UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Doctorado

Programa de doctorado: Programa de Doctorado en Tecnologías de la Telecomunicación e Ingeniería Computacional.
Unidad responsable del programa de doctorado: Instituto Universitario de Microelectrónica Aplicada.

Nombre de la tesis

Novel competitive methods for the processing of spatial data and applications to vector streaming of Big Geo Data

Tesis presentada por D. Sebastián Eleazar Ortega Trujillo

Tesis codirigida por Dr. José Pablo Suárez Rivero y Dr. Agustín Rafael Trujillo Pino

El codirector El codirector El doctorando

Las Palmas de Gran Canaria, a 21 de septiembre de 2020

Resumen en español

Presentación y contexto

Vivimos en un mundo cada día más digital. Allá donde miremos, podemos observar todo tipo de dispositivos electrónicos capaces de ofrecerte información y de capturar datos acerca de tu entorno: desde los clásicos televisores y ordenadores hasta relojes de pulsera digitales pasando por teléfonos móviles o tablets. Todos ellos equipados con pantalla, funciones de red y diferentes sensores que recogen, entre otros, tu posición. Adicionalmente, existen grandes cantidades de sensores repartidos por cualquier lugar del planeta capaces de leer y transmitir continuamente todo tipo de propiedades, como temperatura, humedad, presión, radiación o desplazamiento, entre otros. De nuevo, todos ellos con una posición asociada. Todo esto genera enormes y siempre crecientes volúmenes de datos georreferenciados que gestionar.

Entenderemos por *dato georreferenciado* o *espacial* aquel que lleva asociado una posición concreta en el globo terráqueo. Para esta asociación es generalmente utilizado un *sistema de referencia espacial* (SRS, en sus siglas en inglés) que fija esa posición en base a una tupla de coordenadas. Estas coordenadas pueden ser geográficas (latitud, longitud y altura) o bien cartográficas proyectadas (x,y,z).

Los datos georreferenciados pueden dividirse en dos grandes categorías: datos vectoriales y datos ráster. Los datos vectoriales son variables geométricas, puntos, líneas y polígonos, a las cuales se les asocia una posición en el globo. Los datos ráster, en cambio, relacionan esa posición del globo a píxeles de una imagen o celdas de una rejilla regular. Entre los datos ráster podemos encontrar fotografías aéreas y satelitales, mapas cartográficos o modelos digitales de elevación.

Este trabajo se centra principalmente en los datos vectoriales. Éstos son mayormente generados utilizando tres tipos de tecnología: los *sistemas de posicionamiento global* (GPS, en sus siglas en inglés), la *teledetección* o *remote sensing* y los *sistemas de información geográfica* (GIS, en sus siglas inglesas). Los GPS son sistemas que, mediante triangulación con satélites de posición siempre conocida, determinan al instante y con alta precisión la ubicación terrestre de un dispositivo dado. Esta ubicación puede ser usada en solitario en forma de punto o bien combinarla con otra variable dada. En *remote sensing*, los datos

espaciales se obtienen utilizando sensores que analizan su entorno sin establecer contacto con él. Estos sensores pueden ser activos, enviando una señal contra el entorno, o pasivos, limitándose a capturar información. Las técnicas más conocidas de teledetección para datos vectoriales son de tipo activo: el RADAR y GPR para medición de distancias en entornos aéreos y subterráneos utilizando ondas de radio, el LIDAR para medición de distancias utilizando haces de luz, y el SONAR para medición de distancias generalmente en medios subacuáticos utilizando ondas de sonido. Las respuestas en estos casos suelen venir en forma de nubes de puntos de volumen y densidad muy variable. Finalmente, los GIS son sistemas de información que combinan elementos hardware y software para que un usuario pueda crear, recolectar, editar, organizar, convertir, modelar, almacenar y consultar datos espaciales. Estos sistemas van desde plataformas completas tanto abiertas como privativas (ESRI ArcGis, QGis, Grass, Autodesk, Capaware) hasta bibliotecas de funciones (proj4, GDAL, GeoTools, LasTools) pasando por servidores de datos georeferenciados (Geoserver, MapServer).

Al manejar grandes conjuntos de datos vectoriales, surgen problemas para su almacenamiento, su transmisión a través de la red y su visualización. Esto es especialmente cierto cuando se pretende, desde un dispositivo portable, recolectar los datos o bien visualizar los mismos. Los dispositivos móviles tienen memoria y capacidad de cómputo limitada, lo que hace imposible intentar almacenar en ellos un set entero de datos de cualquier índole. Además, tienen un tamaño de pantalla limitado que hace que se deba cuidar cuánta información se muestra en cada momento para no saturar al usuario con datos que se superponen el uno al otro. Finalmente, existe también una limitación en forma de datos móviles, que habitualmente tienen un coste para el usuario. Esto hace que interese limitar al máximo la cantidad de datos a transferir y hacerlo únicamente bajo demanda del usuario.

En esta tesis doctoral se plantean nuevos métodos para procesar grandes conjuntos de datos espaciales, con vistas por un lado a formar estructuras que faciliten la transmisión selectiva y progresiva de los mismos y, por otro lado, a mejorar el entendimiento que de los mismos tenga el usuario.

Objetivos y organización de la tesis doctoral

En esta disertación se plantea el estudio de nuevos métodos para el procesado de grandes conjuntos de datos georreferenciados o Big Geo Data. En estos métodos se considerarán la entrada de datos, su acceso y almacenamiento eficiente, su transmisión por red y su adecuada visualización. Además, deberán garantizar la coherencia en la representación y muestra de los resultados, no debiendo éstos mostrar singularidades o degeneraciones con respecto a los datos de entrada.

