Universidad Zaragoza
1542

Trabajo Fin de Grado

Grado en Ingeniería Informática

Generación de mapas 3D teselados para su visualización en web

Generation of tiled 3D maps for their visualization on the web

Autor

Pablo Viñuales Sánchez

Director

Rubén Béjar Hernández

Escuela de Ingeniería y Arquitectura (EINA)

2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Pablo Viñuales Sánchez,

con nº de DNI 18056437B en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Generación de mapas 3D teselados para su visualización en web,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 13 de abril de 2018

Fdo: Pablo Viñuales Sánchez

Generación de mapas 3D teselados para su visualización en web

RESUMEN

El objetivo de este proyecto es realizar una aplicación capaz de generar mapas 3D teselados, a partir de unos datos de entrada que pueden ser descargados de repositorios públicos, para posteriormente ser visualizados mediante una aplicación web.

La aplicación permite varias formas de ser ejecutada según si el usuario quiere renderizar toda la extensión disponible o solo una parte de ella. Realiza transformaciones sobre los datos de entrada para poder ser incluidos en los ficheros utilizados para especificar la escena que posteriormente es generada a partir del programa renderizador.

Para generar el fichero utilizado por el programa renderizador, además de los ficheros iniciales transformados, es necesario la especificación de los objetos heightfield, principalmente (esferas en un segundo plano), que son los que producen la sensación de volumen en el aspecto de la escena. También es necesario generar una cámara correcta para la escena según los parámetros especificados por el usuario (dirección de vista, ángulo de incidencia de la cámara...) y generar la escena con un ratio de aspecto que otorgue ese aspecto de perspectiva.

La aplicación también es capaz de teselar el resultado obtenido de forma correcta, atendiendo al código que debe tener cada tesela, el tamaño de esta o dónde ha de ser almacenada.

Dentro del proyecto entra la realización de un cliente sencillo que permita visualizar los datos obtenidos. Para la comprobación de los resultados que se obtienen, en cuanto a lo visual, como de rendimiento, se ha generado una extensión grande como es la Comunidad Autónoma de Aragón y en varios puntos de vista y niveles de zoom. Estos resultados son especificados, además de una serie de estimaciones para mayores extensiones.

El desarrollo del proyecto se ha realizado principalmente en cuatro fases diferentes: una primera fase donde se han analizado los datos necesarios, además de realizar un pequeño acercamiento a ellos y los resultados esperados a partir de pequeñas aplicaciones de procesado de estos datos y aspectos manuales para una zona pequeña; una segunda fase donde se automatiza lo anterior y extendiéndolo a una zona mayor; una tercera fase con la creación del visualizador y la concreción de aspectos relacionados con este (teselas) comenzados en la segunda fase; y una última fase con la realización de la documentación.

A pesar de lo desarrollado habría aun trabajo posible en forma de mejorar la eficiencia del programa, ya sea optimizando los procesos o con la paralelización de estos, y mejorando el aspecto resultante, incidiendo todavía más en el procesado de algunos de los datos.

Índice

1. INTRODUCCIÓN

A continuación, se presenta la memoria del proyecto de fin de Grado o Trabajo de Fin de Grado (TFG) del alumno Pablo Viñuales Sánchez (679609), del Grado de Ingeniería Informática, especialidad Ingeniería del Software, de la Escuela de Ingeniería y Arquitectura (EINA) de la Universidad de Zaragoza (Unizar). El proyecto cuenta con un director perteneciente a la propia EINA y profesor del Grado de Ingeniería Informática Rubén Béjar Hernández, perteneciente al departamento de Informática e Ingeniería de Sistemas del área de Lenguajes y Sistemas Informáticos.

El proyecto se incluye como último paso (asignatura) antes de la obtención del título por parte del alumno, parte desde cero y consiste en la realización de un programa capaz de generar mapas teselados en 3D a partir de unos datos de entrada que pueden ser descargados de repositorios públicos, y a causa de la búsqueda y selección de los datos que se necesitan para la realización del proyecto, se seguirá una metodología iterativa que permita adaptarse a lo que se vaya descubriendo y probando.

Esta memoria está dividida en varias secciones, y estas a su vez en varias subsecciones, en donde se explican los diferentes conceptos del proyecto. Primero, en la sección 2 se realiza un análisis del problema, donde se incluye una primera aproximación al proyecto con los objetivos que se proponen para la realización de el mismo, los requisitos del sistema, las alternativas a este y la información de los datos con los que se va a trabajar. Posteriormente, se continuará con la parte central del proyecto en las secciones 3 y 4: el diseño del programa a utilizar y sus procesos, además de las bibliotecas y programas que han sido utilizados; y detalles de la implementación, con los problemas encontrados en el desarrollo y los resultados obtenidos en lo que a rendimiento y prestaciones se refiere. Tras esto, se incluyen una serie de ejemplos de resultados obtenidos con alguna pequeña explicación y el cómo se presenta el futuro del proyecto (que cosas se esperan hacer para mejorarlo) en la sección 5. Además, se incluye una sección referente a la gestión del proyecto (sección 6) y otra con las conclusiones obtenidas tras la realización de el mismo (sección 7).

Anterior a esta sección de introducción se incluye el índice, y al final de este documento, una serie de anexos que complementan la información.

2. ANÁLISIS DEL PROBLEMA

2.1. OBJETIVOS

El objetivo principal es crear las herramientas y procesos necesarios para producir mapas realistas del territorio en tres dimensiones que puedan teselarse (dividirse en imágenes rectangulares, estilo Google Maps) para su visualización rápida en web. Para ello se generarán con estilos y puntos de vista predeterminados y una proyección ortogonal.

Se busca desarrollar una aplicación web que muestre los resultados obtenidos de generar un mapa 3D extenso a partir de cuatro direcciones de vista diferentes y dos ángulos de vista diferentes, utilizando el programa realizado para este proyecto. Además, se busca también que personas ajenas a este proyecto sean capaces de generar sus propios mapas (siguiendo una serie de instrucciones) mediante la herramienta desarrollada.

El código del proyecto se realizará utilizando el lenguaje de programación Python[1] en su versión 3, ya que es adecuado para la manipulación de datos y la automatización de procesos. Para renderizar los mapas se utilizará la herramienta libre POV-Ray[2]. Todas las librerías y herramientas a utilizar en el proyecto serán libres.

2.2. DATOS DE PARTIDA

En las diferentes páginas web que poseen información geográfica es posible encontrar archivos que contienen fotos aéreas, mapas de alturas o incluso representaciones muy precisas de una zona en particular. La idea del proyecto es que, a partir de estos archivos, se pueda realizar un programa que aglutine todos estos y genere una salida, en forma de imagen, que pueda ser posteriormente dividida en pequeños trocitos (teselas) para ser mostradas en una web.

A continuación, se especifican los archivos que se han obtenido de las diferentes páginas web, la información que representan y la forma de representar esta información (formato). Para trabajar con datos geográficos en Europa, y por lo tanto en España y Aragón, se utiliza el sistema de referencia geodésico ETRS89[3]. A partir de este sistema de referencia (datum), se utiliza UTM[4] como sistema de coordenadas. Con este sistema el mundo se divide en diferentes zonas (husos) y Aragón principalmente se encuentra en el huso 30N. Aun así, la parte más al este de Aragón (aproximadamente a partir de Barbastro) se encuentra en el huso 31N lo que hace que algunas ortofotos se encuentren en UTM 31N y haya que reproyectarlas para adecuarlas al huso 30N que es el que comparte la mayoría de los datos de entrada. Esta es ETRS89 / UTM zone 30N [5] representada por el código (EPSG:25830).

2.2.1. Modelo digital del terreno

Estos archivos, que se representan con una extensión .ASC (ASCII matriz ESRI[6]), son descargados del centro de descargas del Centro Nacional de Información Geográfica (CNIG[7]) (http://centrodedescargas.cnig.es/CentroDescargas/buscador.do). Representan, como su propio nombre indica, un modelo digital del terreno de la zona que se ha descargado, es decir, información altimétrica que representa el relieve de este terreno. Cada archivo abarca aproximadamente una zona de 760km2 (38kmx20km) con paso de malla de 5m (cada punto representa una zona de 5mx5m). Es el más preciso que se puede encontrar en la web.

Cada archivo MDT contiene 2 partes diferenciadas. Una primera parte que podríamos llamar cabecera, donde se encuentran los datos relativos al tamaño del fichero, coordenadas de su punto inferior izquierdo, tamaño de cada punto en la realidad y el valor que se da a los puntos

que no contienen información. La segunda parte del fichero es una matriz de valores reales, donde cada valor representa la altura del terreno en el punto especificado por su posición en la matriz, y las coordenadas geográficas del punto inicial (inferior izquierdo).

Con estos archivos MDT se construyen los modelos 3D del terreno para el proyecto sobre los que luego se aplican las ortofotos, además de otros aspectos para dotarlos de realismo.

2.2.2. Ortofotos del PNOA

Las ortofotos del PNOA (Plan Nacional de Ortofotografía Aérea[8]) son fotos aéreas de una zona concreta a las que se les ha realizado un cierto procesamiento para poder usarse como un mapa. Son descargadas de la web de IdeAragon (Infraestructura de Datos Espaciales de Aragón) (http://idearagon.aragon.es/). Para realizar este proyecto, se han elegido las de máxima actualidad (del año 2015) donde cada pixel de la imagen representa 0.5m en la realidad. Para cada MDT hay 16 ortofotos. Aunque tengan extensión .JPG, estos archivos son directorios donde se encuentran (ahora sí) la imagen en formato .JPG y un archivo .JGW, que contiene la información geográfica (coordenadas del pixel inicial, tamaño del pixel…).

2.2.3. LiDAR

Los archivos LiDAR representan información altimétrica de una zona mediante una nube de puntos que ha sido generada utilizando sistemas LiDAR. Estos sistemas LiDAR (Light Detection and Ranging o Laser Imaging Detection and Ranging[9]) permiten determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz de láser. Cada archivo LiDAR representa una zona de 4km2 y han sido descargados del centro de descargas del CNIG. Se encuentran en formato .LAZ que es el formato de compresión de ficheros LAS. Para poder trabajar con ellos es necesario “descomprimir” estos ficheros (pasarlos a formato .LAS).

Estos archivos están formados por una serie de cabeceras, donde se encuentra la metainformación del fichero, y la parte principal del fichero, donde se encuentran los diferentes puntos. Cada punto tiene unas coordenadas geográficas, altura, color del punto, entre otra información menos relevante. A pesar de que cada fichero contiene una gran cantidad de puntos, en según qué zonas la densidad de estos puntos no es suficiente para poder representar con máxima precisión el terreno al que pertenece, además, el color tampoco es muy preciso. Con LiDAR se pueden representar ciudades de forma muy precisa (si la densidad de puntos es suficiente), algo que los archivos MDT no son capaces de hacer, por ello se ha decidido limitar las zonas de interés del LiDAR a ciudades o alguna zona boscosa.

2.3. ALTERNATIVAS

La idea de realizar el proyecto utilizando estos datos siempre ha estado presente, aunque sí que de forma algo diferente. La primera idea era realizar dos tipos de mapas diferentes, uno utilizando los MDT y PNOA juntos y otro utilizando LiDAR. Al final se ha optado por realizar un tipo de mapa que incluye los tres, aunque el LiDAR de forma limitada a causa de las circunstancias especificadas con anterioridad (poca densidad de puntos).

Se consideró en un primer momento elegir una zona mayor para mostrar el resultado del proyecto; también se planteó hacerlo en una zona algo menor. Sin embargo, se llegó a la conclusión de que un punto intermedio sería lo mejor, como lo es Aragón (ya que además posee bastante variabilidad en lo que accidentes geográficos se refiere), e incluir un cálculo aproximado del tiempo y espacio que costaría realizarlo para toda España.

Otro aspecto que se llegó a considerar fue el sustituir el renderizado mediante el programa POV-Ray por renderizado utilizando Three.js[10]. Three.js es una biblioteca JavaScript para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web. Por el momento se descartó ya que se consigue una mayor calidad de imagen con un sistema como POV-Ray, basado en ray-tracing, a cambio de un mayor tiempo de renderizado que lo hace inviable para visualizar datos 3D en tiempo real. Se podría considerar en un futuro sustituir el sistema de renderizado del proyecto por este.

2.3.1. Proyectos similares

La generación de mapas teselados en 3 dimensiones no es algo de reciente creación. Como ya se sabe, existen varios sistemas muy parecidos que posibilitan el visualizar de forma interactiva mapas 3D. Quizás, el sistema más conocido y más utilizado sea Google Maps[11].

Google Maps es un servidor de aplicaciones de mapas en la web que ofrece mapas desplazables, así como fotografías por satélite del mundo, la ruta entre diferentes ubicaciones e incluso imágenes a pie de calle. Está claro que al tratarse este proyecto de un sistema de no tanta envergadura como Google Maps, no posee tantas características como el desarrollado por Google. Aun así, existen varios puntos en los que el sistema desarrollado es favorable respecto al de Google:

- El sistema desarrollado para este proyecto está basado exclusivamente en datos (y programas) disponibles libremente, es decir, es posible descargarlos gratis e incluirlos en el programa.

- La aplicación de visualización funciona en máquinas muy básicas (móviles, tablets...) porque esta aplicación es bastante simple y su tarea es únicamente mostrar imágenes (teselas) o cambiar entre varias capas de imágenes (cambio de punto de vista).

- El diseño arquitectural en forma de pipeline permite sustituir algunos de sus componentes por soluciones más interactivas. En un futuro se podría sustituir los componentes que generan y renderizan la escena de POV-Ray por la generación de un fichero en lenguaje JavaScript basado en Three.js.

- Al renderizar de forma offline y posibilitando renderizar zonas concretas (o teselas) permite aplicar aspectos visuales que renderizados en tiempo real se generan peor. Por ejemplo, una zona con un lago con unos reflejos perfectamente calculados con ray-tracing (determinar el aspecto del objeto a partir de trazar rayos desde la cámara hasta la escena) o radiosidad (cálculo de la iluminación global para renderizarlo con un aspecto lo más realista posible).

2.4. REQUISITOS

A la hora de especificar los requisitos, se ha optado por diferenciarlos en dos aspectos diferentes. Un primero con los requisitos del programa a desarrollar (con código RP) y un segundo con los requisitos de la aplicación web donde se mostrarán los resultados (con código RC).

2.4.1. Requisitos programa

REQUISITOS FUNCIONALES	
RPF-1	La aplicación debe permitir transformar datos geográficos de fuentes públicas en mapas teselados.
RPF-2	Los mapas teselados generados deben ofrecer una perspectiva isométrica de la escena deseada.
RPF-3	La aplicación debe permitir generar un mapa teselado en vista isométrica a partir de todos los ficheros disponibles en el sistema.
RPF-4	La aplicación debe permitir generar una vista isométrica de una zona especificada por el código de las teselas de su esquina superior izquierda y su esquina inferior derecha.
RPF-5	La aplicación debe permitir generar una vista isométrica de una zona especificada por las coordenadas de su esquina superior izquierda y su esquina inferior derecha.
RPF-6	La aplicación debe permitir generar una vista isométrica según 4 puntos de vista diferentes (norte, sur, este y oeste).
RPF-7	La aplicación debe permitir generar una vista isométrica desde 2 ángulos diferentes (30º y 45º).
REQUISITOS NO FUNCIONALES	
RPNF-1	La aplicación es ejecutada desde una terminal Shell.
RPNF-2	La aplicación trabaja con coordenadas en ETRS89 / UTM zone 30N.

Figura 1: Requisitos programa

2.4.2. Requisitos aplicación web

REQUISITOS FUNCIONALES	
RCF-1	El cliente web debe soportar mapas teselados.
RCF-2	El cliente web debe mostrar mapas a varios niveles de zoom.
RCF-3	El cliente web debe permitir cambiar el modo de visualización del mapa.
REQUISITOS NO FUNCIONALES	
RCNF-1	El cliente debe funcionar en las últimas versiones de los navegadores Mozilla Firefox, Google Chrome y Microsoft Edge.
RCNF-2	Los modos de visualización son: norte, sur, este y oeste, por un lado y 30º y 45º por otro.

Figura 2: Requisitos aplicación web

3. DISEÑO DE LA SOLUCIÓN

3.1. DISEÑO DEL SOFTWARE CREADO

A la hora de diseñar el software necesario para el programa a desarrollar, dadas las características del proyecto (poca o nula interacción con el usuario, transformación de unos datos de entrada en otros de salida...) se ha diseñado una arquitectura de flujo de datos, con el patrón arquitectural de filtros y tuberías[12]. Esta arquitectura dicta que unos datos de entrada navegan mediante unas tuberías por diferentes filtros donde se procesan transformándolos en la salida deseada al final del último filtro. Cada filtro es una entidad independiente sin estado (no importa que datos/ficheros han pasado con anterioridad, realiza siempre el mismo proceso).

En el sistema desarrollado en el proyecto, cada filtro es representado por un componente transformador y el programa principal hace de coordinador, obteniendo la salida de un filtro y pasándoselo al siguiente o a otro proceso. También, es posible una ejecución parcial a través de sus módulos, posibilitando que el cliente pueda utilizar esas funciones de forma ajena al programa principal, por si le es necesario utilizar algunas de estas para sí mismo o sencillamente para no realizar todo el proceso completo. Aun así, decir que es posible "activar/desactivar" algunas de estas funciones para no ser utilizadas siempre. Posteriormente se incidirá en estas opciones.

3.2. PROCESOS DISEÑADOS

Como ya se ha comentado con anterioridad, se ha seguido un diseño de flujo de datos. Como se muestra en el diagrama de la Figura 3, desde el principio se han ido desarrollando una serie de procesos para transformar los datos de entrada (descargados), en pequeñas porciones de imágenes (teselas) que representan un mapa.

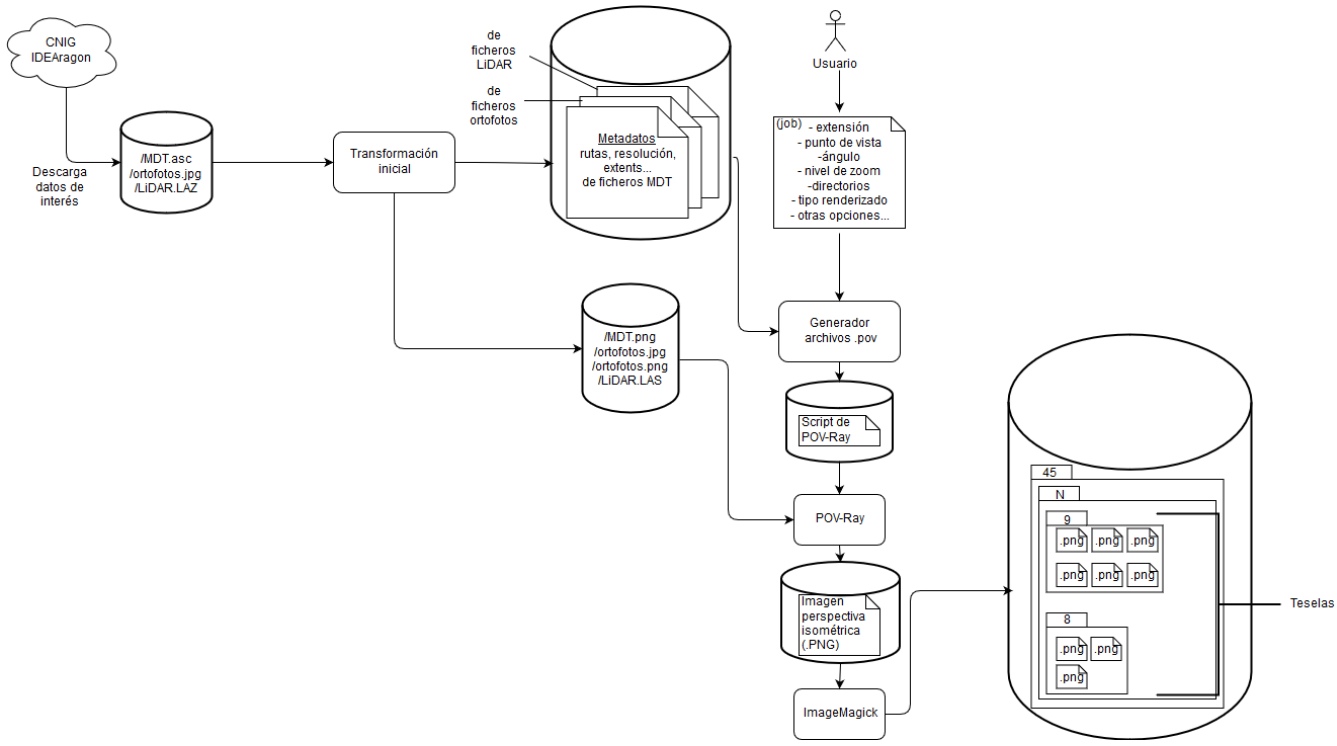

Figura 3: Diagrama procesos del sistema

Principalmente, el sistema se puede dividir en cuatro procesos principales diferentes. A su vez estos procesos principales están compuestos de otros procesos menores. A pesar de tratarse en este diagrama un proceso completo del sistema, normalmente la parte izquierda y central se realiza una sola vez (hasta el momento y si no cambian los datos) y la que se realiza siempre en cada ejecución es la parte de la derecha. A continuación, se extiende esto.

