un análisis de los resultados obtenidos.

En primer lugar, podemos estudiar de la fase de preprocesamiento, el tiempo necesario para la generación de los archivos necesarios para la visualización.

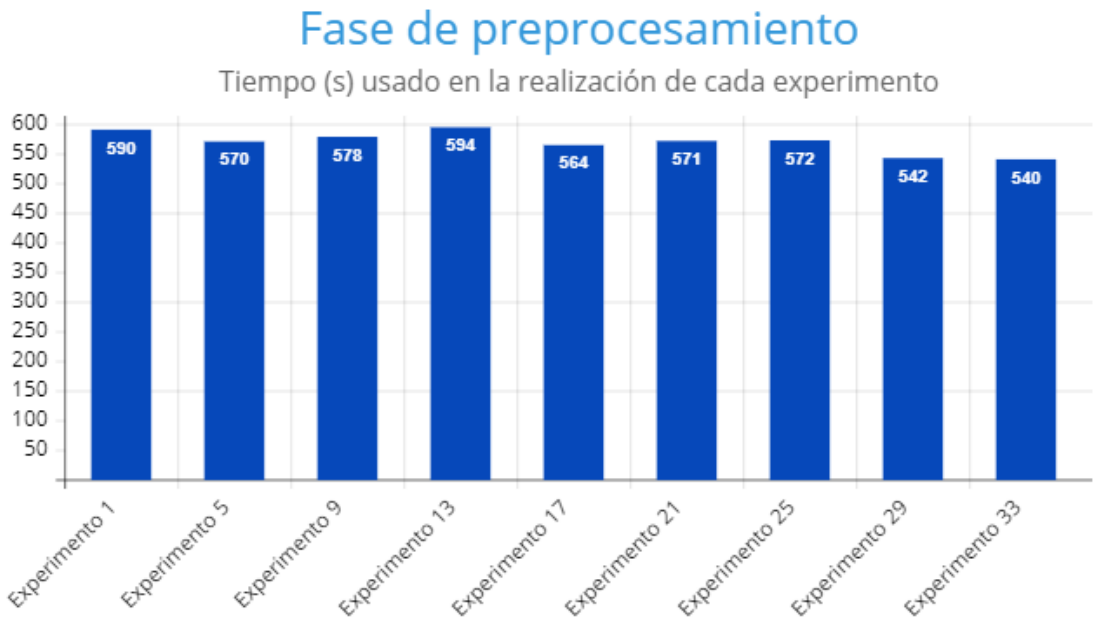

Ilustración 21. Tiempo usado en la realización de cada experimento

De acuerdo a la ilustración 21, que recoge todos los tiempos de preprocesamiento, en segundos, de los experimentos de esa índole se puede ver que la **parametrización indicada no influye significativamente en el tiempo necesario para preprocesar un dataset**. **El factor principal y más influyente** en el tiempo de preprocesamiento depende del **tamaño del conjunto de datos original principalmente**.

En otro orden de elementos a analizar, tenemos la fase de visualización, donde se pueden estudiar otras medidas tomadas en la experimentación.

Concretamente, llama la atención la variación de la tasa de refresco en función del presupuesto de puntos asociado, hecho que se puede ver gráficamente en la ilustración 22.