Para un almacenamiento, transmisión y visualización eficientes, se deben introducir estrategias de nivel de detalle que permitan representar diferentes versiones más o menos detalladas de un mismo volumen de datos, de forma que sean intercambiables en función de la perspectiva o de la importancia desde el punto de vista del usuario de cada tipo de datos. Además, se requerirá de

una técnica de *streaming* que permita descargar esos datos bajo demanda del usuario. Citando como ejemplo un aplicación de globo virtual con un conjunto de datos georeferenciados para consultar, se deberá añadir detalle y por tanto descargar más datos de aquellas zonas en las que el usuario decida navegar, requiriendo técnicas de decisión inteligentes.

Sabiendo esto, se plantean como objetivos principales de esta tesis los siguientes:

1. Estudiar en la literatura científica técnicas clásicas de preprocesado de datos georreferenciados en niveles de detalle (LoD), de retransmisión por *streaming* de los datos y otras de interés como tratamiento de nubes de puntos o mallado triangular. El objetivo es filtrar las características más sobresalientes de cada método.

2. Proponer dos técnicas para el preprocesado de grandes conjuntos de datos georeferenciados, generando estructuras que permitan optimizar su transmisión. Estas estructuras además deben permitir la generación de diferentes niveles de abstracción de los datos de cara a una visualización progresiva de los datos desde cualquier dispositivo.

3. Plantear un esquema de *streaming* y visualización adaptable a cualquier dispositivo de escritorio o móvil. Este esquema deberá permitir una transmisión progresiva e incremental del conjunto de datos a partir de un subconjunto reducido de los mismos. Con cada nueva transmisión, se incrementará el nivel de detalle del conjunto en el dispositivo. De este diseño se deberán incluir detalles de implementación así como un análisis y validación de resultados.

4. Determinar indicadores objetivos que demuestren la validez y eficiencia de los métodos planteados así como realizar comparativas con otros métodos similares del estado del arte.

Aparte de estos cuatro objetivos primarios, se busca también como objetivos adicionales de este trabajo de investigación publicar y contrastar los resultados alcanzados con la comunidad científica a través de congresos especializados, en primer lugar; y el poder realizar transferencias del conocimiento generado durante el mismo a empresas y entidades del sector turístico, geomático o de cualquier otra índole que pueda beneficiarse del mismo, en segundo lugar. Estas transferencias se harán en forma de prototipos o aplicaciones informáticas concretas que ayuden en las actividades diarias de dichas empresas y entidades. De esta forma se busca que la sociedad se beneficie de este trabajo de investigación.

Atendiendo a estos objetivos y en función de las contribuciones que se han aportado durante este trabajo de investigación, este documento se ha dividido en un primer capítulo de introducción a la tesis y en ocho capítulos posteriores de contenido temático. Cada capítulo incluirá una sección sobre el estado del arte del tema a tratar y tras él, la contribución concreta realizada. Como guía para el lector, se facilita un resumen de cada uno a continuación:

Capítulo 2: Este capítulo se centra en la primitiva punto, por ser la más común y de la cual existen más y mayores conjuntos de datos vectoriales. Se presenta la problemática que supone intentar mostrar a la vez muy pocos o bien demasiados puntos, sobre todo si se mostrarán simbolizados con un nombre o una imagen (*marker*). En base a esta problemática, se propone un algoritmo de preprocesado orientado a la transmisión eficiente y visualización adecuada de datos puntuales. Con el objeto de dar soporte a datos que puedan tener escala global, se propone una estructura de nivel de detalle con árbol quadtree en lugar de las más comunes octree o binaria. Para un envío progresivo de los datos, se estudian dos estrategias diferentes: *sorting* y *clustering*. Finalmente, se implementa una arquitectura cliente-servidor para la transmisión de datos a visualizadores móviles de globo virtual de estos datos.

Capítulo 3: En este capítulo se exploran las líneas, un dato vectorial frecuente en datos geográficos, de redes y de ciudades inteligentes. Utilizando datos georeferenciados de redes subterráneas de abasto, gas y electricidad, se explora la mejor manera de presentar dichos datos en un escenario de globo virtual de forma que el usuario interprete correctamente que se tratan de redes subterráneas. Para ello, técnicas del estado del arte como el alpha-blending o las herramientas de excavación son investigadas. Así mismo se propone una nueva: los ditches, consistentes en la adición de una malla con forma semicilíndrica alrededor de la línea a representar. La idoneidad de cada técnica se investiga mediante encuestas de experiencia de usuario con usuarios con y sin experiencia técnica en la materia.

Capítulo 4: En este capítulo se exploran los polígonos, la tercera primitiva vectorial y que es fácil de encontrar en modelos 3D o en parcelas de catastro, entre otros ejemplos. También se presenta aquí por primera vez en el documento la tecnología LiDAR, que permite generar nubes de puntos potencialmente muy grandes capaces de representar detalladamente la realidad. Utilizando ambos, se presenta un método capaz de generar modelos 3D de una ciudad a partir de una representación de la ciudad como nubes de puntos. Los modelos son generados utilizando el estándar CityGML, pensado específicamente para poder almacenar diferentes niveles de detalle de los datos de entrada. Se contempla que los modelos sean compatibles con el nivel LoD2 del estándar, para lo cual se diseña un algoritmo de preprocesado capaz de identificar el tipo de tejado de cada edificio entre 5 posibles categorías y generar el modelo correspondiente. Esto permite reducir mucho el volumen de datos a mostrar sin perder los detalles clave de cada edificio, lo que abre la puerta a su uso en simulaciones y aplicaciones orientadas a smart cities, realidad virtual y aumentada.