El primer proceso principal del sistema es el de descarga de los datos de entrada. Este proceso es hasta el momento completamente manual y se incluye la automatización del mismo como trabajo futuro. El proceso consiste en la selección de datos interesantes para el proyecto como lo son los ficheros MDT y LiDAR del repositorio del CNIG (Centro Nacional de Información Geográfica) y las ortofotos provenientes de IDEAragón. Al tratarse de un proceso manual, el autor del proyecto ha debido buscar en los centros de descargas de estas instituciones los ficheros correspondientes a la Comunidad de Aragón y con las características que se describen en la sección X.X de Tratamiento de datos, y almacenarlos en un disco duro (externo) para ser posteriormente utilizados por el programa.

El siguiente proceso es el de tratamiento de los datos descargados. Tal y como son descargados y almacenados los datos de entrada al sistema no pueden ser utilizados, al menos algunos de ellos. Para poder adaptarlos a las exigencias de las bibliotecas utilizadas y el resultado deseado, estos datos de entrada han debido pasar por un proceso de transformación de datos. Este proceso se puede considerar el primer filtro del sistema. Solo parte de las ortofotos (las que se encuentran en la proyección ETRS 89 / UTM 30N) no han necesitado de este tratamiento de datos y tal como han sido descargadas son utilizadas en el sistema. Una mayor información de los procesos realizados a cada tipo de fichero se puede encontrar en la sección 3.5 de

Tratamiento de datos y en el Anexo B donde se especifican los procesos menores que componen el filtro principal. Comentar también que en este proceso se generan una serie de ficheros (uno para cada tipo de fichero, es decir, tres en total) con los metadatos de los ficheros de entrada del sistema (información de ruta del fichero, tamaño, información geográfica...) que posteriormente son utilizados para obtener los datos necesarios para renderizar la escena deseada.

Estos dos primeros procesos pueden ser realizados una única vez, ya que tanto los ficheros descargados, como los transformados y los ficheros de metadatos generados, no van a cambiar a lo largo del tiempo. Esto permite una mayor eficiencia del programa principal evitando realizar estos procesos en cada ejecución del sistema. Por el contrario, si se desea añadir o eliminar algún dato de entrada (por ejemplo, incluir datos de la Comunidad de Navarra) es necesario descargar los ficheros correspondientes y volver a realizar estos procesos para mantener una coherencia entre los ficheros que se poseen, sus homónimos transformados y los ficheros de metadatos.

El tercer proceso es quizás el más importante, se trata de la generación del fichero en formato POV. Se considera este proceso el más importante o el principal porque su tarea no es solo la de generar este fichero, sino que tiene que interpretar lo que el usuario le pide y actuar en consecuencia. Así pues, se encarga de recibir la información que el usuario incluye a la hora de ejecutar el programa y de contestar a las preguntas que él mismo genera (por ejemplo, ¿desde qué coordenada desea renderizar?). Esta información que incluye datos de la escena como el punto de vista y ángulo de incidencia de la cámara o sencillamente la extensión de la escena deseada, ha de ser interpretada por el programa de forma que sea capaz de generar una cadena de caracteres, de forma que el programa POV-Ray sepa interpretarla, que incluya la definición de una cámara y los objetos heightfield (y esferas) a incluir. Esto se hace incluyendo los ficheros de entrada que corresponden a la escena pedida por el usuario. Por todo ello se considera el proceso principal, ya que debe comunicarse con el usuario, consultar los metadatos, incluir los ficheros de entrada y generar una salida interpretable. Este proceso ha de ser perfecto para que no se incluyan ficheros de más o de menos o sencillamente para que la cámara se encuentre en la posición perfecta y orientada correctamente. Si esto no fuese así, ejecuciones consecutivas del programa de zonas adyacentes podrían no casar unas con otras a causa de los errores en cálculos o sencillamente de inclusión de ficheros de entrada transformados. Una vez generada esta cadena de caracteres, se incluye en un fichero con extensión .POV que puede ser interpretado posteriormente por el programa POV-Ray.

Por último, el proceso final del sistema es el encargado de renderizar la escena y teselarla. Este proceso es quizás a simple vista el más simple porque se apoya en la utilización de dos programas: POV-Ray para renderizar y ImageMagick[13] para teselar. Una vez el fichero POV ha sido generado, este es pasado como parámetro al programa POV-Ray para renderizar la escena deseada y de forma que muestre el ratio de aspecto deseado (calculado al generar la cámara). Tras la ejecución de este programa se obtiene una imagen en formato PNG de toda la escena y con resolución dependiente del nivel de zoom introducido por el usuario. El proceso de teselado no es tan simple porque ha de calcular a qué teselas corresponde la escena y generarlas incluyendo en su nombre el código correspondiente a su posición. Esto es calculado a partir de una serie de especificaciones de extensión de Aragón y de forma que respete el ratio de aspecto

de la escena. Se utiliza el programa ImageMagick, incluido en las distribuciones GNU-Linux, para partir en pequeños trocitos la imagen que se ha obtenido del renderizado del POV-Ray. Estos pequeños trocitos son sencillamente pequeñas imágenes (teselas) en formato PNG que mantienen el ratio de aspecto con el que se ha renderizado la escena y que posteriormente son interpretados en el cliente para "pintar" los mapas en el navegador del usuario.

Estos procesos han sido definidos a alto nivel de forma que no se tienen en cuenta los módulos en los que están repartidos o la forma de implementación. En los posteriores apartados se incluye una mayor información sobre esto.

3.3. MÓDULOS

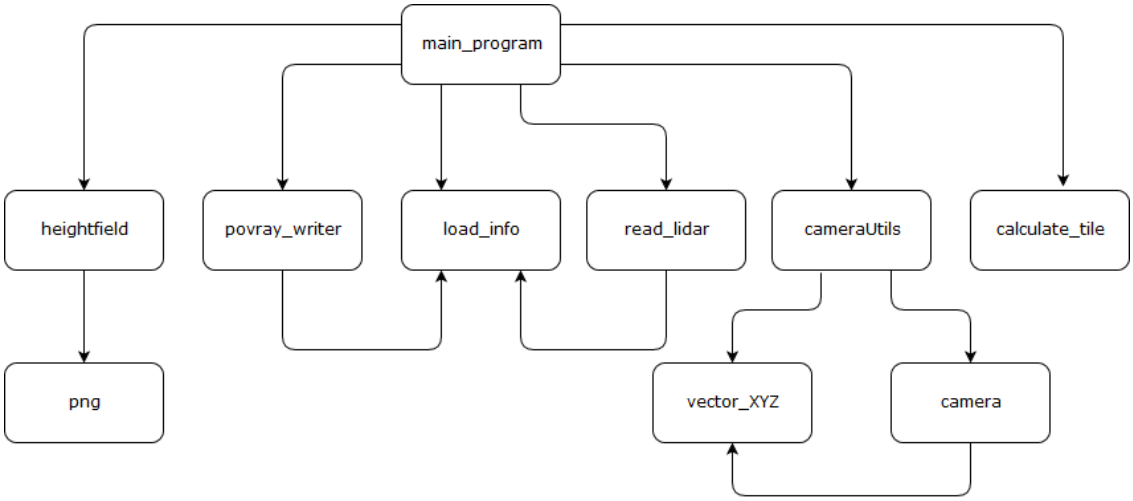

Figura 4: Diagrama módulos

Tal y como se ha explicado con anterioridad y como se puede comprobar en el diagrama de módulos de la Figura 4, main_program es el módulo principal del sistema y el que se comunica con el resto de los módulos, hace de coordinador. Algunos de estos también se comunican unos con otros, pero esto no quiere decir que se produzca un flujo de datos entre ellos, sino que, hacen uso de alguna función pública que se encuentra en el módulo objetivo. Concretamente tanto povray_writer como read_lidar utilizan una función del módulo load_info que permite comprobar si existe una colisión (si abarcan ambos una misma área) entre dos rectángulos definidos por las coordenadas pasadas como parámetro. El módulo main_program se encarga también de ejecutar los programas externos de renderizado y teselado (POV-Ray e ImageMagick), además de la coordinación entre los resultados que se obtienen a la hora de por ejemplo calcular las coordenadas límite de una tesela y aportar estos datos al módulo que se encarga de generar la cámara necesaria para representar la escena.

El módulo heightfield se encarga de realizar la transformación inicial de los MDT, load_info es el encargado de generar los archivos de metadatos y buscar en ellos los ficheros que se necesitan. El módulo read_lidar se encarga de realizar la transformación inicial de los objetos LiDAR, de obtener los puntos que se desean y generar la cadena de caracteres que representa los objetos esfera del fichero POV. Por su parte, povray_writer se encarga de generar la cadena de

caracteres de los objetos heightfield, la cámara y la fuente de luz del fichero POV; también, crea el fichero con extensión POV e incluye tanto estas cadenas de caracteres como la de esferas (si procede). El módulo calculate_tile, se utiliza para todas las operaciones relacionadas con el trato de teselas (cálculo de teselas necesarias, transformación de su código en otro...).

Se ha decidido incluir en el diagrama el módulo png, que como se va a explicar posteriormente, pertenece a una biblioteca que ha sido descargada. Al tratarse de otro fichero incluido en el directorio del proyecto y ser importado como el resto de los módulos se ha decidido darle una función de módulo. Solo es usado por heightfield.

Por último, se incluyen los únicos dos módulos que crean objetos en el sistema (son clases). Estos objetos son representados por la Figura 5.

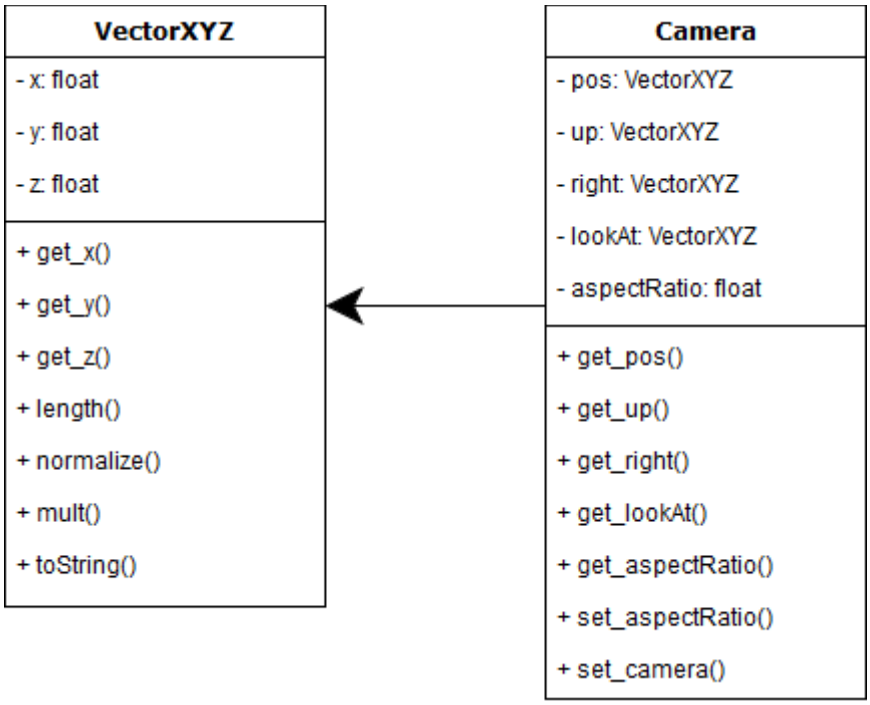

Figura 5: Diagrama clases

Son las únicas clases del sistema y están relacionadas. Los objetos que generan ambas son utilizados por el módulo cameraUtils, que es el encargado de realizar los cálculos necesarios para obtener una cámara que aporte a la escena de ese aspecto en 3D con una vista isométrica. Además, el objeto Camera al estar formado por objetos VectorXYZ, explica la relación entre ellos. Se han decidido incluir por limpieza tan solo los métodos utilizados en el sistema, ya que ambos poseen alguno más que se desarrolló con antelación pero que en la versión actual no es utilizado.

En el Anexo C (Documentación módulos) se puede encontrar una especificación de las operaciones, y sus parámetros, incluidas en cada módulo.

3.4. BIBLIOTECAS Y PROGRAMAS USADOS

3.4.1. Bibliotecas

Como se ha podido comprobar en el anterior apartado, se han utilizado varios módulos y bibliotecas de las que se han obtenido alguna funcionalidad necesaria. Dentro de la librería estándar de Python[14], se han incluido algunos módulos los cuales pueden ser usados incluyendo la cláusula <<import>> seguido del nombre de los módulos que se desean en cada fichero donde van a ser utilizados. Estos son:

- *math*: módulo que provee acceso a funciones matemáticas. Es importado en los siguientes módulos: <<read_lidar>>, <<calculate_tile>>, <<cameraUtils>>, <<vector_XYZ>>.
- *os:* módulo que provee acceso a funcionalidad dependiente del sistema operativo como la navegación por los distintos ficheros de un directorio. Es importado en los siguientes módulos: <<main_program>>, <<read_lidar>>, <<povray_writer>>.
- *random:* módulo que genera números pseudo-aleatorios. Es importado en los siguientes módulos: <<read_lidar>>.
- *sys:* módulo que provee acceso a algunas variables y funciones que interactuan con el intérprete de comandos. Es importado en los siguientes módulos: <<heightfield>>.
- *argparse:* módulo que permite escribir código para generar interfaces de usuario de línea de comandos de forma sencilla. Es importado en los siguientes módulos: <<main_program>>.
- *time:* módulo que provee de varias funciones relacionadas con el tiempo del sistema. Es importado en los siguientes módulos: <<main_program>>.

Por otra parte, el módulo *doctest*, incluido también en la librería estándar de Python, ha sido utilizado para realizar pruebas unitarias sobre el código. Será explicado con más atención posteriormente, en la sección 5.3.1 de pruebas unitarias.

Además de estos módulos, han sido necesarios otros no pertenecientes a la librería estándar de Python. <<PyPNG>>[15] es uno de ellos. Este módulo permite leer y escribir imágenes en formato PNG y está escrito todo en el lenguaje de programación Python. Por ello, se ha descargado e incluido en el proyecto como si de un módulo propio se tratase. No ha sido editado. Es utilizado en el módulo <<heighfield>>.

El resto de módulos utilizados han sido incluidos utilizando la herramienta PIP[16]. PIP es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Estos módulos han de ser descargados mediante esta herramienta y posteriormente importados en cada fichero Python en el que se va a utilizar. Estos módulos son:

- *numpy[17]:* módulo que agrega un gran soporte para operar con matrices y vectores. Es importado en los siguientes módulos: <<load_info>>, <<read_lidar>>.
- *laspy[18]:* módulo que permite leer, modificar y crear ficheros LiDAR con formato LAS. Es importado en los siguientes módulos: <<load_info>>, <<read_lidar>>.
- *PIL[19]:* módulo que añade funcionalidades de procesado de imágenes. Es importado en los siguientes módulos: <<load_info>>, <<povray_writer>>.

3.4.2. Programas

El programa que ha sido absolutamente necesario para el desarrollo de este proyecto, ya que ha sido utilizado en todo momento, ha sido Sublime Text[20]. Sublime Text es un editor de texto y código que puede ser configurado de forma que resalte palabras reservadas de Python o incluso tabule correctamente, además de proporcionar otras funcionalidades interesantes como la búsqueda de palabras. Ha sido utilizado tanto para el desarrollo del código del programa, como para visualizar el interior de alguno de los ficheros de entrada y de salida (ficheros POV).

Otro programa muy usado ha sido POV-Ray. POV-Ray es un programa de ray-tracing. Ha sido utilizado para renderizar las escenas descritas en los ficheros con extensión .POV. En estos ficheros se han de especificar desde la posición de la cámara, a los objetos a crear, pasando por la fuente de luz.

ImageMagick, un conjunto de utilidades de línea de comandos para manipular imágenes ha sido utilizada en varios aspectos. Quizás, el más importante, para teselar (partir en pequeños trozos) la escena renderizada a partir del fichero .POV. También se ha utilizado en algunos aspectos menores en el tratamiento de los datos de entrada.

Para reproyectar los datos de entrada (se explica más adelante) ha sido necesario el programa GDAL Warp[21].

Laszip[22] es un programa que se ha utilizado para transformar (descomprimir) los ficheros LiDAR en formato LAZ a fichero en formato LAS que pueden ser leídos.

Evidentemente también ha sido necesario el intérprete de Python y un navegador con el que se han descargado los datos de entrada y se ha visualizado el resultado del proyecto.

3.5. TRATAMIENTO DE DATOS

Para los datos de entrada ha sido necesario realizar algún tratamiento con ellos, ya sea durante la ejecución del programa (mediante los filtros) o de forma aparte sin ser ejecutado el programa. A continuación, se especifica que se ha tenido que realizar con cada tipo de fichero.

3.5.1. Modelo digital del terreno (MDT)

Como ya se ha explicado con anterioridad en este documento, estos archivos vienen en formato .ASC que el programa POV-Ray no soporta. Para ello se ha tratado de transformar estos ficheros ASCII en imágenes con formato PNG. Como ya se ha comentado, esta transformación se realiza durante la ejecución del programa si se incluye la opción necesaria para ello.

A partir de los valores que se obtienen del fichero y la altura límite establecida a la hora de ejecutar el programa (por defecto 2200), se da un valor desde 0 a 65535, que es el valor máximo

para imágenes PNG en escala de grises de 16 bits, al píxel correspondiente. Una vez se tiene la matriz con todos los valores de los píxeles, se genera la imagen PNG utilizando la biblioteca de Python PyPNG. Además, los datos de cabecera donde se especifican las coordenadas del MDT y su escala se vuelcan en un fichero de texto con el mismo nombre. Posteriormente, este fichero de texto es utilizado en el proceso de carga de información para incluir los datos contenidos en él en el fichero de texto con los datos de todos los MDTs.

Los ficheros MDT en formato PNG son utilizados por el programa POV-Ray para generar los objetos heightfield, los cuales dotan de volumen a la escena. POV-Ray es capaz de transformar estas imágenes en un objeto tridimensional utilizando como altura (tercera dimensión, las otras dos son el ancho y el largo) el valor de cada píxel. Así un píxel más claro representa mayor altura y uno más oscuro una altura menor.

3.5.2. Ortofotos

Para algunas de las ortofotos ha sido necesario un tratamiento fuera de la ejecución del programa. Esto es a causa de que algunas de ellas (las más al este) se encontraban en una proyección diferente a la utilizada en el resto de ficheros. Por ello ha sido necesario utilizar el programa GDAL Warp para reproyectar estas ortofotos a la proyección deseada. A partir de la imagen JPG y el fichero JGW donde se encuentra la información geográfica, utilizando este programa se genera una nueva imagen en formato TIF y un nuevo fichero con información geográfica en formato TFW. La instrucción necesaria para realizar esto ha sido:

gdalwarp -s_srs EPSG:25831 -t_srs EPSG:25830 -of GTiff -dstalpha -co "TFW=YES" in.jpg out.tif

Al reproyectar, la imagen es rotada para adecuarla a la nueva proyección y por ello es necesario utilizar un formato que permita que zonas sin datos sean transparentes (cosa que JPG no puede). A pesar de que POV-Ray soporta ficheros con formato TIF, estos no daban el resultado esperado (había una variación en los colores originales de la ortofoto) así que ha sido necesario convertir esta imagen de formato TIF a formato PNG que también soporta zonas transparentes. El fichero TFW es similar al JGW y tan solo se modifica su formato (cambiándolo manualmente) a JGW para adecuarlo a nuestro programa.

A la hora de añadir las ortofotos necesarias al fichero POV, no cambia nada ya que en el fichero de datos de las ortofotos se incluye el nombre del directorio y no el formato de la imagen. Son añadidos a la textura de su objeto heighfield correspondiente. El programa soporta tanto JPGs como PNGs.

3.5.3. LiDAR

Se ha creado un fichero de texto (areas_interest.txt) donde se han especificado zonas de interés donde se pueda usar los ficheros LiDAR. En casi todas las zonas se ha optado por no utilizar LiDAR

ya que se considera que con los heightfields creados a partir de los MDT y ortofotos es suficiente, ya que aportan a la escena el volumen suficiente para dotar esa apariencia 3D. En otras zonas (como las ciudades), los MDT no son capaces de otorgar ese volumen que, con los datos LiDAR sí es posible.

Para leer estos ficheros en formato LAZ es necesario primero descomprimirlos (pasarlos a formato LAS) lo cual se realiza durante la ejecución normal del programa. Una vez descomprimidos, se comprueba que los datos estén dentro de nuestras zonas de interés y si es así, se genera un objeto esfera para cada dato utilizando sus valores geográficos y de color. La conjunción de estas esferas son las que al ser renderizadas utilizando el programa POV-Ray generan objetos (edificios, bosques...) muy precisos en lo que altura se refiere.

No todas las esferas son renderizadas (se toman un tercio de todas las que están dentro de nuestra zona de interés) ya que la inclusión de demasiadas en según qué zonas solo hace que distorsionar la imagen y no queda tan bien. A pesar de ello, en otras zonas, los ficheros no poseen la suficiente densidad de puntos, lo que hace que, aunque se rendericen todas las esferas resultantes, no dan de forma clara volumen a la zona, quedando una escena con muchas esferas, pero no las suficientes para representar con precisión la zona. Se ha intentado minimizar esto último lo máximo posible ajustando las zonas de interés.

Estos objetos esferas se añaden al fichero POV como si de otro objeto de la escena (heightfield) se tratase.