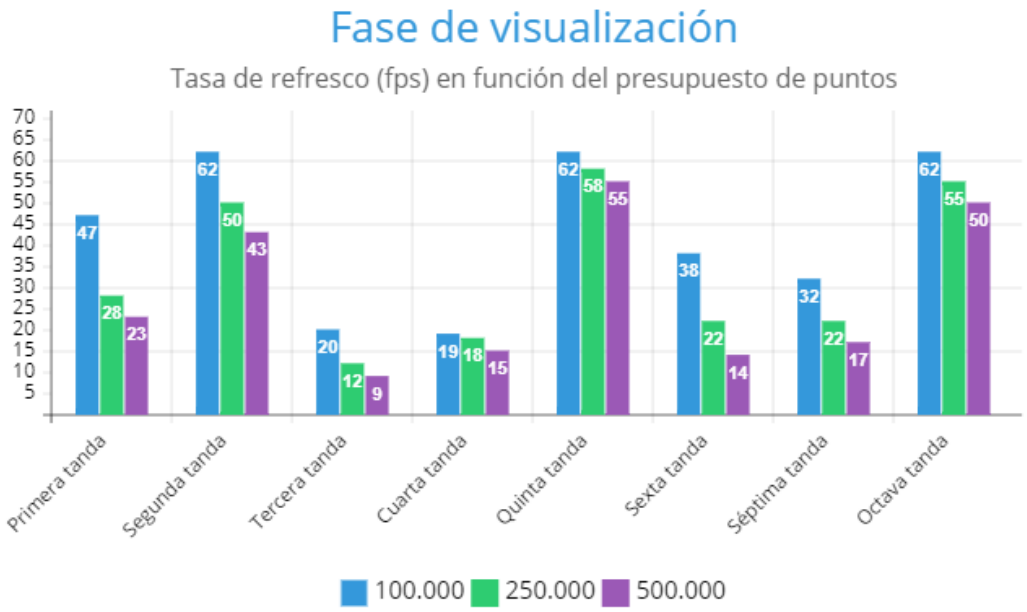

Ilustración 22. Tasa de refresco en función del presupuesto de puntos en cada experimento

Se puede observar que, **independientemente del tamaño de celda usado en la fase de preprocesamiento, existe una relación inversamente proporcional entre la tasa de refresco de cada experimento y el presupuesto de puntos asociado al mismo**. Del mismo modo, **existe una relación directamente proporcional entre el presupuesto de puntos y la mayor calidad de visualización**.

Otro elemento que destaca entre los resultados obtenidos en la experimentación es la correlación presente entre el tamaño de celda y el rendimiento de la aplicación. Esta correlación está reflejada en la ilustración 23; se observa en ella que, tanto para el tamaño de celda más pequeño como para el más grande, la tasa de refresco disminuye en más de la mitad de la obtenida con el tamaño de celda medio. La explicación se debe a que cuando se tienen celdas de tamaño muy pequeño, es necesario cargar más actores en la escena para poder visualizar una zona. En el otro extremo se tienen que cargar menos actores para la misma zona, pero dichos actores abarcan zonas de terreno superiores a las necesarias, de acuerdo a la política de selección de vecindario de este trabajo; por ello, cada uno de ellos

acarrea una cantidad de puntos abultada, lo que ocasiona la bajada de rendimiento de visualización e interacción.