Capítulo 5: Las nubes de puntos, que se utilizaron en el anterior capítulo para generar modelos 3D, tienen un gran número de aplicaciones prácticas por sí solas, que generalmente implican el segmentado previo de cada punto. Además, son en sí mismas grandes conjuntos de datos georeferenciados. En este capítulo se presenta un nuevo método de segmentación de puntos pertenecientes al terreno para nubes generadas en vuelos de adquisición LiDAR. Se basa en la creación de *patches*, agrupaciones de puntos de altura mínima. De cada *patch* se extraen diferentes características, las cuales se utilizarán en un árbol de decisión

para determinar si pertenecen o no al terreno. La evaluación de la idoneidad de este algoritmo se ha realizado sobre *benchmark*s para los cuales hay resultados previos de otros algoritmos del estado del arte. Finalmente, se presenta un caso de uso de los resultados generados por el algoritmo para la creación de mallas triangulares de elevación de terreno multiresolución con diferentes niveles de detalle.

Capítulo 6: Los puntos aún no clasificados como terreno representarán otros objetos del mundo real, como vegetación, edificios, coches o farolas. Adicionalmente, puede suceder que la nube no haya sido adquirida en vuelo sino desde el suelo. En este capítulo, se adapta el algoritmo de suelo visto en el capítulo anterior para su uso en nubes urbanas adquiridas con un sistema LiDAR terrestre. Además, se presenta un algoritmo de preprocesado progresivo no supervisado de los puntos restantes en las clases coche, poste, vegetación y edificio. La evaluación del método se realiza sobre un *benchmark* de nubes urbanas terrestres contra varios algoritmos del estado del arte.

Capítulo 7: En este capítulo se continúa con la segmentación de nubes de puntos, proponiendo un algoritmo capaz de categorizar los diferentes componentes de un corredor eléctrico. La novedad con respecto a los algoritmos anteriores del estado del arte estriba en que no se limita sólo a clasificar torretas y cables, sino que entra en un detalle superior identificando cables guía, cables conductores, puentes y cadenas de aislamiento. El método se evalúa en un *benchmark* contra un algoritmo de estado del arte, y adicionalmente se propone un *benchmark* propio para la evaluación. Una vez identificados estos elementos, se proponen también métodos para generar modelos vectoriales 3D de cada cable y cada torre a partir de la clasificación. Esto se hace con la doble idea de hacer más eficiente el envío por red de los datos del corredor y para facilitar el uso práctico de estos datos en inspección de líneas eléctricas y prevención de incendios forestales.

Capítulo 8: Todos los métodos para preprocesado de nubes de puntos vistos en los anteriores capítulos suponen nubes de puntos que representen fielmente la realidad. Sin embargo, en muchas ocasiones las nubes de puntos incluyen ruido, que conviene eliminar antes de poder utilizarse. En este capítulo, se identifican diferentes tipos de ruido presentes en nubes de datos LiDAR y se propone una metodología para el filtrado conjunto de todos los tipos de ruido observados. Cada tipo de ruido es tratado por un detector específico, lo que permite lanzar detectores de diferentes tipos de ruido a la misma nube en paralelo y combinar cada resultado individual para ofrecer una solución conjunta. Además, se crea un *benchmark* abierto de nubes de puntos con todos los tipos de ruido identificados para la validación de los resultados.

Capítulo 9: En este capítulo se introduce, por último, un visor ligero de escritorio para nubes de puntos desarrollado en C++ como respuesta a necesidades surgidas del trabajo en el resto de capítulos. En el visor propuesto se implementan los controles de cámara necesarios para la navegación. Se añade una estrategia de nivel de detalle para la visualización basada en celdas, a cada una de las cuales se le aplica un factor de diezmado diferente en función del área de pantalla y de la inclinación de vista. Finalmente, se introducen tres modos

de selección de puntos con vistas a su posible edición. Dos de ellos, rectángulo y caja, son conocidos y comunes en el estado del arte. El tercero, caja centrada, es una modificación del modo de caja propuesto con el objetivo de limitar la selección al objeto deseado. La experimentación introducida busca encontrar la mejor forma de ajustar el tamaño de la celda y los factores de área de pantalla e inclinación de vista en la estrategia de LoD elegida.

Finalmente, en un décimo capítulo se resumen las aportaciones y conclusiones obtenidas durante esta tesis doctoral y las futuras líneas de investigación que se han abierto a raíz de este trabajo. Además, se incluyen los diferentes casos de uso y aplicaciones comerciales a los cuales se han transferido alguna de las diferentes propuestas aquí expuestas.