3.6. DISEÑO DEL VISUALIZADOR

Para mostrar los mapas obtenidos una vez ya teselados se ha diseñado un visualizador simple siguiendo el patrón cliente/servidor. Como se puede ver en la Figura 6, el sistema diseñado es bastante sencillo donde en la máquina servidor se encuentran las teselas y el servidor web desplegado a partir del módulo de Python <<HTTPServer>>[23]. Este módulo es suficiente para desplegar el servidor que necesitamos para mostrar las teselas.

Al tratarse de un cliente sencillo, se ha optado por incluir todo lo relacionado con la vista en un solo fichero (<<index.html>>). En este fichero están definidos todos los parámetros necesarios para mostrar los mapas, no sin antes realizar una comunicación con el servidor donde se encuentran los ficheros necesarios para hacer funcionar OpenLayers[24] en el cliente. OpenLayers es una biblioteca JavaScript que permite mostrar mapas interactivos en los navegadores web. Al definir en el fichero <<index.html>> diferentes capas para la zona a mostrar, la biblioteca de OpenLayers[25] se encarga de tomar las teselas necesarias que se han de mostrar en el cliente. Además, ofrece una serie de controles para el nivel de zoom y la capa que se desea que se muestre. Mediante una conexión HTTP la máquina cliente se comunica con la máquina donde se encuentran las teselas y obtiene las que son necesarias en ese momento.

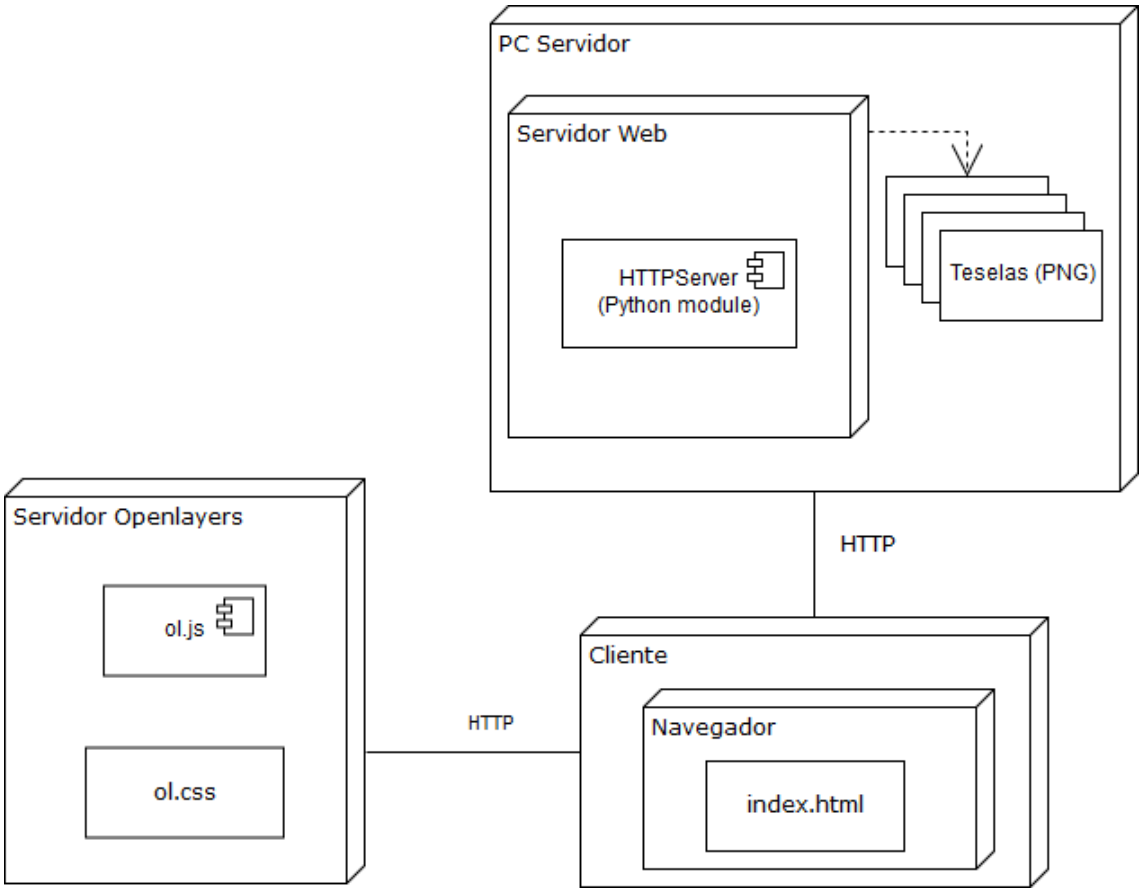

Figura 6: Diagrama despliegue visualizador

La estructura que se ha seguido para almacenar las teselas ha sido por directorios. Así, en el directorio donde se encuentra el fichero HTML, se incluyen un directorio más por punto de vista según el ángulo (es decir, dos directorios más 45 y 30). Dentro de estos directorios se encuentran cuatro directorios adicionales, uno para cada punto de vista según la dirección (N, S, E y W) y dentro de estos uno para cada nivel de zoom generado (7, 8, 9...). Dentro de estos últimos se encuentran las teselas con una estructura de nombre de "map_x_y.png", donde X e Y son los códigos de la tesela según su posición respecto al eje X y al Y respectivamente. Por ejemplo, la tesela más al sureste (la última, la de más abajo a la derecha) para un nivel de zoom de 7 tendrá un nombre igual a "map_127_127.png".

La máquina encargada de conectarse al servidor (máquina cliente) solo necesita de cualquier navegador para acceder a la web y poder interaccionar con ella para mostrar la zona del mapa de Aragón que desee y a varios niveles de zoom.

4. IMPLEMENTACIÓN

4.1. ASPECTOS TÉCNICOS

Dada la cantidad de bibliotecas libres que dispone y de tratarse de un lenguaje adecuado para la manipulación de datos y la automatización de procesos, se ha decidido que el código se desarrolle en el lenguaje de programación Python en su versión 3.

Se ha decidido dividir el código desarrollado según las funciones que realizan, por ello, cada funcionalidad altamente diferenciadora es representada por un fichero Python diferente (salvo el renderizador y teselado que se encuentran en el fichero del programa principal) de esta forma se dota al sistema de un diseño más modular y claro para el desarrollador. El programa principal hace de coordinador entre funciones del resto de ficheros y por lo tanto también entre filtros, salvo en los dos casos comentados. Se ha considerado que al tratarse de las dos funciones clave del sistema (al fin y al cabo, el fin es renderizar mapas y teselarlos), tanto la función de renderizado como el teselado deben permanecer al fichero principal. Además, estas funciones se "nutren" de ejecutar otros programas (POV-Ray en el caso del renderizado, ImageMagick en el caso del teselado), su función principal sigue siendo la de hacer de tubería entre los datos obtenidos del resto de filtros y el programa que renderiza/tesela.

Un aspecto interesante del sistema es el cómo se ha calculado la posición de la cámara que se incluye en el fichero POV, que posteriormente es renderizado por el programa POV-Ray, y que da esa sensación de perspectiva. Antes, hay que explicar que parámetros de la cámara posee este programa y cómo deben utilizarse para calcular la posición de esta. A continuación, se presenta una imagen (Figura 7) descargada directamente de la documentación de POV-Ray[26] para apoyar las explicaciones.

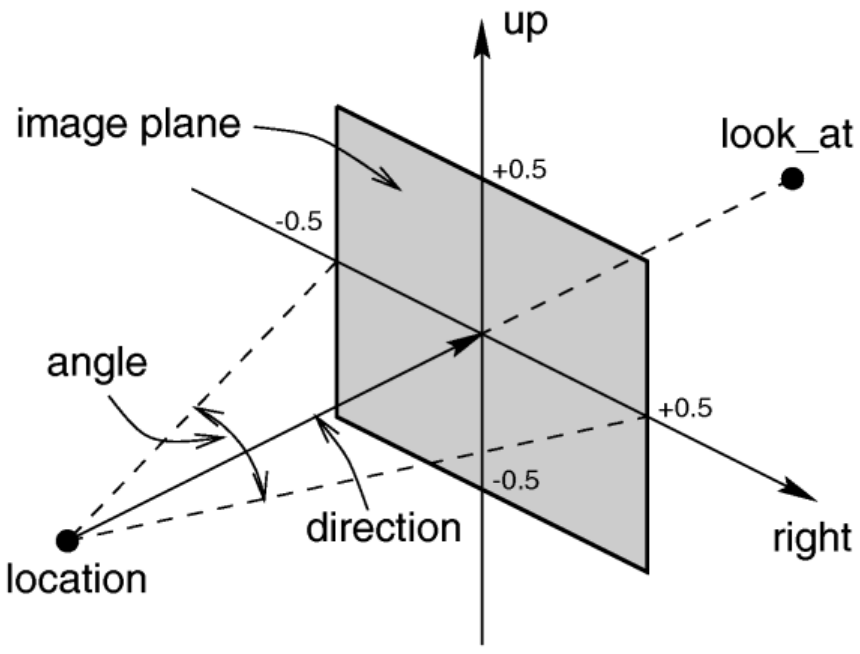

Figura 7: Elementos de la cámara en POV-Ray

El tipo de la cámara siempre va a ser ortográfica (orthographic en POV-Ray). Este tipo de cámara utiliza rayos paralelos para crear la escena y utiliza los vectores <<right>> y <<up>> para determinar el área que se va a visualizar. <<location>> y <<look_at>> son especificados para determinar la posición de la cámara. El vector <<location>> es donde se sitúa la cámara y el vector <<look_at>> es el punto hacia donde debe estar orientada la cámara. Los demás parámetros (<<angle>> y <<direction>>) no son utilizados en este tipo de cámara.

Según la dirección de vista de la escena (norte, sur, este u oeste), una serie de datos se aplican de una forma u otra[27]. Estos datos son: el ancho de la escena (max coordenada X – min coordenada X), el largo de la escena (max coordenada Y – min coordenada Y) y un valor offset (calculado como el valor de la altura de la cámara sin contar la altura del lookAt (10.000), dividido para el valor de la tangente del ángulo especificado a la hora de ejecutar el programa). Este offset y el valor de look_at es el que da a la escena una vista con algo de ángulo (perspectiva) y no perpendicular a la escena. Para especificar la primera y tercera coordenada de <<location>>, se toman las coordenadas X e Y respectivamente del centro de la escena a renderizar y se aplica el offset comentado con anterioridad, restándoselo a la tercera coordenada, en caso de estar en modo norte; sumándoselo a la tercera coordenada, en caso de estar en modo sur; restándoselo a la primera coordenada, en caso de estar en modo este; y sumándoselo a la primera coordenada, en caso de estar en modo oeste. La segunda coordenada es siempre la misma y se especifica como la altura de la cámara, contando la altura del lookAt, (11.200 en total).

El vector <<lookAt>>, que siempre debe ser colocado en última posición en la definición de la cámara para que posteriores inclusiones no puedan modificarlo, toma como primera y tercera coordenadas el centro de la escena, y como segunda coordenada la altura la especificada con anterioridad (1.200).

Para el vector <<right>> solo se incluye la primera coordenada (el resto 0) y esta primera coordenada es igual al ancho de la escena, en caso de estar en modo norte o sur; o igual al largo de la escena, en caso de estar en modo este u oeste.

Se incluye un vector inicial <<up>> con la primera coordenada igual a 0, la segunda igual a la tangente del ángulo y la tercera igual a 1. Posteriormente, este vector es normalizado, multiplicado por el largo de la escena, en caso de estar en modo norte o sur; o multiplicado por el ancho, en caso de estar en modo este u oeste y por último multiplicado por el seno del ángulo de la escena.

Estos son los valores que se incluyen en el objeto camera en el fichero POV aunque también es utilizado un último valor que ya ha sido comentado, el ratio de aspecto, calculado como el ancho, en caso de estar en modo norte o sur; o el largo, en caso de estar en modo este u oeste, dividido para la longitud del vector <<up>>. Este ratio de aspecto es utilizado posteriormente para renderizar la escena.

Además de esto, no se incluye nada especial en el código desarrollado, ya que el proyecto posee mucho más trabajo en tratamiento de datos, que en algoritmos especiales o complicados.

4.2. PROBLEMAS ENCONTRADOS

A lo largo del desarrollo del proyecto han ido sucediendo varios problemas. A continuación, se comentan los más importantes.

Quizás, el mayor problema encontrado durante el desarrollo del proyecto, y que aún sigue presente, es la cantidad de datos de entrada que hay para representar todo Aragón y que a causa de la cantidad de datos y la gran calidad que poseen, hace que los ficheros sean muy grandes y ya no solo por el espacio que ocupan, sino que, a la hora de renderizar, el programa POV-Ray necesite de una gran cantidad de memoria. Esto hace que el proceso de renderizado se ralentice muchísimo. Estos datos de espacio y tiempo de renderizado serán comentados en el siguiente apartado (sección 4.3 de Medidas de prestaciones).

Al inicio del proyecto, el primer problema "gordo" que hubo, además de generar una cámara en una posición y con una vista que proporcionase una visión isométrica de la escena, fue al pasar de renderizar una zona pequeña (incluyendo solo una ortofoto) a otra más grande. El poder incluir varios heightfield y varias ortofotos fue fácilmente solucionable, pero a la hora de unir dos zonas renderizadas, teóricamente adyacentes, las dos imágenes no casaban bien. Al estar creados en exceso (dos ficheros de zonas adyacentes comparten una pequeña área), tanto MDTs como ortofotos, el unir dos imágenes ya renderizadas se convertía en una ardua tarea al intentar calcular cuánto exceso tenía cada imagen para recortarla y casase correctamente con la otra imagen. Esto se solucionó creando escenas a renderizar a partir de las coordenadas de dos esquinas. Se incluían todos los objetos heightfield (con sus ortofotos) en la escena (a pesar de que podía haber partes que no se veían a causa de posición de la cámara) y se renderizaba de forma que la cámara en su posición limitaba la escena al rectángulo formado por las dos esquinas. Si se quería la zona adyacente, se debía introducir coordenadas de aquella zona adyacente (por ejemplo, si para el eje X se había renderizado de la coordenada 2 a la 5 y se quería la zona al este de la anterior, la mínima coordenada para el eje X debía ser 5), y si posteriormente se quería unir con la imagen renderizada con anterioridad, ahora casaban correctamente (siempre y cuando las coordenadas estuviesen bien especificadas). Esta solución sigue utilizándose como la ejecución normal del programa (sin opción teselas o todo). El último quebradero de cabeza que hubo relacionado con esto fue que a pesar de que imágenes de renderizaciones diferentes casaban bien en cuanto a zona, aun así, el aspecto (sombras, color...) era bastante diferente. Esto era a causa de que para cada escena a renderizar se generaba una fuente de luz diferente y proyectaba las sombras incorrectamente. Simplemente se solucionó incluyendo en todas las escenas la misma fuente de luz, posicionada al noreste de Aragón y a una altura considerable.

Otro problema, o más que problema sería elección, con el que hubo que lidiar fue con especificar la exageración vertical, esto es, el valor con el que se escala la altura del objeto heightfield generado a partir de un fichero MDT. Este valor se suele exagerar en estos sistemas para que se diferencien bien las alturas y uno de los problemas era encontrar ese valor que aportase esta exageración sin ser excesivo. En la Figura 8 se pueden observar los resultados de dos valores menores al utilizado actualmente. En la Figura 9, por el contrario, se observa el resultado para un valor mayor al actual.

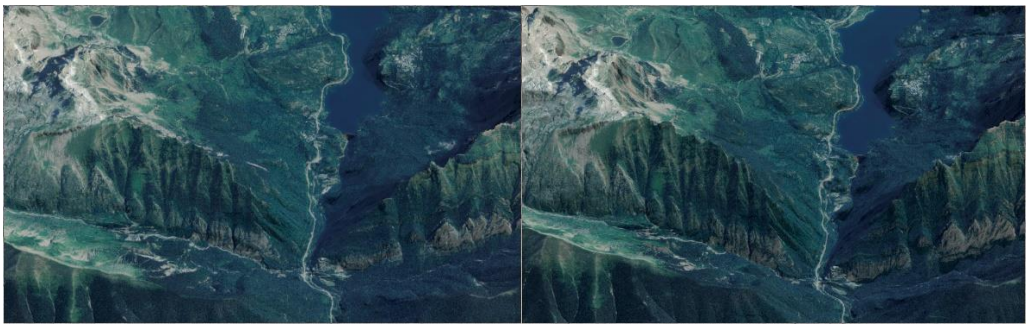

Figura 8: Zona montañosa con exageración vertical de 2000 (izquierda) y 3000 (derecha)

Esto se testeó en varias zonas y al final se optó por un valor de exageración vertical de 4000 ya que se consideró que menor valor no era suficiente y mayor era demasiado, originando un aspecto ideal con el valor de 4000. En la Figura X en la sección X (Resultados) se puede observar un ejemplo de la misma zona para un valor de exageración vertical de 4000.

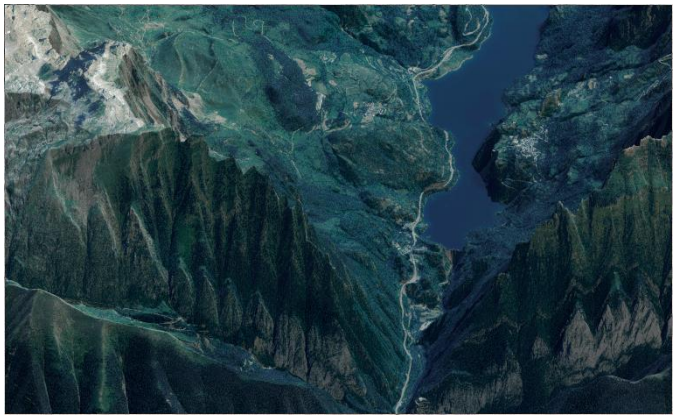

Figura 9: Zona montañosa con exageración vertical de 5000

El siguiente problema importante que hubo que solventar fue el esquema de teselado que se iba a seguir. Principalmente, se iba a seguir el esquema de nombrado que utiliza OpenStreetMaps[28], pero a causa de que las teselas generadas por el programa no son cuadradas (para dar esa sensación de perspectiva, se debe respetar el ratio de aspecto de la escena) o las teselas se modificaban (perdiendo su ratio de aspecto y por lo tanto la sensación de perspectiva) o era complicado renderizar zonas adyacentes a causa de que se producían saltos de nombre (se pasaba por ejemplo de la tesela 5 a la 8) ocasionando zonas sin teselas entre ambas. Tras varios intentos de trabajar con este esquema, se descartó definitivamente y se decidió crear un sistema propio de nombrado para las teselas. Se generó una extensión límite cuadrada, que incluyese todo Aragón, para generar las teselas, pero en el cliente que se encarga de mostrarlas, hubo que acortar esta extensión límite por una dimensión, para que se respetase el ratio de aspecto y las teselas siguiesen dando sensación de perspectiva, además de especificar el tamaño de estas. Relacionado con esto, hubo que añadir modificaciones al programa para que renderizase siempre teselas completas (aunque las coordenadas partiesen una por la mitad), para evitar problemas de continuidad y que, por ejemplo, la tesela 5 se correspondiese a la misma zona; y también se renderizasen el mismo número de teselas tanto para el eje X como para el Y, así evitar que escenas a generar más anchas que largas, modificase el ratio de aspecto.

Al tratar con los datos LiDAR hubo también algunos problemas con los que tratar. El principal era el cómo representar los puntos que eran interesantes. Actualmente, se utilizan objetos esfera que tienen volumen, y la unión de muchos de estos objetos son los que aportan a la escena de volumen y sensación 3D (en zonas urbanas principalmente), pero anteriormente se probaron otras soluciones. La idea principal al empezar a trabajar con estos datos era utilizar splats, es decir, elipses (discos), pero solo aquellos que estuviesen apuntando a la cámara, de esta forma se podía conseguir dotar a la escena de volumen utilizando muchos menos objetos y por lo tanto de manera más eficiente. Esta orientación se podía comprobar calculando el vector normal de un punto a partir de los vectores a dos puntos cercanos y no colineales. A partir del vector de orientación de la cámara y estos vectores normales de cada punto se podía comprobar si estos puntos se veían (orientados hacia la cámara) o por contra no se debían ver (de espaldas a la cámara). Como se puede ver en la Figura 10, no se consiguieron resultados que mejorasen al que se podía conseguir con esferas en lo que aspecto se refiere ni siquiera probando varias combinaciones diferentes de opciones con los splats (diferente tamaño, incluir aquellos casi perpendiculares a la orientación de la cámara...). Quizás, con mayor tiempo para realizar cálculos y procesamiento sobre estos puntos se podía haber conseguido un mejor resultado, pero como objetivo de este proyecto no se contaba con realizar tanto procesamiento sobre estos datos, por eso se eligió utilizar esferas a costa de una peor eficiencia a la hora de renderizar.