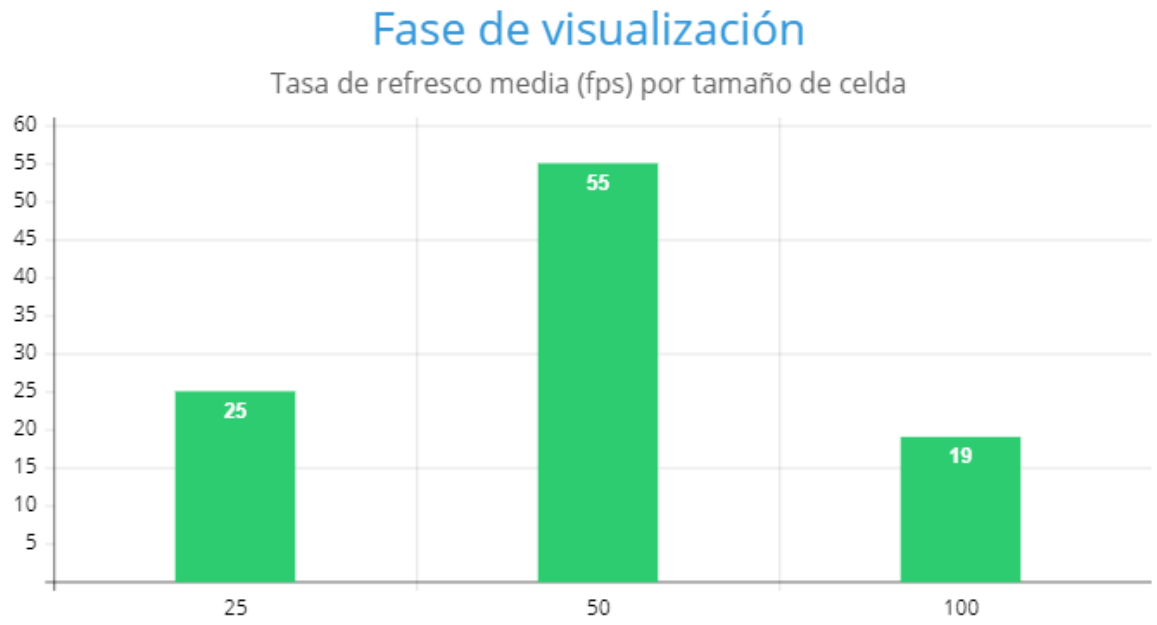

Ilustración 23. Tasa de refresco en función del tamaño de celda

La conclusión a la que se puede llegar después de analizar estos últimos elementos presentes en esta misma ilustración 23 es que, **con un tamaño de celda inadecuado, tanto demasiado pequeño como demasiado grande, el rendimiento de la fase de visualización se ve afectado**. En este caso, con un tamaño de celda de 50, para el dataset de prueba, se ofrece un rendimiento medio más que adecuado para la interacción en tiempo real. **Una posible extensión de este trabajo podría ser un script que determinase, para un determinado dataset, el tamaño de celda más adecuado para maximizar la tasa de refresco media**.

El último aspecto que podemos mencionar es el valor subjetivo del rendimiento del procesamiento de peticiones**. Existe una relación inversamente proporcional entre el presupuesto de puntos y este parámetro. A mayor presupuesto de puntos, menor rendimiento al procesar peticiones**. Esto se debe a que cuando se tiene un presupuesto mayor, se deben cargar más ficheros, y cada uno de ellos tiene un tiempo de lectura y procesamiento en tiempo real asociado. **También se debe tener en cuenta que la calidad de visualización es mayor en el último caso, al tener licencia para renderizar mayor cantidad de puntos por pantalla. Otra**

posible extensión de este trabajo podría ser un script que determinase, para un determinado dataset, el presupuesto de puntos más adecuado para maximizar la tasa de refresco media.

4 CONCLUSIONES Y TRABAJOS FUTUROS

En este proyecto hemos abordado desde la perspectiva que nos otorga el estudio del estado actual del arte de las nubes de puntos y de las estructuras de datos multiresolución más comunes una alternativa basada en un sparse grid parametrizable.

Los resultados obtenidos mediante el continuo testeo y la realización de los experimentos presentados en el tercer epígrafe nos permiten alcanzar unos objetivos previamente definidos que dan paso unas **conclusiones generales**:

1. El uso de un grid disperso se presenta como una solución válida y con una eficiencia válida como para poder trabajar con datasets masivos.
2. Debido a las continuas refactorizaciones llevadas a cabo en el desarrollo del proyecto y a seguir el patrón de diseño Modelo-Vista-Controlador, cualquiera de las partes se puede separar de las demás para poder sustituir el uso de una tecnología, algoritmo o estructura de datos de una manera sencilla; de tal modo que continuos trabajos que partan desde esta base no requieren de una amplia modificación del software.
3. La gran cantidad de experimentos y los resultados previsibles advierten de un sistema robusto y flexible a cambios.
4. El uso de la biblioteca de visualización VTK sirve para validar este prototipo, aunque siguientes iteraciones del software deben recurrir a APIs de más bajo nivel como OpenGL para poder exprimir aún más el rendimiento y evitar cuellos de botella imparalelizables con VTK.
5. El acoplamiento del cliente propuesto en este proyecto con un hipotético servidor, tal y como se planteó en las reuniones llevadas a cabo en el GGGJ no precisarían de una refactorización mayor del software actual.

6. La gran cantidad de parámetros incluidos en ambas fases de preprocesamiento y visualización permiten realizar una serie de experimentos muy variados y abre la puerta a la inclusión de nuevos proyectos para determinar los valores adecuados para cada una de esas variables para un dataset concreto.
7. El ámbito de las nubes de puntos abarca gran cantidad de campos diversos y cada día está más presente en el ámbito científico. Con este trabajo, se explora una vía explotable basada en la visualización de conjuntos de datos masivos; los cuales están cada vez más presentes en esta sociedad del conocimiento.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

El documento relativo al trabajo de fin de grado presente en la web de la escuela es el detallado en las tres siguientes ilustraciones 24, 25 y 26.