Aportaciones realizadas

En las siguientes líneas se presentan las aportaciones clave que se han realizado como resultado de este estudio:

En primer lugar, se introdujo una técnica específica para generar estructuras en nivel de detalle a partir de conjuntos de puntos que pudiera potencialmente contener información de todo el globo terráqueo. Debido a esta característica particular, se decide basar la estructura en un árbol quadtree asociado a una división en rejilla regular del planeta (DGGS) en lugar de en los más comunes árboles binario, kd y octree, usados para generar LoD sobre conjuntos de puntos. Para generar los diferentes niveles de detalle del conjunto, se utilizaron dos estrategias diferentes. La primera, *sorting*, escoge un criterio de ordenamiento de los datos y reparte los puntos existentes entre los diferentes nodos no-hoja creados en el árbol. Esta estrategia está orientada a *streaming* progresivo, en la cual se añade detalle del mismo set con la navegación sin crear ningún dato nuevo. La segunda estrategia, *clustering*, genera un metadato indicando el número de puntos existente en un área concreta del mapa en representaciones poco detalladas del conjunto. Esto aumenta ligeramente el número de datos a transmitir pero favorece una visualización limpia del set de datos.

En segundo lugar, se desarrolló una arquitectura cliente servidor capaz de transmitir las estructuras de datos puntuales anteriormente generadas a dispositivos móviles para su visualización mediante símbolos o *markers*. La arquitectura requiere un servlet capaz de leer de la base de datos espacial, una API para la consulta de los datos y un cliente móvil capaz de pedir bajo demanda los datos para cada nivel de detalle incremental en el área de navegación y generar la simbología necesaria para representarlos. Para esto, se ha adaptado un motor de globo virtual existente, Glob3 Mobile [1], para renderizar los *markers* de forma eficiente. Finalmente, se realizó un estudio experimental para determinar la cantidad adecuada de elementos en pantalla a mantener en el visor para una mejor comprensión de la visualización. Los resultados, obtenidos para un set de datos puntuales abierto, extenso y con información de todo el globo y utilizando un móvil de gama media, sugirieron que la mejor opción es llenar con contenido aproximadamente el 30% del total de la pantalla y mantener los nodos del árbol

propuesto limitados a un máximo de 8 *markers* de punto con texto.

Continuando con la línea de buscar una mejor comprensión de la visualización, la tercera aportación realizada es un estudio para encontrar la mejor manera de mostrar en pantalla conjuntos de redes subterráneas de tuberías de diferentes tipos, representadas como polilíneas, en un entorno de globo virtual. Tomando como referencia el estado del arte y el mismo motor de globo virtual Glob3Mobile, se implementaron técnicas de alpha blending variable, difuminando más la tubería conforme se aleja de la posición de vista. En total, ocho funciones matemáticas diferentes fueron analizadas para encontrar el mejor efecto de difuminado por distancia. Adicionalmente, se implementó una versión de otro método de referencia, la excavación, para globo virtual, y se diseñó un nuevo método de visualización, el *ditch*. El ditch es un mallado semicilíndrico generado alrededor de una tubería. Combinado con una textura adecuada, este mallado consigue un efecto de acequia en la visualización que puede dar la sensación de profundidad deseada. Tras un experimento de experiencia de usuario, se determinó que el método que daba mayor sensación de subterraneidad para el escenario de pruebas era la herramienta excavadora, seguida de cerca por el alpha blending variable con función softsign. La visualización de tipo ditch crea sensaciones muy variables, siendo muy fácilmente entendible por usuarios con experiencia técnica pero muy poco comprensible para los que no la tenían. Por último, todas las metodologías probadas ofrecieron una mejor comprensión que la técnica de referencia más común: el alpha blending con valor de transparencia estable.

Una cuarta aportación ha sido una metodología para la generación automática de modelos 3D de cada edificio de una ciudad a partir de nubes de puntos LiDAR y polígonos de parcela obtenidos de la base de datos libre OSM. Los modelos generados para cada edificio siguen el estándar CityGML, que contempla niveles progresivos de detalle. Se alcanza un nivel LoD2, que ofrece superficies poligonales para cada pared y cada cara del tejado del edificio. Debido a que cada polígono de OSM puede contener uno o varios edificios diferentes en su interior, se establece una primera etapa que los identifica y los separa cuando es necesario. Para cada uno de estos edificios separados, un nuevo polígono se genera automáticamente utilizando un nuevo algoritmo de generalización de líneas basado en detección de esquinas. Tras esto, el método determina de entre cinco categorías, *plano*, *inclinado*, *a dos aguas*, *piramidal* o *complejo*, la que mejor describe el tejado de cada edificio, lo que es necesario para generar un modelo adecuado. Esto se lleva a cabo mediante extracción de planos y de un sistema de reglas en función de los planos detectados, las intersecciones entre éstos y variables estadísticas obtenidas a partir de los puntos de entrada. Una primera evaluación del método aplicada a un conjunto de datos representativo de una ciudad de tamaño mediano demostró resultados prometedores para la identificación de las diferentes categorías de edificio. Finalmente, se demostró que la técnica ofrece una elevada reducción del número de datos a transmitir para representar el mismo contenido: el modelo final de ciudad en CityGML ocupó sólo un 8.5% del tamaño original de la nube.

En la quinta aportación, otra metodología fue planteada para la clasificación

de puntos de terreno en nubes LiDAR con vista tanto a trabajos que requieran segmentación de la nube como a la generación de modelos de elevación de terreno. En ella se presenta un nuevo concepto, el *patch*, que es un clúster de puntos de altura local mínima en la nube. Los *patches* se generan mediante *clustering* jerárquico basado en distancia y un filtro anisotrópico para dar mayor relevancia a la dimensión de altura. De cada *patch* obtenido, se generan medidas estadísticas basadas en sus puntos y en mapas ráster de características multiescala. Con estas medidas, un árbol de decisión es definido para determinar la pertenencia o no de cada *patch* al terreno. Finalmente, y utilizando los puntos así clasificados, un algoritmo es propuesto para generar mallas de triángulos regulares con niveles de detalle progresivamente menor para su utilización como modelos de elevación de terreno. El algoritmo propone la resolución de un sistema de ecuaciones para encontrar alturas para los vértices de mallado regular deseado de forma que se ajusten a los datos de entrada. Para evitar errores máximos demasiado altos, una etapa posterior reajusta aquellos vértices que contribuyan más al error global de la malla. Esta metodología de clasificación de puntos de terreno se comparó contra varios algoritmos del estado del arte en un *benchmark* urbano de referencia, obteniendo los mejores resultados de entre todos los métodos que utilizan únicamente una nube de puntos como punto de partida.