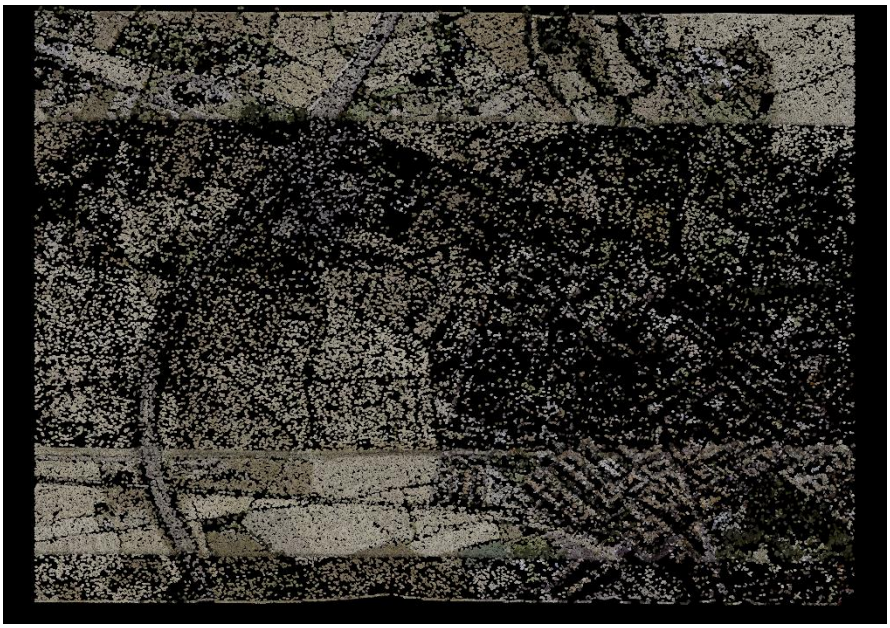

Figura 10: Zona urbana con puntos LiDAR en forma de discos (sin heightfields)

De esta forma no se generaban escenas con suficiente densidad de puntos y en las zonas en las que había mayor densidad, la forma de estos objetos no quedaba excesivamente bien. En la Figura 21 en la Sección 5.1 de resultados se puede ver la solución por la que se ha optado (con objetos esfera).

El siguiente problema se encontró al renderizar sobre todo zonas altas y próximas a cambios de heightfields. En la parte de debajo de la escena se renderizaba una zona negra. Esto era a causa de que la zona que en teoría queda fuera de la zona a renderizar, al ser tan alta, se interponía entre la cámara y la escena, generando esa zona negra. Se solucionó incluyendo heightfields

adyacentes mediante una inclusión al fichero POV de objetos en exceso, aunque no fuesen a ser mostrados en la escena final.

El último problema por comentar se produjo cuando hubo que renderizar todo Aragón. Algunas de las ortofotos más al estaban disponibles en una proyección diferente al resto. La solución a esto ya ha sido comentada con anterioridad.

Ha habido más problemas durante la realización de este proyecto, pero se han incluido solo los que se han considerado mayores, ya sea por el tiempo que costó solucionarlos (si han sido solucionados) o por la situación crítica que causaban.

4.3. MEDIDAS DE PRESTACIONES

Como se ha comentado, existe una gran cantidad de datos de entrada para renderizar Aragón. Esto ralentiza mucho el renderizar todo el mapa en sus diferentes puntos de vista y niveles de zoom. A continuación, se presenta la cantidad de ficheros y el espacio que ocupan de cada tipo de fichero utilizado por el programa.

Para generar los objetos heightfield de todo Aragón, se han descargado: 125 ficheros MDT (con formato ASC) que ocupan 20,2 GB, que al ser transformados en imágenes (con formato PNG) ocupan 3,84 GB adicionales; 1743 ficheros ortofotos (incluyendo en cada uno foto y fichero con información geográfica) que ocupan 260 GB; y, por último, 28 (solo se han descargado los que se incluyesen en zonas de interés) archivos LiDAR (con formato LAZ) que ocupan 0,5 GB. Es decir, un total cercano a 300 GB de datos de entrada. De aquí se puede extraer el por qué lleva tanto tiempo renderizar todo el mapa, más aún, si hay que hacerlo para diferentes puntos de vista.

Para toda España, contando las islas y las ciudades autónomas de Ceuta y Melilla, aproximadamente habría que generar cerca de 1.800 objetos heightfield, es decir, 14,4 veces Aragón. Esto traducido en espacio ocupado serían 4.176 GB (más de 4 TB) de información de los objetos heightfield y sin contar los ficheros LiDAR que serían necesarios. Haría falta una gran base de datos para almacenar todo esto (que evidentemente no entra dentro de los objetivos de este proyecto) y también los resultados generados.

Como se puede ver en la tabla de la Figura 11, todo Aragón renderizado para solo un punto de vista y un nivel de zoom (concretamente nivel de zoom 9 con una resolución de 4,5 metros por pixel) genera un total de 49.432 teselas que ocupan cerca de 3,5 GB. Un nivel de zoom mayor equivale a el doble de teselas para cada dimensión (cada tesela abarca la mitad del territorio), es decir, el número de teselas es cuatro veces mayor y por lo tanto ocupa cuatro veces más y un nivel de zoom menor cuatro veces menos, es decir, que para un punto de vista (por ejemplo, norte y 45º) y niveles de zoom de 5 a 12 se genera un número de teselas cercano a 4.220.000 ocupando aproximadamente 291 GB. Si esto lo multiplicamos por cuatro puntos cardinales y por dos ángulos diferentes (45º y 30º), es decir, por 8 puntos de vista diferentes, da un total de 33.760.000 teselas y 2.400 GB (2,4 TB) que sumados a los datos de entrada daría una cifra cercana a los 3 TB (2,7 TB). Para calcular lo que ocuparía todo España, sería necesario multiplicar estas cifras por 14,4 (España según el número de ficheros de entrada es 14,4 veces mayor), 34,56 TB de teselas en total sería el espacio necesario para mostrar teselado todo España, 38 TB contando los datos de entrada.

Nivel de zoom	Resolución (m/pixel)	Nº teselas	Tamaño total (GB)
5	72	324	0,014
6	36	1.225	0,0556
7	18	4.761	0,218
8	9	12.358	0,859
9	4,5	49.432	3,41
10	2,25	197.728	13,64
11	1,125	790.912	54,56
12	0,5625	3.163.648	218,24

Figura 11: Tabla con datos de las teselas para todo Aragón

El tiempo de renderizado también ha sido muy alto, dadas también las características de la máquina con la que se ha renderizado (un portátil con 4 GB de RAM y procesador de dos núcleos (i3) a 2.40 GHz), donde aproximadamente renderizar todo Aragón para solo un punto de vista y a nivel de zoom de 9 es necesario del orden de cinco días y medio para renderizar todo. En la Figura 12 se puede ver una evolución del tiempo que tarda renderizar según el nivel de zoom.

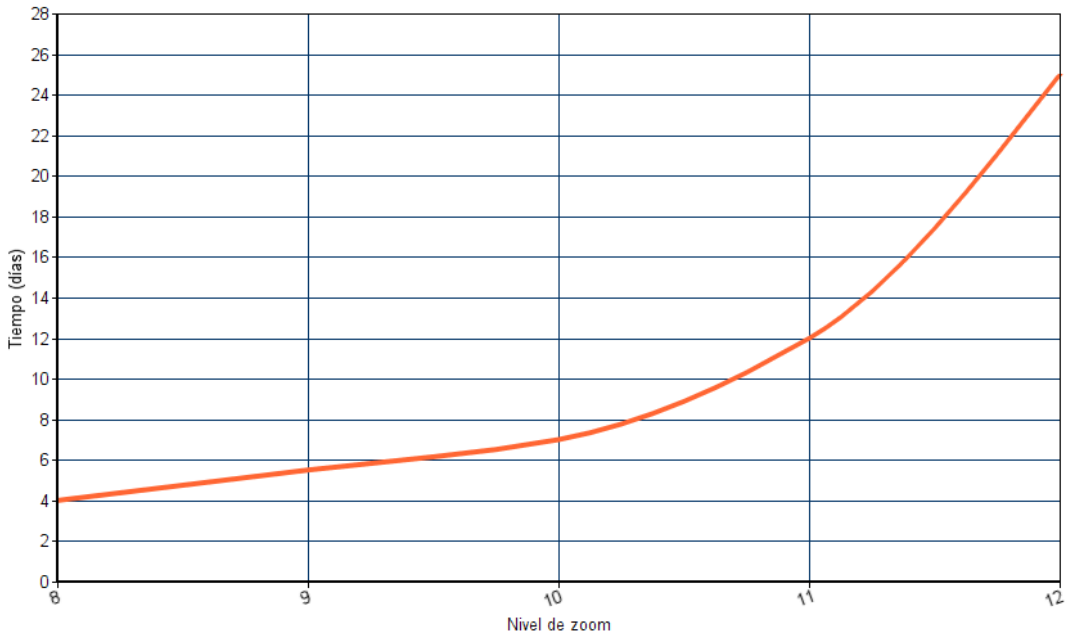

Figura 12: Gráfica del tiempo (días) que se tarda en renderizar todo Aragón según nivel de zoom

Un renderizado a nivel de zoom de 12, para posteriormente generar las teselas de zooms anteriores y no tener que renderizar varias veces, tardaría aproximadamente unos 25 días para solo un punto de vista, es decir para renderizar todo Aragón en todos sus puntos de vista (con la máquina actual) tardaría un total (aproximadamente) de 200 días, es decir, unos 6 meses y medio. Para toda España, cerca de 8 años como se puede ver en el gráfico de barras de la Figura 13.

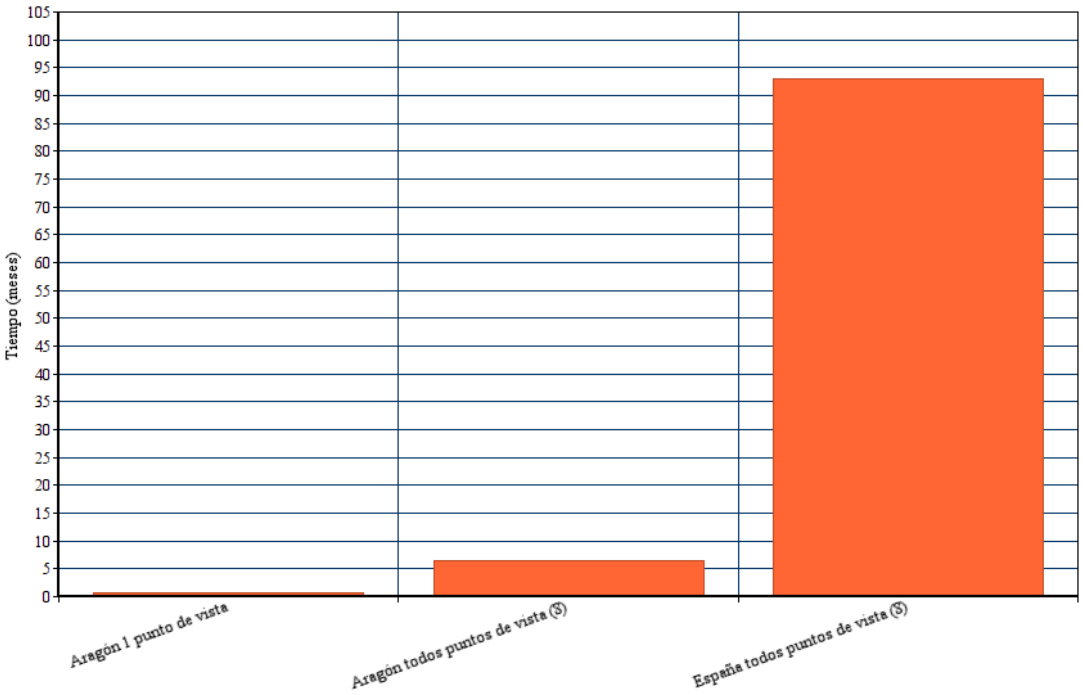

Figura 13: Gráfica barras con el Tiempo (meses) en renderizar según la extensión y puntos de vista

Para solo un punto de vista la barra casi no se ve, pero al tratar con una magnitud de meses, al ver sobretodo la barra de España, se puede comprobar como crecería en tiempo el renderizar toda España.

A partir de todas estas cifras, se puede comprobar la magnitud del proyecto realizado y la magnitud que puede llegar a tener hacerlo para toda España. Estas cifras tan poco eficientes y de dimensión tan alta para un ordenador personal como con el que se ha realizado este proyecto, hace que no se pueda incluir un mapa de Aragón para todos los puntos de vista posibles. Mínimo sería necesario una máquina con bastante más recursos (principalmente RAM), que fuese capaz de renderizar los resultados de forma mucho más rápida y que además fuese capaz de almacenar todos los resultados obtenidos. Por ello, se ha decidido realizar dos clientes diferentes, uno local donde se incluye todo Aragón para dos puntos de vista (45ºNorte y 45ºSur) y tres niveles de zoom; y un cliente disponible online con una zona mucho más pequeña, pero con todos los puntos de vista disponibles.

A pesar de estas cifras, el proceso es muy fácilmente paralelizable. Gracias a las diferentes opciones de renderizado que ofrece la aplicación, pueden estar varias máquinas al mismo tiempo renderizando escenas, lo que hace que el tiempo de renderizado disminuya mucho, más si estas máquinas son mejores que con la que se ha trabajado. Suponiendo que hay 444 columnas de teselas para solo un punto de vista y se poseen cuatro máquinas diferentes, cada máquina podría encargarse de 111 columnas de teselas (la primera de la 1 a la 110, la segunda de la 111 a la 221...) paralelizando el renderizado de un solo punto de vista y reduciendo 4 veces el tiempo necesario. Otra opción sería que cada máquina trabajase con un punto de vista diferente (la primera 45-N, la segunda 30-S...).

Podrían buscarse mayores formas de reducir el tiempo renderizado como afinando aún más el número de objetos heightfield que se pueden renderizar en cada escena, renderizando zonas más grandes o más pequeñas, no incluir LiDAR nunca... Para mejorar las prestaciones en caso de tratar con datos que cubrieran un mayor terreno sería necesario realizar un análisis, y actualizarlo periódicamente, de que soluciones hardware serían óptimas, o que alternativas software para el renderizado podrían mejorar las prestaciones.

5. RESULTADOS Y PRUEBAS

5.1. RESULTADOS

A continuación, se presentan algunos de los resultados obtenidos en ejecuciones del programa. Primero, se muestran algunas imágenes de los procesos intermedios antes de los mapas finales. En la Figura 14 se puede comprobar el aspecto de un MDT transformado en imagen con formato PNG para que posteriormente pueda ser leído por el programa POV-Ray.

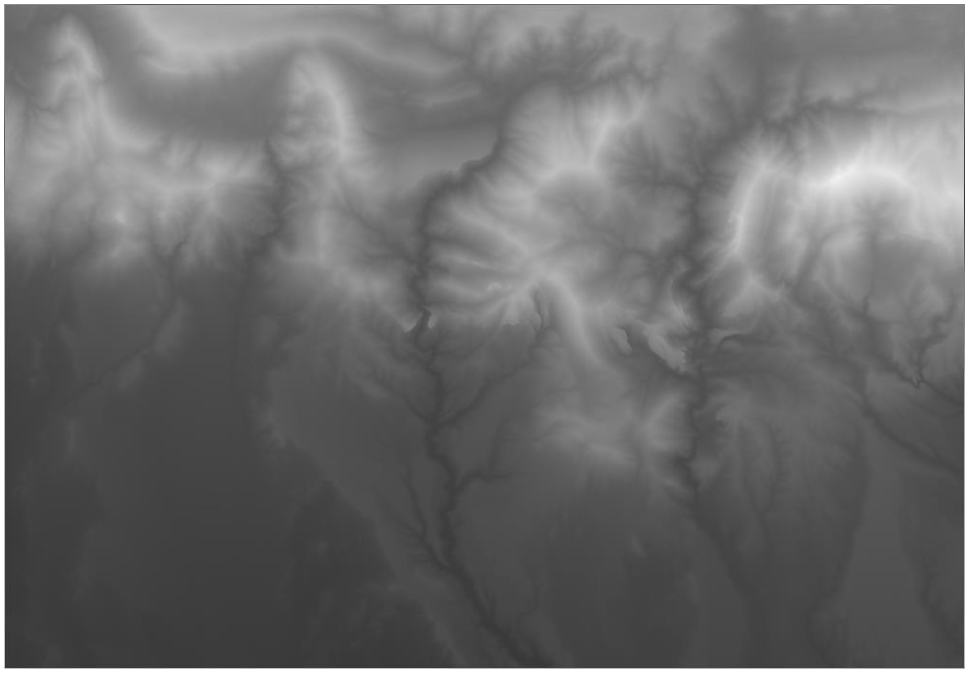

Figura 14: MDT en formato PNG

Las zonas más blancas corresponden a zonas más altas, así pues, esas líneas negras que se atisban en la imagen corresponden seguramente a ríos o pequeños valles donde la altura del terreno es menor.

La siguiente imagen corresponde a una ortofoto la cual ha sido necesaria una reproyección para adaptarla a la proyección que se usa en el proyecto. Como se puede comprobar en las 4 esquinas, existen zonas representadas por una trama a cuadrados grises. Estas zonas se han

generado al rotar la imagen para adaptarla a la proyección y representan zonas transparentes, para que, posteriormente, se adapten correctamente con el resto de ortofotos y no genere zonas negras.

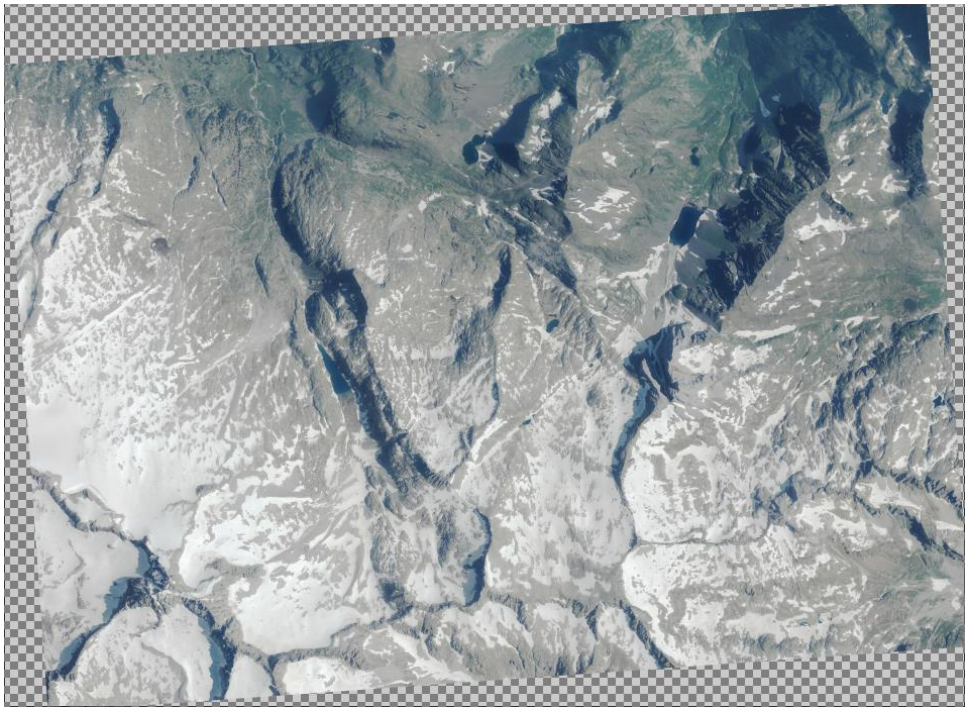

Figura 15: Ortofoto reproyectada con zonas transparentes

En la Figura 16 se puede ver un ejemplo de ejecución, en la que simplemente se va escribiendo que teselas se están renderizando y a que coordenadas corresponde y los procesos por los que pasa el programa (identificación tesela, lectura LiDAR y generación de esferas, creación del fichero POV, renderizado y teselado).

```
Rendering from tile [478, 456] to [481,459] with coordinates from [675250.431640625, 4618845.7890625] to [677555.380859375
Generating spheres...
Generating spheres from /media/pablo/280F8D1D0A5B8545/TFG_files/LiDAR/PNOA_2010_LOTE1_ARA-NORTE_676-4618_ORT-CLA-COL.LAZ
Reading all points...
Taking 1000000 points...
Generating spheres from /media/pablo/280F8D1D0A5B8545/TFG_files/LiDAR/PNOA_2010_LOTE1_ARA-NORTE_674-4618_ORT-CLA-COL.LAZ
Reading all points...
Taking 530446 points...
Generating pov-ray file...
Rendering ./result.png
Creating tiles from [478, 456]...
```

Figura 16: Ejemplo ejecución programa

Solo se ha decidido incluir en este apartado una pequeña muestra de uno de los ficheros que se generan cuando se carga la información de los datos de entrada. Concretamente, la Figura 17 corresponde a un pequeño pedazo del fichero “orto_data.txt” encargado de almacenar la información de las ortofotos disponibles para el sistema.

```
PNOA_MDT05_ETRS89_HU30_0521_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0521_LID/pnoa_2015_25831_0521_1_1.
jpg 0.5000985084 -0.5000985084 764996.0484492846 4525610.4471459826 15418 11106
PNOA_MDT05_ETRS89_HU30_0521_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0521_LID/pnoa_2015_25831_0521_1_2.
jpg 0.5000986542 -0.5000986542 765162.5602971179 4520990.5792112257 15439 11127
PNOA_MDT05_ETRS89_HU30_0521_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0521_LID/pnoa_2015_25831_0521_1_3.
jpg 0.5000987681 -0.5000987681 765329.4897439278 4516360.0367180835 15437 11106
PNOA_MDT05_ETRS89_HU30_0521_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0521_LID/pnoa_2015_25831_0521_2_1.
jpg 0.5001423011 -0.5001423011 772025.9789481468 4525872.3107589865 15418 11106
PNOA_MDT05_ETRS89_HU30_0521_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0521_LID/pnoa_2015_25831_0521_2_2.
jpg 0.5001424747 -0.5001424747 772193.1915935054 4521242.0585077237 15436 11087
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_2_3.
jpg 0.5 -0.5 603920.25 4493439.75 14660 9880
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_2_4.
jpg 0.5 -0.5 603980.25 4488809.75 14680 9860
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_3_2.
jpg 0.5 -0.5 610900.25 4498169.75 14660 9880
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_3_3.
jpg 0.5 -0.5 610970.25 4493549.75 14680 9900
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_3_4.
jpg 0.5 -0.5 611040.25 4488919.75 14680 9880
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_4_1.
jpg 0.5 -0.5 617880.25 4502909.75 14660 9900
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_4_2.
jpg 0.5 -0.5 617950.25 4498289.75 14680 9900
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_4_3.
jpg 0.5 -0.5 618020.25 4493659.75 14700 9900
PNOA_MDT05_ETRS89_HU30_0540_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0540_LID/pnoa_2015_25830_0540_4_4.
jpg 0.5 -0.5 618100.25 4489039.75 14680 9900
PNOA_MDT05_ETRS89_HU30_0541_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0541_LID/pnoa_2015_25830_0541_1_1.
jpg 0.5 -0.5 624920.25 4503029.75 14680 9900
PNOA_MDT05_ETRS89_HU30_0541_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0541_LID/pnoa_2015_25830_0541_1_2.
jpg 0.5 -0.5 625000.25 4498409.75 14680 9920
PNOA_MDT05_ETRS89_HU30_0541_LID /media/pablo/280F8D1D0A5B8545/TFG_files/PNOA/PNOA_MDT05_ETRS89_HU30_0541_LID/pnoa_2015_25830_0541_1_3.
jpg 0.5 -0.5 625080.25 4493779.75 14680 9900
```

Figura 17: orto_data.txt

Se muestra para cada fichero: MDT al que pertenece, ruta, tamaño pixel para ejes X e Y (dos valores), coordenadas de su esquina superior izquierda (dos valores) y tamaño de la imagen (dos valores). Algunos ficheros, como los primeros tienen valores de coordenadas y tamaño de pixel con mayor número de decimal, estos son los que han sido reproyectados y son los valores que dan sin manipular tras su reproyección.