(Cód.: 19/20-2856) Visualización progresiva de datasets LIDAR de gran escala

Tutor del TFG: **CARLOS JAVIER OGAYAR ANGUITA**
Modalidad: Trabajo Teórico/Experimental | Tipo: TFG Específico
Número máximo de estudiantes: 1 (1 asignados)
Idioma: Castellano
Asignado al alumno con DNI:26054785V
Segundo tutor del TFG: **RAFAEL JESÚS SEGURA SÁNCHEZ**

Este trabajo consiste en la realización de un **estudio teórico-práctico** sobre las técnicas a aplicar para conseguir la visualización progresiva de grandes conjuntos de datos procedentes de escaneado 3D LiDAR. Este tipo de escaneado a gran escala es muy utilizado en ámbitos como la Ingeniería Civil, la Arquitectura (BIM), las ciudades inteligentes, la gestión del territorio (topografía y cartografía) o la arqueología. Uno de los grandes problemas que se presentan con este tipo de datos es la incapacidad de la mayoría de los sistemas para almacenar todos los datos en memoria principal, así como el insuficiente rendimiento a la hora de visualizar y monitorizar conjuntos de datos a gran escala. Con este trabajo se plantea la elaboración de un prototipo que permita visualizar de forma progresiva y adaptativa grandes conjuntos de datos LiDAR, para lo cual será necesario realizar un estudio teórico-práctico de las técnicas disponibles en la actualidad para conseguir dicho objetivo

Conocimientos Previos

Es recomendable tener conocimientos básicos de OpenGL, Vulkan o Metal, así como dominar el desarrollo de aplicaciones gráficas en alguna de las plataformas actuales. Además es importante conocer sistemas de tratamiento de nubes de puntos, como Point Cloud Library o Cloud Compare.

Ilustración 24. Presentación del TFG y conocimientos previos

Objetivos del TFG

- Realizar un estudio teórico-práctico sobre métodos de visualización de nubes de puntos, así como estructuras de datos y algoritmos orientados al tratamiento masivo de puntos 3D.
- Diseñar e implementar un prototipo de aplicación que permita el procesamiento de grandes conjuntos de datos LiDAR en tiempo real. Estudiar una posible estructura de datos multirresolución orientada a la carga progresiva de datos bajo demanda.
- Plantear un sistema productor-consumidor multitarea para resolver la carga de datos bajo demanda del sistema de visualización de datos.
- Realizar un benchmarking que permita comprobar la calidad de los resultados, así como el rendimiento obtenido, mediante el uso de diversos datasets reales de gran escala.
- Redactar las conclusiones del estudio destacando los aspectos más interesantes sobre las estructuras de datos y los métodos considerados, la implementación realizada y los resultados obtenidos. Realizar una propuesta de marco de trabajo consistente en la unión de las tecnologías experimentadas de cara a la elaboración de un posible proyecto de mayor envergadura.

Metodología a Desarrollar

- Elaborar una pequeña revisión bibliográfica del estado del arte sobre estructuras de datos, compresión, transmisión y mecanismos de visualización aplicados tanto a nubes de puntos en general, como a datos LiDAR en particular. Presentar las principales ventajas e inconvenientes de los métodos principales, así como una exposición de los motivos por los que se han seleccionado algunos de los métodos y estructuras de datos.
- Para el prototipo a realizar: seleccionar y utilizar una metodología basada en Ingeniería del Software para el análisis de requisitos, diseño, implementación, prueba y documentación. Se recomienda utilizar una metodología incremental.
- Detallar la estimación temporal y los costes de desarrollo.
- Completar el estudio teórico-experimental, incluyendo el desarrollo del prototipo.
- Generar una batería de pruebas (benchmarking) para el prototipo que cubra todos los aspectos implementados. Aplicar a uno o varios conjuntos de datos con características propias, como escaneados de edificios y zonas urbanas, así como grandes extensiones de terreno. Documentar convenientemente los resultados.
- Redactar la documentación final requerida, prestando especial atención a las conclusiones obtenidas con el estudio.
- Si se propone alguna solución relativamente novedosa, podría generarse una publicación científica en un congreso o revista.

Ilustración 25. Objetivos del TFG y metodología del mismo

Documentos y Formatos de Entrega

- Para el formato de documentación de los Trabajos teóricos o experimentales se utilizará una documentación de formato libre. Tal y como especifican las normas de estilo de la EPSJ: '*Los Trabajos teóricos o experimentales se estructurarán, en la medida de lo posible, en los siguientes apartados: Introducción, Antecedentes, Objetivos, Material y Métodos, Resultados y Discusión, Conclusiones, Planos y Anexos (si proceden) y Bibliografía*'. No obstante, es conveniente incluir algunas secciones propias de los proyectos de Ingeniería, ya que complementan adecuadamente el trabajo. Estas secciones pueden incluir, entre otros y sin carácter limitativo: normas y referencias, definiciones y abreviaturas, alcance del trabajo, especificación del prototipo realizado, planificación temporal y presupuesto.
- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente completo y ejecutables del prototipo o software desarrollado.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Fuentes de datos utilizadas. De especial importancia en trabajos teóricos o experimentales y estudios técnicos, donde es imprescindible contar con los datasets utilizados (incluidos o referenciados).
- Vídeo de demostración de la aplicación o de la experimentación realizada.
- Anexos para la presentación del trabajo:□ Informe favorable del tutor.□ Autorización de publicación, si procede.