La sexta aportación de esta tesis busca otorgar un significado semántico a aquellos puntos de las nubes no clasificados como terreno. Partiendo de nubes LiDAR capturadas desde el suelo, se introduce una metodología de trabajo no supervisada que clasifica estos puntos en otras cuatro categorías: edificio, vegetación, postes y coches. Para ello, utiliza un esquema de detectores progresivos dedicados a una única clase, donde la entrada de una etapa es la salida de la anterior. Los detectores utilizan mapas ráster de características extraídas a partir de los puntos presentes en cada celda del ráster para determinar las áreas donde pueden potencialmente haber postes y coches, y mediante técnicas de agrupamiento de los puntos determinan cuáles de estas en efecto incluyen estos datos. Los puntos restantes también son agrupados, y se determina la pertenencia a edificios o vegetación de cada grupo en función de un algoritmo recursivo de extracción de planos. La técnica propuesta fue comparada con varios algoritmos similares del estado del arte basados en redes neuronales usando un *benchmark* urbano específico de nubes terrestres. Los resultados mostraron que la detección progresiva de clases con métodos no supervisados fue capaz de mejorar a todas las técnicas basadas en aprendizaje supervisado salvo una, demostrando su competitividad para clasificación semántica de nubes de puntos urbanas.

Continuando en esta línea, una séptima aportación realizada es una propuesta para la clasificación y el modelado 3D de elementos pertenecientes a corredores eléctricos a partir de nubes de puntos. El algoritmo de preprocesado comienza con una búsqueda del corredor en la nube, utilizando combinaciones de mapas ráster de características a partir de los datos de altitud e intensidad de los puntos, así como de la cantidad de puntos presentes en un área local de la nube. Esto devolverá una clasificación inicial de torres y cables eléctricos,

similar a las ya existentes en otros métodos del estado del arte. A partir de este punto y mediante técnicas de *clustering* y ajuste de los puntos a ecuaciones de línea, se refina la clasificación en las áreas donde cable y torre se usen y se introducen detectores para clasificar subcategorías dentro del corredor: aisladores, que son una subcategoría dentro de la torre; y conductores de cadena puente, cables guía de toma de tierra y cables conductores, que son subcategorías dentro del cable. Teniendo en cuenta el estado del arte en el momento de escribir este capítulo, la expuesta aquí es la primera propuesta realizada para la diferenciación de las subcategorías de un corredor eléctrico en nubes de puntos, al menos hasta donde tenemos conocimiento. Debido a esto, la comparación contra otros métodos del estado del arte sólo se ha podido realizar al nivel general, aunque se ofrecen también unos resultados iniciales prometedores para cada subcategoría realizada.

Dentro de esta línea de trabajo, también se ofrece un método para segmentar individualmente cada cable y cada torre detectada en el corredor y generar, a partir de ellos, modelos vectoriales en 3D. El modelo generado para cada conductor corresponde a los parámetros que definen una curva catenaria en tres dimensiones: un punto de origen o, otros dos puntos por los que el cable queda suspendido en el aire y un parámetro de torsión, a. Los parámetros o y a son obtenidos mediante optimización multivariable PSO a partir de los puntos de cada conductor en la nube. El margen de error RMSE de cada modelo así obtenido alcanza los requisitos necesarios para su uso en aplicaciones industriales. A su vez, un modelo vectorial es generado para la torre tras una identificación automática del número de brazos presentes en la misma. Obtener estos modelos permite una reducción muy importante del número de puntos necesarios para poder representar un conductor en cualquier visor: en lugar de enviar los cientos o incluso miles de puntos de la nube, pueden enviarse 3 puntos y un parámetro numérico que permitirán mostrar en todo momento el cable con cualquier nivel de detalle. Idéntica solución puede realizarse para el poste y los brazos de la torre de alta tensión, pues se reduce la torre a una línea central vertical y a un número de líneas horizontales equivalentes al número de brazos del cable. Finalmente, los modelos pueden ser aprovechados por sí mismos para cálculos posteriores de anomalías y llevar a cabo tareas periódicas de mantenimiento en el área del corredor.

La octava aportación de esta tesis busca poder aprovechar nubes de puntos afectadas por ruido en todos los procesos propuestos anteriormente. Para ello, se identifican hasta 7 esquemas diferentes de ruido presente en las nubes de puntos en función de su apariencia visual y se propone una metodología de procesado en paralelo para filtrar cada una de ellas por separado. Para estabilizar el tiempo de ejecución, la nube se divide en rodajas y el filtrado se realiza en cada detector mediante técnicas de *clustering* y características basadas en intensidad, posición y distancia punto - emisor LiDAR. Los resultados parciales de cada detector se unifican en una única salida de ruido filtrado. Finalmente, se propone un nuevo *benchmark* compuesto por 20 nubes de puntos afectadas con ruido, con el cual se ha validado la propuesta realizada. Los resultados obtenidos han sido prometedores y muestran su utilidad para el filtrado de la gran mayoría del

ruido presente en las nubes de puntos.