A continuación, se muestran ya datos finales (renderizados e incluso teselados). Una tesela en nivel de zoom 8 que muestra una escena algo montañosa tiene este aspecto.

Figura 18: Tesela zoom 8 zona algo montañosa

Como se aprecia en la imagen, esta tesela, que abarca una zona de 2,3x2,3 km, ofrece una zona algo montañosa con la que se es capaz de visualizar el aspecto 3D que posee. La Figura 19 representa una zona de mayor extensión donde se puede comprobar lo anterior en una zona más grande y con la diferencia que hay entre una zona más llana (el lago) y zonas más altas (montañas).

Figura 19: Zona montañosa (45-N)

Zonas urbanas, como la de la Figura 20, se ven casi llanas pues a pesar de tener muchos cambios de altura (los edificios), estos nos son relevantes (no es tan alto un edificio como lo puede ser una montaña) y los MDT, como su propio nombre indica, realiza un modelado del terreno. En la Figura 21, está representada la misma zona, pero esta vez incluyendo las esferas generadas a partir de los ficheros LiDAR. Como se puede comprobar, gracias a estas esferas, la sensación de volumen en las ciudades aumenta. A cambio, el color de estas esferas y el color de las ortofotos no coinciden, y se generan esferas que no aportan demasiado a la escena, sino que más bien estorban.

Figura 20: Zona urbana sin LiDAR

Figura 21: Zona urbana con LiDAR

Se ha intentado ajustar lo máximo posible las esferas que se renderizan evitando estas esferas que no aportan demasiado, quitando las esferas que representan la zona del suelo. Por el contrario, en algunas zonas la densidad de puntos no es suficiente como para representar con precisión la escena. En la Figura 22 se puede ver un ejemplo de esto.

Figura 22: Zona urbana con LiDAR densidad pobre

Para nivel de zoom muy alto (nivel 12 donde aproximadamente cada píxel ocupa una zona de 0,5x0,5m) el LiDAR tampoco es muy buena solución, por lo menos con esta configuración que favorece el resto de zooms. En la Figura 23 se puede comprobar esto, donde a una distancia tan cercana, las esferas se diferencian entre ellas y, a pesar de que el efecto 3D sigue presente, el que se vean estas esferas empeora el resultado final.

Figura 23: Zona urbana con LiDAR a nivel de zoom 12

En la Figura 24 se representa el cliente web visualizando una zona de Aragón. Se pueden ver los botones que se han incluido para realizar los cambios de punto de vista. Por defecto, están activados norte (N) y 45º, pero si se pulsa un botón diferente, como, por ejemplo, sur (S), cambia el punto de vista dejando el sur arriba en el mapa como se puede comprobar en la Figura 25.

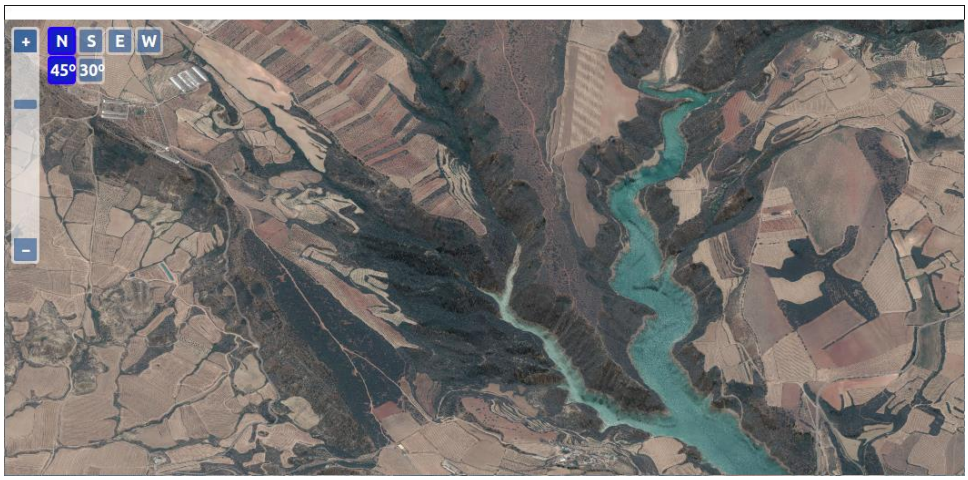

Figura 24: Cliente web en modo 45-N (zoom 9)

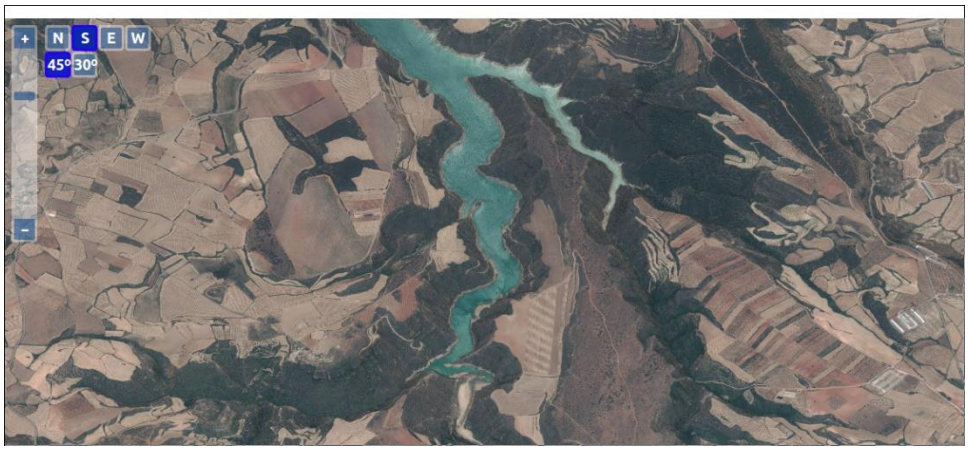

Figura 25: Cliente web en modo 45-S (zoom 9)

A la izquierda de los botones de cambio de punto de vista, se incluye el slider del zoom. Ya sea arrastrando este slider, o pulsando los botones "+" y "-" del zoom, el nivel de zoom cambia y se cargan las teselas correspondientes a ese zoom. En la Figura 26 se puede comprobar el resultado de presionar el botón menos a partir de la situación de la imagen anterior.

Figura 26: Cliente web en modo 45-S (zoom 8)

Se incluyen teselas que abarcan mayor espacio (una tesela en zoom 8 son 2x2 teselas en zoom 9), pero cada tesela sigue teniendo la misma resolución (256x181 en modo 45º).

Por último, se incluyen 3 imágenes similares a la de la Figura 19 pero en puntos de vista 45-E, 45-W y 30-N.

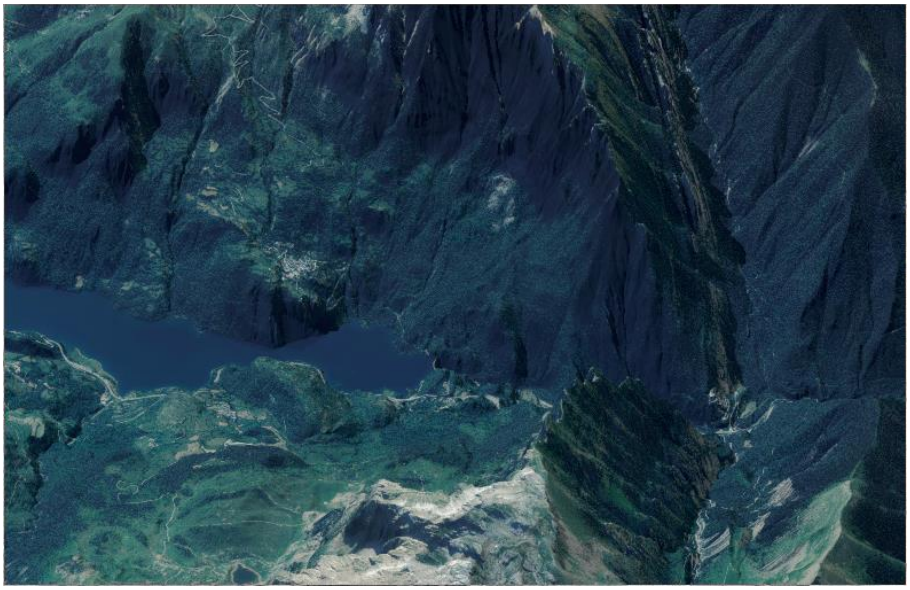

Figura 27: Zona montañosa (45-E)

En la Figura 27, se puede ver la primera de ellas, 45-E. Esta figura muestra una zona montañosa con ángulo de la cámara de 45º, con la dirección de vista hacia el este (el este está hacia arriba en la foto, norte a la izquierda...) y nivel de zoom 8. En la Figura 28 (45-W), se muestra la misma zona, pero con dirección de vista hacia el oeste (el oeste está hacia arriba, a la derecha el norte...).

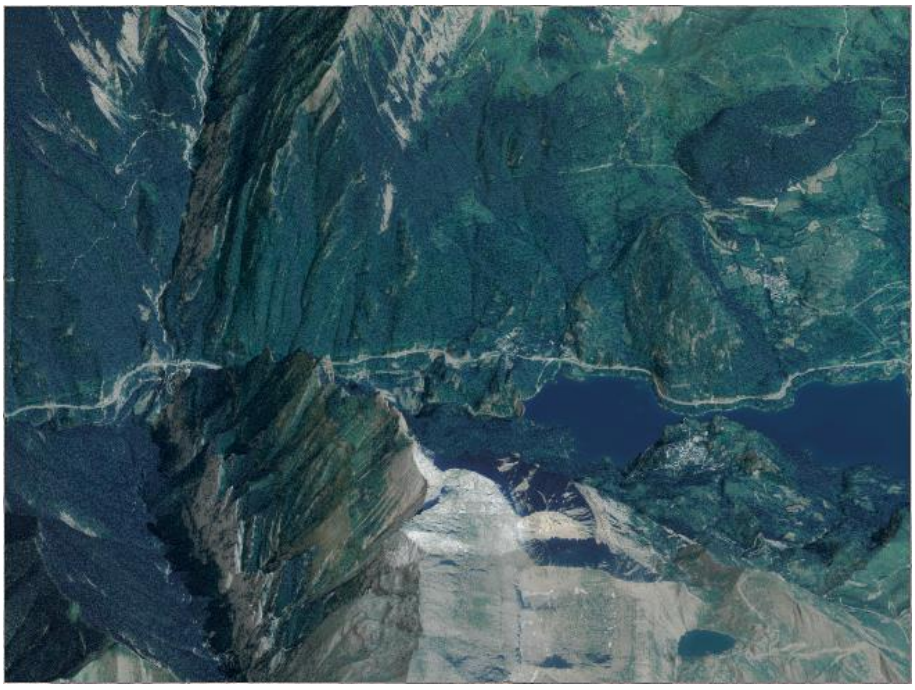

Figura 28: Zona montañosa (45-W)

Por último, la Figura 29 muestra un ejemplo con dirección de vista norte y nivel de zoom 8 de la misma zona, pero con ángulo de la cámara de 30º. Este ángulo dota a la escena de aun mayor perspectiva al no estar tan perpendicular al suelo. Por el contrario, parece que la zona es más pequeña, pero esto es a causa de que la vista al estar más paralela al suelo, y el intentar respetar el ratio de aspecto que otorga este ángulo, achata la imagen del mapa. Gracias a esto, se consigue una sensación de mayor volumen.

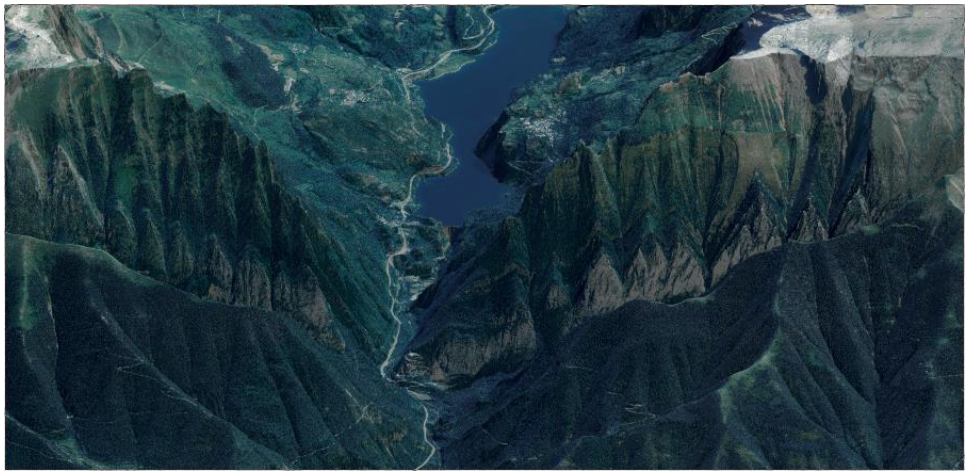

Figura 29: Zona montañosa (30-N)

Mediante esta serie de pequeños resultados obtenidos, se ha intentado dar una pequeña pincelada de lo que el programa consigue en sus ejecuciones. Gracias a estos resultados y al apartado anterior en esta memoria de Medidas de Prestaciones, se puede intuir el por qué es complicado conseguir un mapa muy extenso con muchos puntos de vista diferentes. Con el estado actual del proyecto, se ha querido aportar todo Aragón renderizado desde dos puntos de vista diferentes (norte y sur) y a unos pocos niveles de zoom, para poder visualizar la magnitud del proyecto en lo que extensión del mapa y gestión de espacio y tiempo se refiere. Además, con la inclusión del cliente alternativo online, poder mostrar todas las posibilidades actuales del programa realizado, en una zona pequeña.

5.2. POR HACER

A pesar de todo lo desarrollado durante el proyecto, hay cosas que se esperan hacer o mejorar en el futuro. La primera de ellas sería el renderizar todo Aragón para todos los puntos de vista disponibles además de aumentar el número de niveles zoom disponibles. Actualmente, solo está disponible Aragón tanto para dirección de vista norte como para sur en la versión local del proyecto; y una versión disponible online (https://strummertfiu.github.io/) donde se incluye una zona muy pequeña dentro de Aragón, pero con todos los puntos de vista disponibles, posibilitando consultar un pequeño ejemplo de lo que sería renderizarlo para todo Aragón. Esto es a causa de las limitaciones tanto de tiempo como espacio comentadas con anterioridad.

Una vez renderizado todo Aragón, se buscarán formas de mejorar el resultado de los datos obtenidos de los ficheros LiDAR, ya fuese mejorando la eficiencia usando menos esferas, pero colocadas de tal forma que diesen una mayor información; o generándolo en mayor número de zonas de forma eficiente.

Se seguirá mejorando el renderizado dando al usuario nuevas posibilidades, añadiendo aspectos diferentes a la escena según la situación de la fuente de luz (amanecer, atardecer...) o incluso incluyendo algún fenómeno meteorológico como niebla.

Añadir más puntos de vista (noreste, 20º de ángulo...) será otra forma de mejorar el proyecto y completarlo aún más.

Evidentemente, mejorar la eficiencia del proyecto ya sea paralelizando los diferentes procesos o adquiriendo mejores máquinas sería otro aspecto a mejorar para así renderizar de forma más rápida.

Y, por último, tener España al completo renderizado sería la situación final que se espera conseguir, pero como se ha comprobado con anterioridad, esto supondría una gran cantidad de espacio y tiempo para renderizar todo el país (en incluir los datos de entrada) que la máquina actual no puede aportar. Además, habrá que incluir todos los mapas renderizados en una plataforma online, para que estuviesen disponibles para todo el mundo.

5.3. PRUEBAS

Dada la naturaleza del proyecto, automatizar una serie de pruebas a realizar en todo el código no es sencillo. Por ello, se ha decidido incluir una serie de pruebas unitarias en algunos módulos y realizar una serie de pruebas sistemáticas sobre las mismas zonas.

5.3.1. Pruebas unitarias

Se ha decidido incluir pruebas unitarias en algunos de los módulos del proyecto. Estos módulos son: <<main_program>>, <<calculate_tile>>, <<cameraUtils>> y <<vector_XYZ>>. Estos módulos, realizan cálculos matemáticos o sencillamente devuelven objetos fáciles de comprobar automáticamente. El resto de módulos, necesitan o generan ficheros muy específicos (los MDT con formato ASC por ejemplo), lo que complica comprobar los resultados obtenidos de forma automática. En el módulo principal del proyecto (<<main_program>>) se han incluido pruebas unitarias, pero solo sobre funciones que no necesitan de ficheros externos o sencillamente no utilizan un programa externo que genera una salida particular (la función render además de generar el fichero de POV, invoca al programa POV-Ray para renderizar la escena, lo que hace que comprobar automáticamente esta función sea muy complicado).

Para realizar estas pruebas se ha utilizado el módulo doctest[29] incluido en la librería estándar de Python. Este módulo, permite incluir pruebas en el encabezado comentado con triples comillas de cada función. El módulo busca, en las zonas comentadas, trozos de texto similares a

una sesión interactiva de Python. Cuando encuentra estos trozos de texto ejecuta las sesiones para verificar que han funcionado como se muestra a continuación. Como se puede comprobar en la Figura 30, la cláusula ">>>" simula el intérprete de Python, y lo que hay justo debajo es el resultado esperado.

```
def calculate_tile(x, y, z):
    """
    Calculate tile number from the coordinates passed as parameter.

    Normal test
    >>> calculate_tile(650000, 4400000, 9)
    (217, 417)
```

Figura 30: Prueba doctest incluida en un módulo

Para ejecutar estas pruebas se utiliza una terminal en la que se ejecuta la instrucción:

python3 -m doctest

seguido del módulo a probar. Cada vez que se ha realizado algún cambio sobre este código, se han ejecutado las pruebas de módulo en módulo utilizando esta instrucción. Mediante un script que realizase esta instrucción sobre todos los módulos con pruebas doctest, automatizaría el proceso para todos los módulos al mismo tiempo en vez de ir de uno en uno (al fin y al cabo, es lo que hace el script pero sin la intervención del usuario). Si la ejecución ha sido correcta para todas las pruebas incluidas en un módulo, en la terminal no se muestra nada a no ser que se active el modo verbose (mediante la opción -v), con la que, como se puede comprobar en la Figura 31, genera una traza de los tests realizados y el resultado.

```
Trying:
    tile_to_west((600, 300), 9)
Expecting:
    'null'
ok
1 items had no tests:
    calculate_tile
8 items passed all tests:
   2 tests in calculate_tile.calculate_coordinates
   3 tests in calculate_tile.calculate_tile
   2 tests in calculate_tile.tile_from_east
   2 tests in calculate_tile.tile_from_south
   2 tests in calculate_tile.tile_from_west
   2 tests in calculate_tile.tile_to_east
   2 tests in calculate_tile.tile_to_south
   2 tests in calculate_tile.tile_to_west
17 tests in 9 items.
17 passed and 0 failed.
Test passed.
```

Figura 31: Traza pruebas doctest sobre un módulo (-v activado)

Todas las pruebas han sido correctas en la traza especificada, pero si alguna hubiese fallado, se mostraría que prueba ha fallado comparando el resultado esperado con el que se ha obtenido.