Ilustración 26. Documentos y formatos de entrega del TFG

6 DEFINICIONES Y ABREVIATURAS

En esta sección se definen abreviaturas o acrónimos de conceptos mencionados en el trabajo.

- **LAS**: LASer. Formato de fichero estándar de nubes de puntos.
- **LAZ**: LASer comprimido (Zip). Formato de fichero estándar de nubes de puntos.
- **LiDAR**: Light Detection and Ranging. Tecnología que permite escanear zonas para obtener nubes de puntos.
- **MVC**: Modelo-vista-controlador. Patrón de diseño de aplicaciones ampliamente usado.
- **RAM**: Random Access Memory. Memoria temporal del computador donde se almacenan datos que la unidad central de procesamiento está procesando o va a procesar en un determinado momento.
- **CSV**: Comma Separated Values. Tipo de documento sencillo para representar datos en un formato estandarizado.
- **VTK**: Visualization Toolkit. Conjunto de herramientas de visualización disponible para la realización de gráficos 3D por computadora, procesamiento de imagen y visualización.
- **LOD**: Level of detail. Elemento que determina el nivel de detalle de un modelo en 3D para así disminuir la complejidad que presenta conforme se aleja del observador o acorde a otro tipo de métricas.
- **TDA**: Tipo de dato abstracto. Usado para referirse a conjuntos de datos no estándar. Término ampliamente usado en la programación orientada a objetos.
- **EEDD**: Estructuras de datos. Elementos que permiten organizar datos en una computadora para que puedan ser utilizados de manera eficiente.
- **MS**: Milisegundos. Unidad de medida de tiempo del sistema internacional. 1000 milisegundos conforman un segundo.

- **GLM**: OpenGL Mathematics. Biblioteca de operaciones matemáticas para C++ centrada en el desarrollo de aplicaciones gráficas.
- **STL**: Standard Template Library. Biblioteca de clases contenedor, algoritmos e iteradores en C++ para proporcionar estructuras de datos programables comunes y funciones tales como listas, pilas, arrays, etc.
- **ROI**: Region of interest. Muestras de un conjuntos de datos identificadas para un propósito particular.
- **MNO**: Modifiable Nested Octree. Estructura de datos tridimensional de gran complejidad e implementada en el software de visualización de nubes de puntos Potree.
- **API**: Application programming interface. Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro sotfware como una capa de abstracción.
- **FPS**: Fotogramas por segundo. Frecuencia a la cual un dispositivo muestra imágenes llamados fotogramas. Este término es ampliamente acuñado en el ámbito de los gráficos por computadora.

7 BIBLIOGRAFÍA

Wand M. et al. 2007. Processing and interactive editing of huge points clouds from 3D scanners. Eurographics Symposium on Point-Based Graphics (2007).

Fernández J.C. et al. 2008. An Overview of Lidar Point Cloud Processing Software. Civil and Coastal Engineering Department, University of Florida

Kovac B, Zalik B. 2010. Visualization of LIDAR datasets using point-based rendering technique. Computers & Geosciences.

Isenburg, M., 2011. LASzip: lossless compression of LiDAR data. Photogrammetric Engineering and Remote Sensing.

Wenzel K. et al. 2014. An out-of-core octree for massive point cloud processing. Institute for Photogrammetry, University of Stuttgart.

Yu Fou et al. 2014. Simple Octree Solution for Multi-resolution LiDAR Processing and Visualization. Nokia Finland Research Center.

Schuetz, M. 2015. Potree – Rendering Large Point Clouds in Web Browsers. Vienna University of Technology.

Maximilian-Fraiss, S., 2017. Rendering Large Point Clouds in Unity. Vienna University of Technology.

Deibe, D., et al., 2017. GVLiDAR: An Interactive Web-based Visualization Framework to Support Geospatial Measures on LiDAR Data. International Journal of Remote Sensing.

Deibe, D., et al., 2019. Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. International Journal of Geographical Information Science.

Kraft, V., 2019. Efficient rendering of massive and dynamic point cloud data in state-of-the-art graphics engines. Universität Bremen.

Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.