Y finalmente, la novena y última aportación realizada en esta tesis fue un visor ligero para nubes de puntos, surgida como necesidad a partir del trabajo diario en el resto de capítulos. En ella se implementaron controles de navegación adaptados para nubes de puntos, tres métodos de selección de puntos en escena 3D, siendo uno de ellos, la *caja centrada*, novedoso, y un esquema de LoD para una carga fluida y bajo demanda de la nube. Esta estrategia se basa, al igual que en el caso de los *markers*, en una rejilla regular en lugar de en un árbol octree. No obstante, al no ser necesario un servicio de red en este visor, se escogió un diezmado en función del tamaño de las celdas, la inclinación de vista y un área de pantalla de referencia como forma para introducir más o menos detalle en la visualización. La experimentación llevada a cabo demostró mayores similitudes con una versión sin diezmar cuando el tamaño de celda es escogido en función de la densidad de puntos de la nube, el área de referencia se define de gran tamaño y el parámetro de importancia de la inclinación se mantiene en un rango estable.

En conclusión, como resultado de este trabajo se han obtenido:

- Dos estrategias novedosas para la generación de niveles de detalle aplicables a conjuntos de puntos y a generación de modelos de terreno.
- Dos estrategias novedosas para la generación de modelos 3D vectoriales reducidos a partir de nubes de puntos, con vistas a aplicaciones de smart cities e inspección de líneas eléctricas.
- Cuatro metodologías novedosas diferentes aplicadas a la segmentación en categorías diversas de grandes volúmenes de datos puntuales en forma de nube de puntos.
- Una arquitectura de transmisión progresiva de conjuntos de puntos estructurados en nivel de detalle quadtree a aplicaciones móviles de globo virtual.
- Y finalmente, tres estudios que analizan las mejores maneras de mostrar información georeferenciada al usuario en forma de *markers*, líneas de tuberías subterráneas y nubes de puntos.

Estos resultados han sido expuestos y revisados ante la comunidad científica, dando lugar a tres artículos expuestos en revistas indexadas, un capítulo de libro y cuatro artículos presentados en conferencias internacionales. El listado completo de los mismos se ofrece en el Anexo I. Finalmente, se ha conseguido transferir gran parte de las aportaciones aquí expuestas a diversas aplicaciones en los ámbitos turístico, educativo, investigador e industrial. Estos diferentes casos prácticos de uso de esta tesis han formado a su vez una parte de los contenidos de otro artículo en revista indexada, otro capítulo de libro y seis artículos expuestos en conferencias especializadas. Con estas transferencias se consiguen completar, al menos desde nuestro punto de vista, todos los objetivos marcados al inicio de esta tesis doctoral.

Agradecimientos

Quiero utilizar estas líneas para expresar mi agradecimiento hacia todas las personas que, directa o indirectamente, han tenido un papel en que yo pudiera completar este trabajo de investigación durante estos cuatro años.

El primero y más grande de mis agradecimientos es para mis directores de tesis, **José Pablo Suárez Rivero** y **Agustín Trujillo Pino**. Ellos me dieron la oportunidad de comenzar esta línea de trabajo y me han ido guiando por todos los aspectos de la misma. Sus consejos, desde su experiencia y pericia, lograron sacarme de muchos atascos sobre cómo abordar los problemas del trabajo diario: qué descriptor puede ser mejor para detectar un cable bajo, o cuál es la mejor manera de simplificar una polilínea, entre otros muchos. También me mostraron las mejores maneras de estructurar un artículo científico y dar a conocer los resultados que hemos ido consiguiendo. Ha sido mucho lo que he aprendido trabajando con ellos, y mucha la paciencia que ellos han tenido conmigo. Para ellos va mi más sincero reconocimiento.

En segundo lugar, quiero acordarme de mis compañeros de laboratorio y de todos los coautores de mis trabajos, tanto los de la ULPGC como los externos. Mención especial aquí para **José Miguel Santana**: él me animó a dedicarme a investigar, me recomendó para el grupo de investigación, y ha sido constante fuente de debates interesantes en todos nuestros trabajos comunes, como ¿deberíamos usar redes neuronales para este problema? o ¿es ésta la mejor manera de expresar esta aportación en inglés?. No me olvido tampoco de **Daniel Santana**, dios del LaTeX y aún mejor persona, ni de **Cristian Ramírez**, compañero de fatigas durante la implementación de todos los prototipos para empresas. Además, tuve la suerte de contar durante mis estancias en Alemania con **Fran Marzabal** y **Jaisiel Santana**, que hicieron mucho más fácil mi adaptación allí y siempre están dispuestos a conversar sobre Ingeniería del Software, Fotografía y lo que se tercie. Quisiera finalmente en este punto mencionar a todos los profesores miembros del **Centro de Tecnologías de la Imagen de la ULPGC**, que me acogieron en su laboratorio de investigación y me han prestado su ayuda siempre que la he necesitado.

En tercer lugar, quiero mencionar a **Tomás Herrera**, **Francisco Suárez** y el resto del personal de **Aerolaser System, S.L**. Ellos aportaron una visión industrial sobre la temática de nubes de puntos, que estábamos empezando a explorar. Nos sugirieron también posibles vías de trabajo que harían más valiosa nuestra investigación, y, aún más importante, nos dieron acceso a todas

las nubes de puntos que pudiéramos desear.