5.3.2. Pruebas sistemáticas

A la hora de comprobar si la ejecución del programa ha sido correcta y se renderiza la escena como se desea, principalmente se realizaron pruebas sobre las mismas zonas. Al principio, se probaba sobre una misma zona situada en una zona no urbana y sin mucha diferencia de altura. En cada ejecución, se comprobaba visualmente que se renderizaba toda la escena y que la unión de ortofotos y MDTs fuese la correcta y no se produjesen irregularidades, principalmente en la unión entre 2 ortofotos diferentes. Cada vez que se realizaba un cambión en la generación de la escena (por ejemplo, modificación de algún parámetro de la cámara) se realizaban las pruebas sobre esta misma zona para comprobar que seguía renderizándose exactamente la zona deseada sin irregularidades y habiendo aplicado las modificaciones que se hubiesen hecho a la escena.

Posteriormente, se utilizaron (y se siguen utilizando) dos zonas diferentes: una zona urbana (desde las coordenadas actuales (coordenadas precisas para coger teselas completas) [712129.62, 4670707.15] a las [716739.52, 4666097.25]) y otra zona montañosa (desde las coordenadas [773210.77, 4731788.30] a las [775515.72, 4729483.35]). La primera zona, sirvió sobretodo para comprobar que la escena renderizada presentaba la perspectiva deseada. Gracias a los edificios, y principalmente a la plaza de toros, que al ser redonda facilitaba la comprobación del aspecto, se comprobaba que el ratio de aspecto y el ángulo deseado se mostraban correctamente. Posteriormente, esta zona sirvió también como base de pruebas para la inclusión del LiDAR y comprobar como afectaban las esferas a la escena. La zona montañosa, se decidió incluir en las pruebas porque se consideró que una zona alta podía ocasionar algún problema en la renderización de la escena. Y como se puede ver en la Figura 32, hubo problemas.

Figura 32: Zona montañosa con irregularidad

A causa de la altura de la escena, una zona, que queda fuera de nuestro límite de la escena, se interpone entre la cámara y la zona a la que apunta y al no estar pintada con la ortofoto (como quedaba fuera de la escena no se añadía la ortofoto) se ve una zona negra; o en la intersección entre dos MDT se pueden generar estas zonas negras si queda arriba un MDT que no tiene las

ortofotos adyacentes del MDT con el que colisiona. Se realizaron pruebas sobre esta zona hasta que se dio con la solución mínima (sin incluir ortofotos de más) para hacer lo más eficiente posible el renderizado. En la Figura 33 se puede ver un resultado actual.

Figura 33: Zona montañosa sobre la que se probaba

Ambas zonas fueron posteriormente utilizadas para comprobar que el teselado se realizaba correctamente y tras esto siguen utilizándose como zonas de pruebas cada vez que se cambia algo en el renderizado ya sea por deseo o necesidad. Además, comentar que todas estas pruebas no se limitaban a un punto de vista, sino que, aunque se nombra una escena como tal, cada vez que se realizaban (o se realizan) pruebas sobre estas zonas, se comprueban todas las combinaciones de puntos de vista posibles hasta el momento. Otras zonas han sido también objeto de pruebas (principalmente si se ha detectado algún problema), pero han sido pruebas más esporádicas y no se han repetido por cada cambio como si se ha hecho con estas zonas.

6. GESTIÓN DEL PROYECTO

6.1. PLANIFICACIÓN

El proyecto se ha dividido en cuatro fases diferentes, cada una de ellas con unas tareas específicas. A pesar de ello, en algunos casos, ha sido necesario corregir o incluso volver a realizar una tarea de una fase anterior para corregir algún defecto que se ha encontrado en las fases posteriores.

6.1.1. Primera fase

La primera fase corresponde a la fase de inicio del proyecto. En esta primera fase, se realiza el lanzamiento del proyecto seleccionando los datos geográficos de interés y realizando una

primera elección de las herramientas a utilizar. También, se especifica de forma general, las tareas a realizar en el resto de fases y el coste estimado en tiempo de realizarlas.

Además, se realiza un primer acercamiento a los datos de entrada, procesando algunos de ellos y generando un mapa 3D con proyección ortogonal desde diferentes puntos de vista y ángulos de una pequeña zona de forma manual, esto es, sin la ayuda de un programa que automatice todo. Se desarrolla un pequeño programa para procesar estos datos de entrada (los referentes a los MDT) para que nuestra herramienta de renderizado sea capaz de abordar.

6.1.2. Segunda fase

La segunda fase corresponde a la parte más extensa del proyecto. Se desarrolla un programa en el lenguaje de programación Python capaz de automatizar el proceso realizado en la fase anterior y de forma precisa. El programa debe procesar los datos de entrada para adaptarlos a las herramientas a utilizar posteriormente, ser capaz de generar una vista isométrica de una zona especificada por coordenadas introducidas por el usuario. Además, debe ser capaz de trocear (teselar) la salida para adaptar la zona generada a una web. Una vez funcione para una zona pequeña, el programa se ha de extender y ser capaz de realizar lo mismo para una zona más grande.

Se considera esta la fase más extensa y la que más tiempo va a costar llevar a cabo porque contiene la funcionalidad principal que se ha de desarrollar para el proyecto. A causa de esto, al finalizar esta fase y continuar con las siguientes, si se encuentran fallos en el programa que genera los mapas, es necesario volver a tareas referentes de esta fase para corregirlos y que no sigan reproduciéndose posteriormente.

6.1.3. Tercera fase

En esta fase la tarea principal es la de desarrollar una aplicación web capaz de visualizar los resultados obtenidos en la segunda fase. La aplicación no debe ser muy compleja, solo debe representar el mapa (y sus diferentes vistas) y tener la posibilidad de que el usuario pueda navegar por las diferentes vistas. También en esta fase hay que concretar definitivamente el proceso de teselado de la segunda fase para que se adapte correctamente a la aplicación web desarrollada.

6.1.4. Cuarta fase

Durante la última fase del proyecto las tareas principales son las de terminar la documentación y memoria de este y si es necesario realizar un pequeño repaso por todo lo realizado.

6.2. ESFUERZOS

Para llevar un seguimiento de los esfuerzos se ha usado una hoja de cálculo (fichero Excel con formato XLSX) que divide el trabajo en cuatro apartados distintos (análisis y diseño, desarrollo, pruebas y documentación). Cada celda representa el trabajo de uno de estos apartados por día. También existen celdas que muestran la suma del trabajo total por meses, por apartado o sencillamente por día. En la Figura 34 se puede ver un gráfico del reparto de horas de trabajo según el mes.

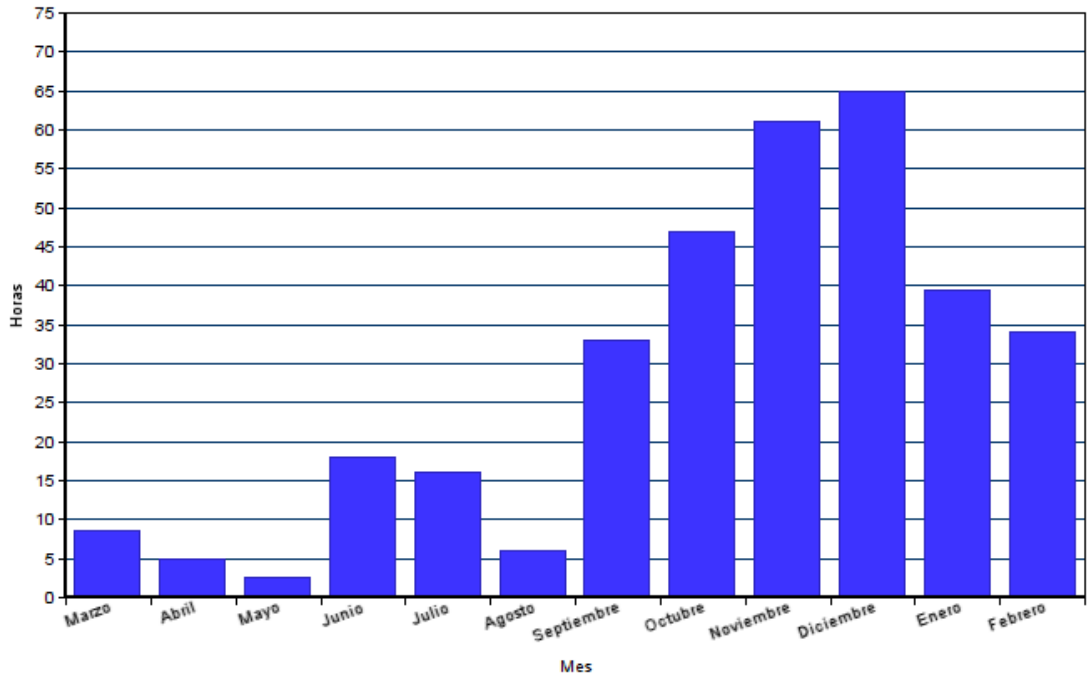

Figura 34: Gráfica con las horas/mes dedicadas al proyecto

Como se puede comprobar, durante los primeros meses la carga de trabajo no fue alta a causa de las exigencias del resto de asignaturas. Principalmente el trabajo se desarrolló durante los meses desde septiembre a diciembre ambos incluidos. Durante enero y febrero la carga de trabajo es menor porque se dedicó a concluir el código del programa y retocar mínimamente este código para mejorarlo. También se dedicó a realizar la documentación. Comentar también, que las horas que la máquina estaba renderizando los mapas no se han contabilizado a pesar de que, durante ese tiempo en el que la máquina se encontraba ocupada trabajando, no era posible realizar algo adicional (documentación, refactorización de código, realización de pruebas...) al mismo tiempo con la misma máquina. Estas horas (y días) de renderizado han copado muchas horas de los meses de enero y febrero, pero tal como se ha visto anteriormente es a causa de la gran cantidad de recursos que se necesita para generar todos los mapas. Las pequeñas pruebas que se realizaban renderizando alguna zona pequeña (que no llevaban más de 10 minutos las más lentas) sí que han sido contabilizadas. Se incluyen en el apartado del desarrollo si se

realizaban durante el desarrollo del código (para ir comprobando como afectaba un cambio en la cámara); o en el apartado de pruebas si eran pruebas más sistemáticas que se realizaban al incluir algún cambio grande o sencillamente para comprobar que todo funcionaba según lo esperado.

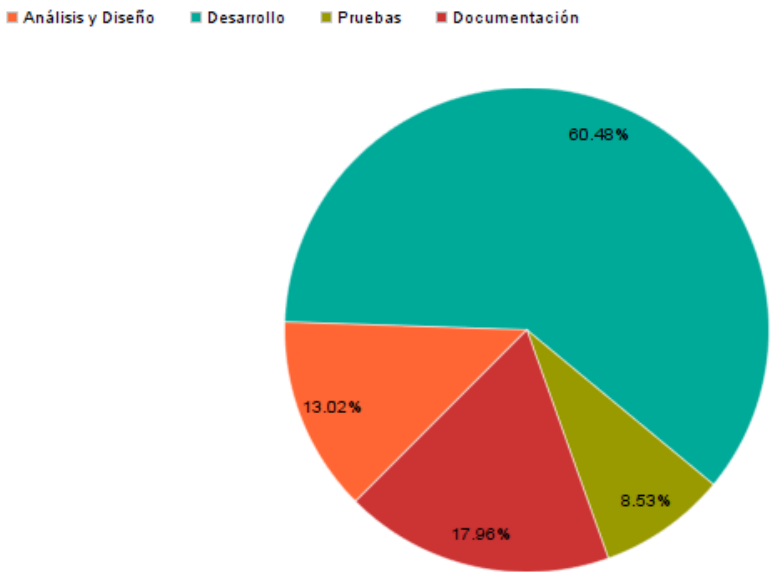

Figura 35: Gráfica con porcentaje de tiempo dedicado a cada apartado del proyecto

Como se puede observar en la Figura 35, el desarrollo a copado el mayor tiempo en el proyecto, aunque se considera que, para el resto de los apartados, también se ha realizado una carga de trabajo correcta.

El tiempo total destinado a la realización del proyecto (sin contar el tiempo de renderizado comentado con anterioridad) ha sido de 334 horas. Este número se considera alto y es a causa de la gran carga de trabajo del apartado del desarrollo. Esta gran carga es debida a la gran cantidad de tiempo que se consumió en desarrollar en forma de ensayo y error (probar un dato de la cámara e ir variando ese dato hasta que quede como queremos) y por las veces que hubo que subsanar algún error o los problemas que han sido comentados en la sección de problemas, los cuales, alguno de ellos llevó bastante tiempo subsanar.

6.3. ANÁLISIS DE RIESGOS

Analizar daño y probabilidad por riesgo		Probabilidad de fallo		
		Alto	Medio	Bajo
Daño en caso de fallo	Alto	A	A	B
	Medio	B	B	C
	Bajo	B	C	C

Riesgo	Prob.	Daño	RC	Justificación	Estrategia
Retraso en la entrega del proyecto	Medio	Medio	B	Si el proyecto no está disponible para la fecha de entrega habría que retrasarlo a una entrega posterior.	Realizar un seguimiento del proyecto a partir de las fases establecidas al inicio del mismo y el código almacenado en GitHub.
No poder volver a una versión anterior	Medio	Bajo	C	Si no se puede volver a una versión anterior del proyecto se podría perder tiempo modificando la actual.	Todo el código y los documentos se encuentran en sistemas bajo control de versiones.
Falta de espacio	Alto	Alto	A	Si hay problemas para almacenar todos los mapas, es posible que algunos puntos de vista o zonas queden sin renderizar.	Buscar un punto intermedio entre renderizar lo máximo posible, pero solo lo necesario.

Figura 36: Tablas análisis de riesgos

6.4. GESTIÓN DE CONFIGURACIONES

6.4.1. Políticas de nombrado

Todos los ficheros referentes a la documentación (memoria, imágenes utilizadas, diagramas...) tienen sus nombres en castellano, pero vienen precedidos de la etiqueta <<TFG_D_>>. Por ejemplo, este fichero (memoria del proyecto) es nombrado como <<TFG_D_memoria.pdf>>.

En cuanto a todos los ficheros referentes a el programa y su código, lo único destacable es que, tienen sus nombres en inglés y se intenta que estos nombres representen el contenido del fichero o la función que tienen en el proyecto.

6.4.2. Control de versiones

El control de versiones se ha realizado utilizando las plataformas GitHub y Dropbox. GitHub se ha usado para llevar el control de versiones del código del programa como se puede comprobar en el repositorio público https://github.com/strummerTFIU/TFG-IsometricMaps, además, también se ha utilizado un repositorio adicional privado en el que se almacenan las teselas de los diferentes mapas generados.

Dropbox, en cambio, ha servido para almacenar la documentación del proyecto.

6.4.3. Copias de seguridad

Como se ha especificado en el anterior apartado, se ha utilizado GitHub[30] y Dropbox[31] como plataformas de control de versiones. Esto permite almacenar copias de seguridad en la nube con las que, además, es posible recuperar una versión anterior de los ficheros en caso de intentar mitigar o sencillamente localizar algún error en alguna versión más reciente del programa o la documentación.

También, se incluyen copias locales en el ordenador personal del autor del proyecto y en un disco duro externo, también propiedad del autor.

7. CONCLUSIONES

Tras la finalización del proyecto, se considera el trabajo realizado como bueno ya que se han conseguido los objetivos propuestos. El objetivo principal era crear las herramientas y procesos necesarios para producir mapas realistas en 3D capaces de ser teselados y se considera que este objetivo ha sido conseguido.

Se es capaz de realizar mapas 3D a partir de unos datos de entrada públicos, pueden ser descargados de repositorios nacionales, ya sea mediante la especificación de una serie de coordenadas, especificando que teselas (si el usuario conoce ya como están nombradas y que tesela corresponde a que zona) o incluso teselando todo lo posible a partir de todos los datos de entrada. Dentro de este objetivo, una idea importante era poder realizar esto para una zona bastante extensa, y se pensó en la Comunidad de Aragón como la zona a renderizar. Se considera que hubiese sido mejor poder renderizar todo Aragón en bastantes puntos de vista diferentes y niveles de zoom, pero, a causa de las limitaciones de hardware, se ha renderizado Aragón, pero de forma más limitada. Son dos los puntos de vista disponibles para visionar todo Aragón, pero existe también una alternativa en la que, para una zona bastante menor, se ha renderizado utilizando todas las posibilidades. Esto ayuda a ver el potencial del proyecto.

Si se consiguiesen máquinas importantes para poder seguir el proyecto, se conseguiría mejorar aún más el resultado y posibilitar mayores puntos de vista para la zona de Aragón o incluso extenderse a toda España, por ello se considera que el proyecto tiene potencial.

A pesar de no ser un proyecto con gran carga en lo que a diseño de software se refiere, se considera que se ha realizado un buen trabajo de manipulación de datos y realización de mapas. Gracias a la naturaleza del proyecto, es bastante sencillo demostrar la funcionalidad del programa, pues se obtienen resultados gráficos, al fin y al cabo, son mapas, y por ello se considera un proyecto agradable para la vista de un usuario o cliente ajeno al proyecto.

En su defecto, el tratar todos estos datos de entrada y producir las salidas, conlleva mucho tiempo en el que la máquina propia a de trabajar, evitando poder usar la máquina para otros propósitos al mismo tiempo, lo que dificulta bastante el trabajo.

La gran experiencia obtenida en el tratamiento de estos datos, en la misma generación de teselas, en la información sobre proyecciones y coordenadas, e incluso en el desarrollo de código en Python (el autor no había trabajado aun con este lenguaje), se considera un gran valor para el futuro profesional ya sea relacionado con el proyecto como no.

8. BIBLIOGRAFÍA

1. Web oficial de Python (https://www.python.org/). Junio 2017.
2. Web oficial de POV-Ray (http://www.povray.org/). Junio 2017.
3. ETRS89 - Wikipedia (https://es.wikipedia.org/wiki/ETRS89). Julio 2017.
4. UTM - Wikipedia (https://es.wikipedia.org/wiki/Sistema_de_coordenadas_universal_transversal_de_Mercator). Julio 2017.
5. EPSG:25830 – Spatial Reference (http://spatialreference.org/ref/epsg/etrs89-utm-zone-30n/). Julio 2017.
6. .ASC files – StackExchange (https://gis.stackexchange.com/questions/71867/understanding-esris-asc-file). Abril 2017.
7. CNIG (https://www.cnig.es/). Abril 2017.
8. PNOA - Wikipedia (https://es.wikipedia.org/wiki/Plan_Nacional_de_Ortofotograf%C3%ADa_A%C3%A9rea). Abril 2017.
9. LiDAR – Wikipedia (https://es.wikipedia.org/wiki/LIDAR). Octubre 2017.
10. Three.js – Wikipedia y web oficial (https://es.wikipedia.org/wiki/Three.js) (https://threejs.org/). Septiembre 2017.
11. GoogleMaps (https://www.google.es/maps). Abril 2017.
12. Estilos de Arquitectura – Dallan (http://dallanmnr.blogspot.com.es/2011/08/estilos-de-arquitectura.html). Septiembre 2017.
13. Web oficial ImageMagick (https://www.imagemagick.org/script/index.php). Noviembre 2017.
14. Librería estándar de Python (https://docs.python.org/3.5/library/index.html). Abril 2017 - Diciembre 2017.
15. Documentación PyPNG (https://pythonhosted.org/pypng/). Junio 2017.
16. PIP – Wikipedia (https://es.wikipedia.org/wiki/Pip_(administrador_de_paquetes)). Junio 2017.
17. Web oficial de NumPy (http://www.numpy.org/). Octubre 2017.
18. Documentación Laspy (https://pythonhosted.org//laspy/). Octubre 2017.
19. PIL (http://www.pythonware.com/products/pil/). Junio 2017.
20. Web oficial de Sublime Text (https://www.sublimetext.com/). Abril 2017.
21. Web oficial GDAL, documentación GDALWarp (http://www.gdal.org/gdalwarp.html). Diciembre 2017.
22. LASzip (https://www.laszip.org/). Octubre 2017.
23. Módulo http.server de Python3 (https://docs.python.org/3/library/http.server.html). Diciembre 2017.
24. Tutorial OpenLayers (https://mappinggis.com/2013/04/como-crear-un-mapa-con-openlayers-3/). Diciembre 2017.
25. Documentación OpenLayers (https://openlayers.org/en/latest/apidoc/). Diciembre 2017.
26. Documentación POV-Ray (http://www.povray.org/documentation/3.7.0/). Junio 2017.
27. Código OSM2World para apoyo cálculo posición cámara – GitHub (https://github.com/tordanik/OSM2World). Julio 2017.

28. Teselado en OpenStreetMaps (https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames). Noviembre 2017.
29. Tutorial de doctest (https://docs.python.org/2/library/doctest.html). Octubre 2017.
30. Web oficial de GitHub (https://github.com/). Abril 2017.
31. Web oficial de Dropbox (https://www.dropbox.com/). Abril 2017.

ANEXOS

A. MANUAL DEL RENDERIZADOR

A continuación, se presentan las acciones necesarias para poder renderizar una zona deseada de Aragón (se podría expandir a España descargando ficheros desde otras fuentes).

a. Descarga de datos de interés

Como se ha comentado en la sección 2.2 Datos de partida, existen tres tipos de datos diferentes con los que se parte y sobre los que se transforman y combinan para obtener el resultado final.

Tanto los MDT como los LiDAR pueden ser descargados desde el centro de descargas del CNIG (http://centrodedescargas.cnig.es) seleccionando "modelos digitales de elevaciones". Para los datos LiDAR solo hay un tipo, pero los MDT hay de tres diferentes tipos. Para el desarrollo de este proyecto se ha seleccionado el más preciso con paso de malla de 5 metros (MDT05), pero cualquiera podría valer, incluso si se desease una vista desde un nivel de zoom bajo sería recomendable utilizar un MDT con menos precisión para aligerar el programa. A partir de aquí, se pueden seleccionar los ficheros que se deseen a partir de un listado organizado por comunidades, código o municipios; o seleccionando directamente en el mapa la zona deseada.

Las ortofotos pueden ser descargadas desde la web de IDEAragón (http://idearagon.aragon.es/descargas) (si se buscasen ortofotos fuera de esta comunidad sería necesario otra fuente de descargas). En el desplegable de colección si ha de buscar el apartado de ortofotos y seleccionar las deseadas del PNOA (existen ortofotos más recientes y otras menos) y como en el listado del CNIG buscar los deseados en los otros dos desplegables. Se han de descargar y descomprimir. Comentar que, si la proyección de las ortofotos no es posible hacerla coincidir con la proyección de los MDT, es necesario reproyectar estas ortofotos utilizando el programa GDALWarp tal cual se explica en la sección 3.5 de Tratamiento de datos.

Se recomienda incluir cada tipo de fichero en directorios diferentes, de forma que la aplicación pueda encontrarlos y diferenciarlos fácilmente.

b. Bibliotecas necesarias

Para que funcione la aplicación es necesaria la inclusión de varias bibliotecas. La descarga e instalación de estas bibliotecas se puede realizar utilizando la herramienta PIP3, esto se hace mediante la instrucción escrita por terminal de: pip3 install <<la biblioteca a instalar>>. Como se comenta en la sección 3.4 de Bibliotecas, las bibliotecas a instalar son: <<NumPy>>, <<laspy>> y <<PIL>>.

También es necesario el programa laszip encargado de descomprimir los ficheros en formato LAZ, que está incluido con el resto de fuentes de la aplicación y el programa POV-Ray que ha de ser instalado.

c. Ejecución aplicación

Una vez descargados los datos de interés y las bibliotecas necesarias, solo queda ejecutar la aplicación. Para ello, es necesario saber las opciones que proporciona el programa a la hora de ejecutarlo.

i. Parámetros obligatorios

- *mdt_directory*: directorio de los ficheros MDT iniciales.
- *png_directory:* directorio de las imágenes generadas de la transformación de los MDT iniciales.
- *orto_directory:* directorio de los ficheros ortofotos.
- *lidar_directory*: directorio de los ficheros LiDAR.
- *dir_view*: dirección de la vista (solo posibles los valores: N, S, E y W). Por ejemplo: el valor N representa que la cámara de la escena estará orientada hacia el norte.
- *angle:* ángulo de la vista (solo posibles los valores 45 y 30).
- *zoom:* nivel de zoom máximo a renderizar (solo valores de 8 - 12).

ii. Parámetros opcionales

- *--max_height <<entero>>:* máxima altura de los heightfield creados, mayores valores no se tendrán en cuenta y serán similares al valor que se le dé. Por defecto este valor es 2200 y no se recomienda su cambio ya que está optimizado para conseguir el mejor aspecto.
- *--renderAll:* renderiza todas las zonas disponibles a partir de los ficheros incluidos en los directorios especificados.
- *--renderTiles:* renderiza las teselas deseadas por el usuario.
- *--transform:* realiza la transformación de los ficheros MDT iniciales en imágenes (activarlo siempre en primera ejecución).
- *--load:* carga los metadatos de los ficheros en sendos ficheros de texto para facilitar la consulta en posteriores ejecuciones (activarlo siempre en primera ejecución).
- *--tile:* permite introducir posteriormente el directorio destino de las teselas generadas.
- *--deletePov*: si se incluye se elimina el fichero POV que se genera con la escena.
- *--lidar:* si se incluye activa el renderizado con LiDAR.

iii. Ejemplo ejecución

Para la primera ejecución del programa una vez descargados los ficheros sería necesario especificar los directorios de los parámetros obligatorios y si se desease la renderización de todas las zonas disponibles a partir de los ficheros descargados, incluir la opción –renderAll. Al tratarse de la primera ejecución, sería necesario incluir las opciones –transform y –load. Habría que incluir el ángulo, dirección de vista y el zoom y si se desease incluir esferas a partir de los ficheros LiDAR, sería necesario la opción --lidar. Así pues, un ejemplo de instrucción sería este:

python3 main_program.py ./MDT/ ./PNG/ ./ORTO/ ./LIDAR/ E 45 10
–renderAll –transform –load –deletePov –lidar

donde se transformarían los MDT a imágenes, se cargaría su información en ficheros de texto, se renderizarían todas las escenas posibles con punto de vista 45-E, incluyendo esferas generadas a partir de los ficheros LiDAR y generando teselas para niveles de zoom de 8-10. Estas teselas serían almacenadas en el directorio por defecto (./result_dir) siguiendo el esquema explicado con anterioridad, y se eliminaría el fichero POV resultante.

Si el usuario quisiese modificar una zona porque no le ha gustado como queda, o sencillamente desea solo esa zona, podría no incluir la opción renderAll, de forma que la aplicación le exigiría las coordenadas necesarias, no incluir la opción --deletePov, para poder modificar el fichero al gusto, --transform y –load, ya que, si no han sido modificados los ficheros de partida ya se encontrarían transformados y los ficheros de metadatos ya creados, haciendo más eficiente la ejecución.

A partir de ahora, el usuario ya posee las teselas con su nombre correctamente generado según su zoom y zona a la que pertenece y podría elegir incluirlas en un cliente sencillo generado por él mismo para visualizarlas en su conjunto, eso sí, atendiendo a el ratio de aspecto de estas.

B. MÁS INFORMACIÓN DISEÑO

a. Proceso transformación inicial

A continuación, se explican varios aspectos del diseño que no se han concretado con anterioridad.

En la Figura 37 se especifican los procesos que se incluyen dentro del proceso principal de transformación de datos inicial. Se incluyen también los repositorios desde donde se descargan los datos iniciales.

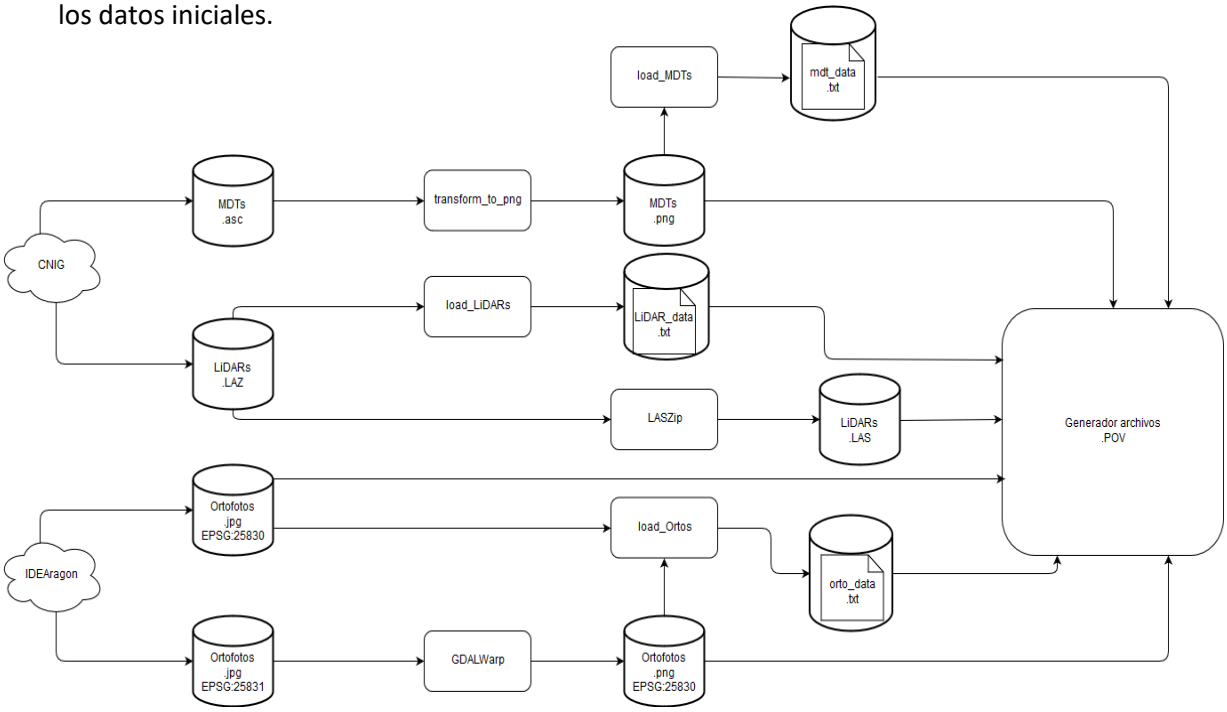

Figura 37: Procesos transformación inicial

Este diagrama muestra los procesos realizados desde que se descargan los datos hasta que son utilizados por el generador de archivos para POV-Ray. Esta "zona" está diseñada de forma que solo sea necesario ejecutarla una vez para que los datos obtenidos permanezcan almacenados e inalterables, y favorecer el rendimiento a la hora de renderizar escenas concretas obteniendo los datos que ya han sido transformados en una ejecución anterior. Para que se ejecuten estos procesos debe activarse la opción correspondiente a la hora de ejecutar el programa, como se ha explicado en el anterior anexo.

Desde el CNIG se descargan (manualmente) los ficheros MDT y LiDAR, que son almacenados en un disco duro externo pero en directorios diferentes, los cuales ambos deben pasar por un filtro para posteriormente ser utilizados para generar la escena. Los MDT, que tienen formato ASC, son transformados a imágenes PNG para que puedan ser incluidos en los objetos heightfield que se crean a la hora de generar el fichero POV. Mediante el proceso load_MDTs, se genera un fichero con los metadatos de cada fichero MDT incluyendo la ruta de la imagen, el tamaño de esta, las coordenadas geográficas en las que se encuentra... Por el contrario, el fichero de metadatos de los LiDARs se crea antes de realizar la transformación inicial de estos ficheros. Esto

es porque la transformación consiste en descomprimir (pasar de formato LAZ a formato LAS) los ficheros LiDAR, y estos ficheros descomprimidos ocupan mucho espacio en disco y como la transformación es bastante rápida y no siempre se utilizan estos ficheros a la hora de renderizar la escena, se ha decidido que se descomprima el fichero solo cuando debe ser utilizado. Para generar el fichero de POV comprueba en el fichero de metadatos cuales necesita y los descomprime entonces. Genera las esferas necesarias y elimina el fichero descomprimido dejando el comprimido intacto.

Desde IDEAragon se descargan las ortofotos del PNOA. La mayoría de estas no pasan por la transformación inicial porque están proyectadas sobre el sistema ETRS89 / UTM 30N que es con el que se trabaja, pero algunas de ellas, las que se encuentran más al este, se encuentran proyectadas en ETRS89 / UTM 31N lo que hace que haya que reproyectarlas para adecuarlas al resto. Este proceso de reproyección es el que se hace durante esta transformación inicial y se realiza mediante el programa GDALWarp y un script para que vaya reproyectando los ficheros. A partir de las ortofotos reproyectadas y de las que no ha sido necesario reproyectar, se genera el fichero de metadatos correspondiente a las ortofotos.

A partir de aquí, el siguiente proceso principal (el generador de archivos POV), recibe la información de la escena del usuario y consulta en los ficheros de metadatos que ficheros son necesarios para renderizar la escena y los obtiene de la ruta donde se encuentren almacenados.

b. Diagrama de secuencia

Como se muestra en el diagrama de secuencia de la Figura 38, una ejecución completa del programa conlleva una serie de procesos que transforman los datos de entrada en nuestra salida deseada, los mapas teselados de la zona especificada. Este diagrama sirve como un apoyo entre el diseño a alto nivel de abstracción de los procesos y los trabajos que deben realizar los módulos desarrollados.

Este diagrama muestra una ejecución donde se pide renderizar todo Aragón (opción –renderAll), las otras dos opciones son similares. No se incluyen los módulos VectorXYZ y Camera porque se considera que ya han sido explicados con antelación y se considera la creación de estos en el diagrama implícita durante la ejecución del proceso de cameraUtils.

El usuario debe incluir la ruta al directorio en el que se encuentran tanto los ficheros MDT (tanto transformados en imagen como no), como las ortofotos y los ficheros LiDAR. Además, debe incluir la dirección de la vista (norte (N), sur (S), este (E) y oeste(W)), el ángulo de la cámara (40 o 35) y el zoom que desea.

Como se ha explicado en esta misma sección, como en la sección X de Tratamiento de datos, lo primero que se hace (si está la opción activada) es el proceso de transformar los MDTs utilizando el módulo heightfield (se considera que las ortofotos en la proyección errónea ya han sido reproyectadas con anterioridad).

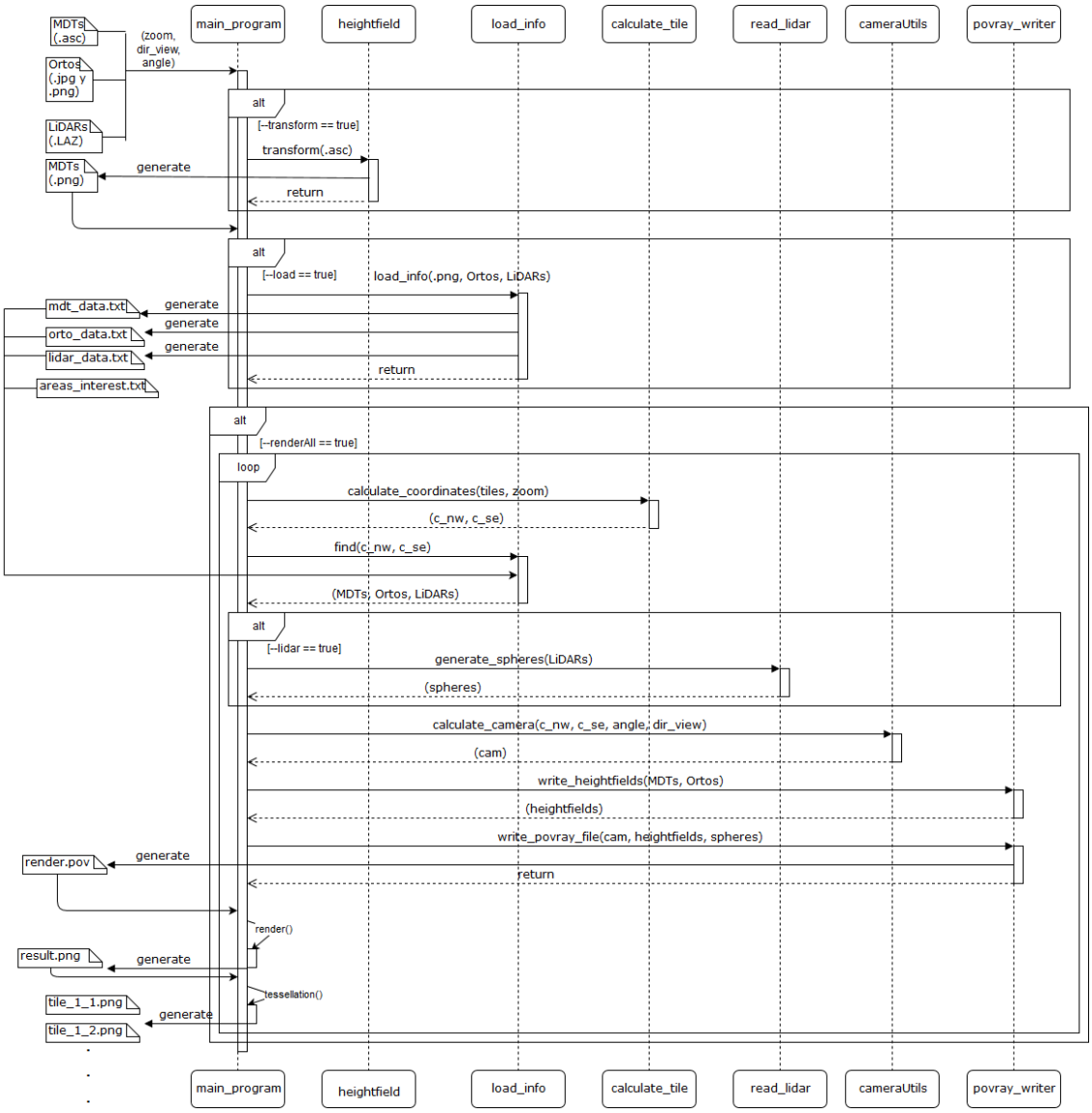

Figura 38: Diagrama secuencia procesos

Se realiza la carga de información (creación de ficheros de metadatos). Se incluye para cada fichero información sobre tamaño de el mismo, coordenadas geográficas donde se encuentran los datos representados en el fichero; entre algún dato más, además de su nombre y localización. Estos ficheros de texto permiten que en cada ejecución no sea necesario ir fichero por fichero obteniendo su información y comprobando si es necesario incluirlo en la zona de renderizado para la ejecución actual, haciendo que mejore el rendimiento. Tanto este proceso como el anterior son opcionales (es necesario incluir un parámetro para cada uno para que se ejecuten), esto es así porque si se trabaja con datos de entrada iguales para ejecuciones diferentes, tanto los MDTs transformados como la información cargada en los ficheros de texto va a ser la misma, y no es necesario volver a ejecutar estos procesos. Esto aligera algo el tiempo por el que el programa es ejecutado para ejecuciones del programa seguidas.

A continuación, el programa principal, a partir de la información que se encuentra en los ficheros de datos, generados en el paso de cargar la información, obtiene los MDTs, ortofotos y LiDAR

que necesita (aplicando un pequeño offset para evitar zonas negras en zonas altas o en zonas con cambios de MDT). A partir de las coordenadas de la zona a renderizar, la dirección y ángulo de la vista, se genera una cámara que vaya a mostrar la zona en perspectiva isométrica. Gracias a los MDTs y ortofotos necesarias, se generan los objetos heightfields; y gracias a los ficheros LiDAR, si procede (debe ser incluido por el usuario), se generan objetos esferas, obtenidos del proceso de lectura del LiDAR (read_lidar), que dotan a la escena de ese aspecto 3D. Todos estos objetos, la cámara y una fuente de luz, se incluyen en un fichero con formato POV el cual es la salida del último filtro (povray_writer) antes de renderizar la escena. Con este fichero POV el programa POV-Ray renderiza la escena resultante de incluir todos estos aspectos. Esta escena es renderizada respetando el ratio de aspecto (imagen más ancha o menos) de la zona a renderizar y con una resolución según las teselas que abarca.

Por último, una vez renderizada la escena, se procede al último aspecto del sistema, el teselado. Se trocea la escena según las teselas que abarca, el tamaño de estas (respetando el ratio de aspecto) y el nivel zoom. Para niveles de zoom altos (de 9 a 12) se incluye también un teselado de la misma escena para niveles menores (hasta 8), siempre y cuando sea posible (la escena abarque al menos una tesela completa en el zoom anterior).

Aunque tanto la instrucción de renderizado como la de teselado se encuentran en el programa principal (main_program), ambos necesitan acceder a la salida de dos procesos. La primera salida, como ya se ha comentado, el fichero POV con el que renderizar proveniente de povray_writer; y la segunda, las teselas necesarias para renderizar la escena (calculate_tile).

Al final de la ejecución del programa, se muestra el tiempo total de ejecución en minutos y se elimina (o no, según si se ha especificado la opción) el fichero con formato POV de la última escena renderizada, por si el usuario desea modificar algo de la escena de forma manual (ha encontrado un fallo o decide que no se deben mostrar todos los objetos heightfield) y volverla a renderizar.

Las opciones a incluir para realizar una ejecución u otra están especificadas en el manual del renderizador del Anexo A.

C. DOCUMENTACIÓN MÓDULOS

En el siguiente anexo se describen los módulos creados, para la realización del proyecto, y sus interfaces. No se incluye el módulo PyPNG ya que ha sido descargado y no ha sido ni creado ni modificado para la realización del proyecto.

a. calculate_tile

Módulo encargado de realizar las operaciones referentes al cálculo de las teselas necesarias y a la transformación del código de estas según el punto de vista.

i. Variables

- **origin:** Coordenadas de la esquina superior izquierda de toda la zona teselable (zona cuadrada mayor que Aragón). Su valor actual es de [399809, 4881610].
- **end:** Coordenadas de la esquina inferior derecha de toda la zona teselable (zona cuadrada mayor que Aragón). Su valor actual es de [989876, 4291543].

ii. Operaciones

- **calculate_tile(x, y, z):**

Calcula el código (nombre) de la tesela a partir de las coordenadas y el nivel de zoom pasados como parámetro.

Parámetros:

- **x:** Coordenada x del punto del que se desea el código de tesela.
- **y:** Coordenada x del punto del que se desea el código de tesela.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores:** Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida.
- **'null':** Si la ejecución ha obtenido un fallo (coordenadas superan los límites que forman el rectángulo a partir de las variables origin y end) se devuelve este valor.

- **calculate_coordinates(xtile, ytile, z):**

Calcula las coordenadas del vértice superior izquierdo de la tesela y nivel de zoom pasados como parámetro.