En cuarto lugar, y de forma especial, quiero nombrar a mi familia. A **mis padres**, que me han dado su apoyo incondicional y me han ofrecido un modelo a imitar durante toda mi vida. A **mis hermanas**, capaces tanto de escuchar sin quejas mis alegrías y mis frustraciones como de hacerme preguntas sobre lo que hago desde una óptica muy diferente. Más de una solución expuesta aquí surgió como idea tras un intento de contestar esas preguntas. Y a mis **tíos y primos**, que se prestan con alegría a encuestas, tests y cualquier cosa que me haga falta. Mención especial aquí también a la aventura familiar fallida de **El Nublo. Turismo Sostenible S.L.**: sin esos senderos, no me hubiera puesto a curiosear Glob3 Mobile; y curiosear Glob3 Mobile me llevó a lo que acabó siendo una tesis doctoral. En mi vida, todas las buenas historias siempre comienzan con un pateo, y ésta no fue una excepción.

Y por último, tendré un pequeño recuerdo para mis amistades, que me han sufrido a mí y a mi incomprensible trabajo: **Aday, María, Javier, Eduvigis, Laura, Josué, Beatriz, Alejandro, Carlos, Ester y Zenaida**. También para mi ex-pareja, **Omaira**. Personas, todas ellas, que entre pateos, quedadas y conversaciones nocturnas contribuyeron a mantenerme cuerdo en al menos una ocasión durante estos cuatro años.

Soy consciente de que es muy probable que se me quede gente sin nombrar, así que y en general, si estás leyendo estas líneas y me has conocido, te extiendo también mi agradecimiento. A todos,

¡Gracias!

A mis abuelos, in memoriam,

a los que le hubiera gustado ver el final

de la senda que siempre me animaron a caminar.

Contents

Chapter 1

Introduction

Our world becomes more digital every day. Wherever you look, you can see lots of electronic devices capable of providing you with information and capture data about the environment. Those devices go from televisions and computers to smartwatches, without forgetting others like mobile phones and tablets. All of them are equipped with screen, network functionalities and different sensors which collect, between other data, your position. Additionally, there are huge amounts of sensors distributed around any place on the Earth. Those sensors are capable of reading and continuously stream any kind of physical properties, like temperature, humidity, pressure, radiation or displacement, between others. Again, all of them are associated with a certain position. These devices generate enormous and always growing volumes of geo-referenced data that should be properly managed.

A *geo-referenced* or *spatial* datum is considered to be any datum associated with a certain position on the planet. For this association, a *spatial reference system* (SRS) is generally used to express this position in terms of a tuple of coordinates. The coordinates can be geographic (latitude, longitude, height) or projected cartographic (x,y,z).

Geo-referenced data can be divided into two categories: vector data and raster data. Vector data are geometrical variables, points, lines and polygons, that are associated with a position on the Earth. On its part, in raster data the pixels of an image or the cells of a regular grid are the ones to be associated with a location in the world. Aerial and satellite photographs, cartographic maps or digital elevation models fall into the raster data category.

This work focuses mainly on vector data. These data are commonly generated using three types of technology: the *global positioning systems* (GPS), the *remote sensing* technology and the *geographic information systems* (GIS). GPS are systems that determine instantaneously and accurately the terrestrial location of a device, via triangulation with respect to several different satellites whose location is always well-known. This location can be used alone as a single point or can be combined with extra information. In remote sensing, the spatial data are created by sensors that analyze the surrounding environment

without keeping contact. Those sensors can be active, which means a signal is sent against the environment, or passive, limiting themselves to gather the information. The best known remote sensing technologies for vector data are active: *RADAR* and *GPR* to measure distances in aerial and subsurface environments respectively using radio waves; *LiDAR* to measure distances using light beams, and *SONAR* to measure distances using sound waves, generally in sub aquatic environments. The outputs of such sensors usually come in the form of point clouds with very variable volume and density. Finally, the GIS are information systems that combine hardware and software elements so a user can create, gather, edit, organize, convert, model, save and query spatial data. Such information systems go from full open and privative platforms (ESRI ArcGis, QGis, Grass, Autodesk, Capaware) to libraries of functions (proj4, GDAL, GeoTools, LasTools), without forgetting specific servers for geo-referenced data (Geoserver, Mapserver).

When large sets of vector data need to be managed, problems arise for their storage, their transmission through the network and their visualization. This is specifically true when the data is intended to be displayed or captured in a portable device. Mobile devices normally have limitations on memory and computational power. This makes keeping entire data sets of any kind very difficult. Moreover, they have limited screen sizes. This fact forces a careful management of the amount of information to show on screen, so no overlapping or cluttering occurs and the user does not become saturated. Due to all these reasons, it is advisable to limit the amount of data to transfer to the device at any moment and do it only under demand of the user.

In this doctoral thesis, new methodologies for the processing of large volumes of spatial data are introduced. They aim to ease the progressive and selective streaming by generating level of detail structures out of the data, and to improve the understanding of such data.

1.1 Thesis goals and organization

In this dissertation, new methods for processing of large geo-referenced data sets or *Big Geo Data* are proposed. In those methods, the data input, their efficient access and storage, their streaming and their proper visualization must be taken into account. Moreover, they should keep the coherence in the representation and display of the results: they should not generate singularities or deformations with respect to the input data.