Parámetros:

- **xtile:** Código de la tesela para el valor X.
- **ytile*:*** Código de la tesela para el valor Y.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores:** Si la ejecución ha sido correcta se devuelven las coordenadas X e Y de la esquina superior izquierda de la tesela.
- **'null':** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_to_south(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista sur a partir de la tesela para punto de vista norte y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile:** Tupla con los códigos de la tesela para punto de vista norte a transformar.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores:** Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null':** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_to_east(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista este a partir de la tesela para punto de vista norte y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile:** Tupla con los códigos de la tesela para punto de vista norte a transformar.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores*:*** Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null'*:*** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_to_west(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista oeste a partir de la tesela para punto de vista norte y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile:** Tupla con los códigos de la tesela para punto de vista norte a transformar.
- z: Nivel de zoom.

Devuelve:

- **Tupla 2 valores***:* Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null'***:* Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_from_south(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista norte a partir de la tesela para punto de vista sur y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile***:* Tupla con los códigos de la tesela para punto de vista sur a transformar.
- **z***:* Nivel de zoom.

Devuelve:

- **Tupla 2 valores***:* Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null':** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_from_east(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista norte a partir de la tesela para punto de vista este y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile:** Tupla con los códigos de la tesela para punto de vista este a transformar.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores:** Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null':** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

- **tile_from_west(tile, z):**

Calcula el nuevo código (nombre) de la tesela para punto de vista norte a partir de la tesela para punto de vista oeste y el nivel de zoom pasados como parámetro.

Parámetros:

- **tile:** Tupla con los códigos de la tesela para punto de vista oeste a transformar.
- **z:** Nivel de zoom.

Devuelve:

- **Tupla 2 valores:** Si la ejecución ha sido correcta se devuelven los valores X e Y de la tesela obtenida tras la transformación.
- **'null':** Si la ejecución ha obtenido un fallo (código de la tesela supera el límite según el nivel de zoom) se devuelve este valor.

b. camera

Módulo que incluye la clase Camera encargada de generar objetos Camera (cámara) y de realizar las operaciones simples sobre estos.

i. Atributos

- **pos:** VectorXYZ que representa la posición de la cámara.
- **up:** VectorXYZ que representa el vector up de la cámara.
- **right:** VectorXYZ que representa el vector right de la cámara.
- **lookAt:** VectorXYZ que representa el vector lookAt de la cámara.
- **aspectRatio:** valor real que representa el ratio de aspecto de la escena.

Para mayor precisión de estos vectores consultar la sección 4.1 (Aspectos técnicos).

ii. Operaciones

- **get_pos():**

Devuelve el VectorXYZ que representa la posición de la cámara.

Parámetros:

- **Ninguno**

Devuelve:

- **Objeto VectorXYZ:** Objeto con los valores de la posición de la cámara.

- **get_up():**

Devuelve el VectorXYZ que representa el vector up de la cámara.

Parámetros:

- **Ninguno**

Devuelve:

- **Objeto VectorXYZ***:* Objeto con los valores del vector up de la cámara.

- **get_right():**

Devuelve el VectorXYZ que representa el vector right de la cámara.

Parámetros:

- **Ninguno**

Devuelve:

- **Objeto VectorXYZ***:* Objeto con los valores del vector right de la cámara.

- **get_lookAt():**

Devuelve el VectorXYZ que representa el vector lookAt de la cámara.

Parámetros:

- ***Ninguno***

Devuelve:

- **Objeto VectorXYZ***:* Objeto con los valores del vector lookAt de la cámara.

- **get_aspectRatio():**

Devuelve el valor del ratio de aspecto de la cámara.

Parámetros:

- ***Ninguno***

Devuelve:

- **Valor real:** Número real que representa el ratio de aspecto de la cámara.

- **set_aspectRatio(new):**

Cambia el valor del ratio de aspecto de la cámara por el pasado por parámetro.

Parámetros:

- **new***:* Nuevo valor (real) del ratio de aspecto.

Devuelve:

- ***Nada***

- **setCamera(posX, posY, posZ, upX, upY, upZ, lookAtX, lookAtY, lookAtZ, rightX, aspectRatio):**

Cambia los atributos de la cámara a partir de los valores pasados como parámetro.

Parámetros:

- **posX***:* Nuevo valor X (real) de la posición de la cámara.
- **posY***:* Nuevo valor Y (real) de la posición de la cámara.
- **posZ:** Nuevo valor Z (real) de la posición de la cámara.
- **upX:** Nuevo valor X (real) del vector up de la cámara.
- **upY***:* Nuevo valor Y (real) del vector up de la cámara.
- **upZ:** Nuevo valor Z (real) del vector up de la cámara.
- **lookAtX***:* Nuevo valor X (real) del vector lookAt de la cámara.
- **lookAtY:** Nuevo valor Y (real) del vector lookAt de la cámara.
- **lookAtZ***:* Nuevo valor Z (real) del vector lookAt de la cámara.
- **rightX:** Nuevo valor X (real) del vector right de la cámara.
- **aspectRatio:** Nuevo valor X (real) del ratio de aspecto.

Devuelve:

- ***Nada***

c. cameraUtils

i. Variables

- **Ninguna**

ii. Operaciones

- **calculate_camera(xz1, xz2, angleDeg, direction):**

Crea un objeto Camera y calcula sus valores a partir de los valores pasados como parámetro.

Parámetros:

- **xz1:** Tupla con las coordenadas del vértice superior izquierdo.
- **xz2:** Tupla con las coordenadas del vértice inferior derecho.
- **angleDeg:** Ángulo de perspectiva deseado para la escena (45 o 30).
- **direction:** Dirección del punto de vista de la escena (N, S, E o W).

Devuelve:

- **Objeto Camera:** Cámara con los valores calculados a partir de los datos de la escena pasados como parámetro.

d. heightfield

i. Variables

- **Ninguna**

ii. Operaciones

- **transform_file_to_heightfield(file_in, file_out, maxHeight):**

Crea un fichero de texto y una imagen en formato PNG a partir de un fichero MDT con formato ASC pasado como parámetro.

Parámetros:

- ***file_in:*** Ruta del fichero ASC a transformar.
- ***file_out:*** Ruta del fichero PNG a crear.
- ***maxHeigth:*** Altura máxima que se va a representar en la imagen resultante. Valores superiores a este serán, en la nueva imagen, iguales a este parámetro.

Devuelve:

- ***Nada***

e. load_info

Módulo encargado de realizar las operaciones referentes a la creación y consulta de ficheros de texto que contienen todos los ficheros de entrada y datos metadatos sobre ellos.

i. Variables

- **m_file:** Ruta del fichero encargado de almacenar los datos de los ficheros MDT.
- **o_file:** Ruta del fichero encargado de almacenar los datos de los ficheros ortofotos.
- **l_file:** Ruta del fichero encargado de almacenar los datos de los ficheros LiDAR.
- **a_file:** Ruta del fichero encargado de almacenar las áreas de interés para el LiDAR (zonas donde se necesita LiDAR).
- **laszip:** Ruta del programa LASZip encargado de descomprimir los archivos LAZ para transformarlos en archivos LAS.

ii. Operaciones

- **load_mdt_info(png_directory):**

Crea el fichero de texto que incluye todos los MTDs disponibles en el sistema y algunos datos sobre ellos (tamaño, ruta...).

Parámetros:

- **png_directory:** Directorio donde se encuentran los ficheros MDT.

Devuelve:

- **Nada**

- **find_mdt(x1, y1, x2, y2):**

Busca en el fichero de texto de los MDT y devuelve aquellos en los que algunos de sus puntos se encuentran dentro del rectángulo formado por las coordenadas pasadas como parámetro.

Parámetros:

- **x1:** Coordenada del punto más al oeste.
- **y1:** Coordenada del punto más al norte.
- **x2:** Coordenada del punto más al este.
- **y2:** Coordenada del punto más al sur.

Devuelve:

- **Lista de listas:** Lista con información de los MDTs que se incluyen dentro de las coordenadas especificadas como parámetros. Si la lista está vacía es que no se ha encontrado ningún MDT dentro de los límites establecidos.

- **load_orto_info(orto_directory):**

Crea el fichero de texto que incluye todas las ortofotos disponibles en el sistema y algunos datos sobre ellos (tamaño, ruta...).

Parámetros:

- **orto_directory:** Directorio donde se encuentran las ortofotos.

Devuelve:

- **Nada**

- **find_orto(x1, y1, x2, y2, mdts):**

Busca en el fichero de texto de los MDT y devuelve aquellos en los que algunos de sus puntos se encuentran dentro del rectángulo formado por las coordenadas pasadas como parámetro.

Parámetros:

- **x1:** Coordenada del punto más al oeste.
- **y1:** Coordenada del punto más al norte.
- **x2:** Coordenada del punto más al este.
- **y2:** Coordenada del punto más al sur.
- **mdts:** Lista de ficheros MDT.

Devuelve:

- **Lista de listas:** Lista con información de las ortofotos que se incluyen dentro de las coordenadas especificadas como parámetros. Si la lista está vacía es que no se ha encontrado ninguna ortofoto dentro de los límites establecidos.

- **load_lidar_info(lidar_directory):**

Crea el fichero de texto que incluye todos los ficheros LiDAR disponibles en el sistema y algunos datos sobre ellos (tamaño, ruta...).

Parámetros:

- **lidar_directory:** Directorio donde se encuentran los ficheros LiDAR.

Devuelve:

- **Nada**

- **find_lidar(areas, c1, c2):**

Busca en el fichero de texto de los LiDAR y devuelve aquellos en los que algunos de sus puntos se encuentran dentro del rectángulo formado por las coordenadas pasadas como parámetro y dentro de las áreas de interés.

Parámetros:

- **areas:** Lista con las áreas de interés incluidas en las escenas.
- **c1:** Tupla con las coordenadas de la esquina superior izquierda.
- **c2:** Tupla con las coordenadas de la esquina inferior derecha.

Devuelve:

- **Lista de listas:** Lista con información de los ficheros LiDAR que se incluyen dentro de las coordenadas especificadas como parámetros. Si la lista está vacía es que no se ha encontrado ningún fichero LiDAR dentro de los límites establecidos.

- **find_a_interest(x1, y1, x2, y2):**

Busca en el fichero de texto de las áreas de interés y devuelve aquellas en las que algunos de sus puntos se encuentran dentro del rectángulo formado por las coordenadas pasadas como parámetro.

Parámetros:

- **x1:** Coordenada del punto más al oeste.
- **y1:** Coordenada del punto más al norte.
- **x2:** Coordenada del punto más al este.
- **y2:** Coordenada del punto más al sur.

Devuelve:

- **Lista de listas:** Lista con las áreas de interés que se incluyen dentro de las coordenadas especificadas como parámetros. Si la lista está vacía es que no se ha encontrado ninguna área de interés dentro de los límites establecidos.

- **is_collision(x1, y1, x2, y2, mx1, my1, mx2, my2):**

Comprueba si existe colisión, es decir, tienen alguna coordenada común en su interior, entre los dos rectángulos formados por las coordenadas pasadas como parámetros. El primer rectángulo está formado por los parámetros x1, y1, x2, y2 y el segundo por los parámetros mx1, my1, mx2, my2.

Parámetros:

- **x1:** Coordenada del punto más al oeste (rectángulo 1).
- **y1:** Coordenada del punto más al norte (rectángulo 1).
- **x2:** Coordenada del punto más al este (rectángulo 1).
- **y2:** Coordenada del punto más al sur (rectángulo 1).
- **mx1:** Coordenada del punto más al oeste (rectángulo 2).
- **my1:** Coordenada del punto más al norte (rectángulo 2).
- **mx2:** Coordenada del punto más al este (rectángulo 2).
- **my2:** Coordenada del punto más al sur (rectángulo 2).

Devuelve:

- **True:** Existe colisión entre los dos rectángulos que forman las coordenadas pasadas como parámetro.
- **False:** No existe colisión entre los dos rectángulos que forman las coordenadas pasadas como parámetro.

- **load_info(png_directory, orto_directory, lidar_directory):**

Crea los ficheros de textos que incluyen la información de los ficheros MDT y LiDAR y las ortofotos disponibles en el sistema.

Parámetros:

- **png_directory:** Directorio donde se encuentran los ficheros MDT.
- **orto_directory:** Directorio donde se encuentran las ortofotos.
- **lidar_directory:** Directorio donde se encuentran los ficheros LiDAR.

Devuelve:

- **Nada**

f. main_program

Módulo principal encargado de realizar la interacción con el usuario y de invocar al resto de funciones del sistema para desarrollar las acciones requeridas. Ofrece el cálculo del tiempo de ejecución.

i. Variables

- **Ninguna**

ii. Operaciones

- **tiles_to_render(c1, c2, zoom):**

Calcula que teselas y sus coordenadas son límite para la escena (tesela superior izquierda e inferior derecha) limitada por los valores de las coordenadas y el nivel de zoom pasados como parámetro.

Parámetros:

- **c1:** Tupla con las coordenadas de la esquina superior izquierda.
- **c2:** Tupla con las coordenadas de la esquina inferior derecha.
- **zoom:** Nivel de zoom.

Devuelve:

- **Tupla de 4 tuplas:** Dos tuplas que contienen el código de las teselas límite y 2 tuplas que contienen las coordenadas de estas teselas (vértice más al norte y oeste y vértice más al sur y este).
- **'null':** Si se produce algún error en el cálculo de teselas (algún valor superior a el posible a partir del nivel de zoom).

- **dir_view_tile(tile, dir_view, zoom):**

Transforma, a partir del punto de vista norte, el código de la tesela al punto de vista especificado como parámetro.

Parámetros:

- **tile:** Tupla con el código de la tesela a transformar.

- **dir_view:** Carácter que especifica el punto de vista de la tesela nuevo (N, S, E o W).
- **zoom:** Nivel de zoom.

Devuelve:

- **Tupla de 2 valores:** Contiene el nuevo código de la tesela para el punto de vista especificado por parámetro.

- **render(tile1, tile2, c1, c2, dir_view, angle, result, lidar):**

Genera el archivo de POV-Ray que representa la escena pasada a través de los valores de los parámetros. Posteriormente utiliza el programa POV-Ray para renderizar la escena a partir del archivo generado.

Parámetros:

- **tile1:** Tupla con el código de la tesela límite al noroeste.
- **tile2:** Tupla con el código de la tesela límite al sureste.
- **c1:** Tupla con las coordenadas de la esquina superior izquierda.
- **c2:** Tupla con las coordenadas de la esquina inferior derecha.
- **dir_view:** Carácter que especifica el punto de vista de la tesela nuevo (N, S, E o W).
- **angle:** Ángulo de perspectiva deseado para la escena (45 o 30).
- **result:** Ruta donde guardar la escena renderizada.
- **lidar:** Booleano para activar la generación de esferas a partir de ficheros LiDAR.

Devuelve:

- **Tupla de 3 valores:** Dos valores para el tamaño de las teselas y otro con el número de teselas necesarias para la escena.
- **'null':** Si se produce algún error en el renderizado (la zona no contiene ningún heightfield).

- **tessellation(result, tile1, tile_size_x, tile_size_y, w_tiles, zoom, dir_view, angle, dist_tile):**

Genera las teselas a partir de la escena renderizada. La ruta de la escena renderizada y la información del teselado son pasados por parámetros.

Parámetros:

- **result:** Ruta donde se encuentra la imagen de la escena renderizada.
- **tile1:** Tupla con el código de la tesela inicial.
- **tile_size_x:** Tamaño de la tesela para el eje X.
- **tile_size_y:** Tamaño de la tesela para el eje Y.
- **w_tiles:** Número de teselas a generar (para una dimensión).

- **zoom:** Nivel de zoom.
- **dir_view:** Carácter que especifica el punto de vista de la tesela nuevo (N, S, E o W).
- **angle:** Ángulo de perspectiva deseado para la escena (45 o 30).
- **dist_tile:** Ruta donde guardar las teselas generadas.

Devuelve:

- **Nada**

- **main():**

Función principal del sistema. Realiza la interacción con el usuario y calcula el tiempo de ejecución del programa.

Parámetros:

- **Ninguno**

Devuelve:

- **Nada**

g. povray_writer

Módulo encargado de generar el archivo de POV-Ray que representa la escena a renderizar.

i. Variables

- **Ninguna**

ii. Operaciones

- **write_heigthfields(mdt_list, orto_list):**

Genera los heightfields necesarios para la escena a partir de los ficheros MDT y ortofotos necesarias pasados como parámetros.

Parámetros:

- **mdt_list:** Lista con los MDTs a incluir en la escena.
- **orto_list:** Lista con las ortofotos a incluir en la escena.

Devuelve:

- **Cadena de texto:** Cadena de texto que contiene todos los heightfields a incluir en formato de fichero para POV-Ray.

- **write_povray_file(cam, heightfields, spheres):**

Crea el fichero POV e incluye los objetos pasados como parámetros.

Parámetros:

- **cam:** Cadena de texto en formato para POV-Ray que incluye la definición de la cámara de la escena.
- **heightfields:** Cadena de texto en formato para POV-Ray que incluye los heightfields incluidos en la escena.
- **spheres:** Cadena de texto en formato para POV-Ray que incluye los objetos esfera que representan puntos pertenecientes a ficheros LiDAR.

Devuelve:

- **Nada**.

- **write_headers_and_camera(pov, cam):**

Escribe en el fichero POV la definición de la cámara.

Parámetros:

- **pov:** Objeto que representan el stream de datos al fichero de POV-Ray.
- **cam:** Cadena de texto en formato para POV-Ray que incluye la definición de la cámara de la escena.

Devuelve:

- **Nada**.

- **write_texture_finish():**

Escribe en el fichero POV las características finales del objeto heightfield (difuminación, color ambiente...).

Parámetros:

- **Ninguno**

Devuelve:

- **Nada**.

h. read_lidar

Módulo encargado de descomprimir los ficheros LiDAR y generar las esferas que representan los puntos contenidos en estos ficheros.

i. Variables

- **laszip:** Ruta del programa LASZip encargado de descomprimir los archivos LAZ para transformarlos en archivos LAS.

ii. Operaciones

- **generate_spheres(lidar_list, areas_list, c1, c2):**

Genera una cadena de texto que contiene objetos esfera en formato para POV-Ray que representa parte de los puntos de los ficheros LiDAR pasados como parámetro. Estos puntos son escogidos según si se encuentran dentro de los límites establecidos por el resto de los parámetros.

Parámetros:

- **lidar_list:** Lista con los ficheros LiDAR a incluir.
- **areas_list:** Lista con las áreas de interés que se incluyen en la escena.
- **c1:** Tupla con las coordenadas de la esquina superior izquierda.
- **c2:** Tupla con las coordenadas de la esquina inferior derecha.

Devuelve:

- **Cadena de texto:** Contiene las esferas generadas a partir de los puntos que se encuentran dentro de los límites establecidos y con la información que poseen en los archivos LiDAR. Si esta cadena de texto está vacía es que no se han encontrado esferas que cumplan lo establecido.

i. vector_XYZ

Módulo que incluye la clase VectorXYZ encargada de generar objetos Vector y de realizar las operaciones simples sobre estos.

i. Atributos

- **x:** Valor para la dimensión de las X.
- **y:** Valor para la dimensión de las Y.
- **z:** Valor para la dimensión de las Z.

ii. Operaciones

- **get_x():**

Devuelve el valor que representa la dimensión X del vector.

Parámetros:

- **Ninguno**

Devuelve:

- **Valor real:** Número real que representa la dimensión X.

- **get_y():**

Devuelve el valor que representa la dimensión Y del vector.

Parámetros:

- **Ninguno**

Devuelve:

- **Valor real:** Número real que representa la dimensión Y.

- **get_z():**

Devuelve el valor que representa la dimensión Z del vector.

Parámetros:

- **Ninguno**

Devuelve:

- **Valor real:** Número real que representa la dimensión Z.

- **length():**

Devuelve la longitud del vector.

Parámetros:

- **Ninguno**

Devuelve:

- **Valor real:** Número real que representa la longitud del vector.

- **length_squared():**

Devuelve la longitud del vector al cuadrado.

Parámetros:

- **Ninguno**

Devuelve:

- **Valor real:** Número real que representa la longitud del vector al cuadrado.

- **normalize():**

Devuelve el vector normalizado.

Parámetros:

- **Ninguno**

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el vector normalizado.

- **add(other):**

Devuelve la suma de dos vectores.

Parámetros:

- **other:** Objeto VectorXYZ que se suma al objeto VectorXYZ propio.

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el vector sumado al pasado como parámetro.

- **subtract(other):**

Devuelve la resta de dos vectores.

Parámetros:

- **other:** Objeto VectorXYZ que se resta al objeto VectorXYZ propio.

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el vector pasado como parámetro restado al propio.

- **cross(other):**

Devuelve el producto vectorial de dos vectores.

Parámetros:

- **other:** Objeto VectorXYZ que se multiplica vectorialmente al objeto VectorXYZ propio.

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el vector producto vectorial entre el propio y el pasado por parámetro.

- **mult(scalar):**

Devuelve el resultado de multiplicar el vector por el escalar pasado como parámetro.

Parámetros:

- **scalar:** Escalar que se multiplica al objeto VectorXYZ propio.

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el producto escalar entre el vector propio y el escalar pasado por parámetro.

- **invert():**

Devuelve el vector invertido.

Parámetros:

- **Ninguno.**

Devuelve:

- **Objeto VectorXYZ:** Objeto VectorXYZ que representa el vector invertido.

- **toString():**

Devuelve el objeto Vector en formato texto.

Parámetros:

- **Ninguno**.

Devuelve:

- **Cadena de texto:** Representa los valores del objeto VectorXYZ separados por comas y limitados por los símbolos “<” y “>”.