In order to achieve the desired efficiency in the storage, transmission and rendering of the data, strategies of level of detail should be introduced, so different and progressively less detailed versions of the same dataset coexist. These representations should be interchangeable according to the importance or the perspective from the point of view of the user. Moreover, a streaming architecture that allows to query the data on demand is required. Using a virtual globe application as an example and a spatial dataset which we desire to show on the globe, more detail should be downloaded and added in the areas of the globe

the user visits.

With all these considerations in mind, the main objectives of this research work have been defined as the following:

1. Studying the classical techniques of *level of detail* (LoD), progressive streaming and others of interest such as triangle meshing or point cloud processing. The objective is to filter the most highlighted findings in the state of the art on these topics, in order to process geo-referenced data.

2. Proposing, at least, two methodologies for the preprocessing of large spatial data sets, generating structures which optimize their further transmission. These structures should allow the generation of different levels of abstraction of the data for a progressive visualization from any device.

3. Proposing, at least, a streaming and visualization scheme which adapts to desktop and mobile applications. This scheme should allow an incremental streaming of new data from a reduced subset. With each new transmission, the level of detail of the subset in the viewer must be increased. Regarding this architecture, details of implementation should be provided as well as an analysis and validation of the results.

4. Determining objective indicators to demonstrate the efficiency and viability of the proposed algorithms, as well as for comparisons with similar algorithms from the state of the art.

Apart from the four main objectives, it is also expected as an extra objective of this research work to transfer part of the generated knowledge. These transferences will be made to different companies and entities in touristic, geomatic or other fields that may take advantage of them. The transferences will be done as prototypes or applications that help the daily activities of the commented companies and entities. This way, the society benefits from this research work.

According to the contributions that have been made to address these objectives, this document has been divided in eight different chapters. Each one of them includes a related work section with the state of the art on the covered topic. After that, a contribution on the topic is detailed. As a guide for the reader, a summary of each chapter is introduced in the following paragraphs:

Chapter 2: This chapter is focused on the *point*, the most common type of vector data, of which there are more and larger data sets. The issues regarding displaying very few or too much points on screen is introduced, specifically for the case in which they will be shown symbolized as a name or an image (a *marker*). To address those issues, a preprocessing algorithm is proposed for an efficient transmission and a proper visualization of punctual data as markers. In order to support data sets of global scale, a level of detail structure based on a quadtree is proposed instead of the common binary and octree solutions. For the generation of the coarser levels of detail, two different strategies are followed: sorting and clustering. Finally, a client-server architecture for transmission of data to virtual globe mobile applications is developed.

Chapter 3: This chapter explores the lines, a type of vector data frequent in geography, networks and smart city applications. The best manner to present geo-referenced data sets related to underground water, gas and power networks is studied so the user correctly understands they are placed below the surface. To do so, existing techniques such as α-blending and excavation tools are considered. Additionally, a new one, the *ditch*, was designed. This novel method adds a semi-cylindrical mesh around the pipeline before its display on screen. The suitability of each explored method is analyzed with user experience surveys with people with and without technical knowledge.

Chapter 4: In this chapter, polygons are explored. Polygons are the third type of vector data and are found in 3D models or cadastre footprints, among others. The LiDAR technology is also used for the first time in the research. LiDAR enables generating potentially enormous point clouds which represent the real world in a detailed manner. Using both of them, a method capable of creating 3D city models is proposed, using a point cloud representation of the city as the inputs. Models are generated using the CityGML standard, which integrates support for level of detail. The chosen level of detail for the models was LoD2, which supports polygonal surfaces to represent the rooftop of the building. The designed preprocessing pipeline categorizes the type of rooftop between 5 possible classes and generates the corresponding rooftop model. This reduces heavily the volume of data to display without losing the key details of each building, which opens the possibility of its use in simulations and smart city, virtual reality and augmented reality applications.

Chapter 5: The point clouds, which have been used in the previous chapter for the generation of 3D models, are large geo-referenced data sets and have a great number of practical applications by themselves. These applications normally require a previous segmentation of the points. In this chapter, a new ground classification method for aerial LiDAR point clouds is proposed. It relies on the generation of *patches*, groups of points of minimum local height. From each patch, different feature descriptors are extracted and passed as input for a decision tree classifier, which determines whether the patch represents terrain or not. The suitability of this algorithm is analyzed by comparing its behaviour with the results of previous methodologies on a benchmark. Finally, a use case of the generated results for the creation of multi-resolution triangular regular meshes is presented. Such meshes have utility as digital elevation models with progressively coarser levels of detail.

Chapter 6: Points that are not related with terrain represent other objects of the real world, as buildings, vegetation, cars or poles. Moreover, the point cloud can be acquired from a vehicle instead of a flight. In this chapter, the patch decision tree algorithm is adapted for vehicle-borne clouds, and then a progressive, non-supervised segmentation algorithm is proposed for classifying the remaining points into the classes car, pole, building and vegetation. The validation of this methodology is conducted on a benchmark that contains urban vehicle-borne point clouds. Comparisons with other proposals are also described in the chapter.

Chapter 7: This chapter continues the line of point cloud segmentation. A

pipeline for the classification of different components of a power line corridor in LiDAR point clouds is proposed on it. The novelty with respect to prior methods lies in their ability to identify smaller pieces as insulator strains, bridge chains and shield wires instead of classifying only wires and pylons. The behavior of the new pipeline is analyzed with tests conducted on a benchmark for which prior results of a reference method are available. Additionally, a new benchmark is proposed. Once the corridor elements are categorized, an algorithm for the generation of 3D vector models from each detected wire and pylon is also introduced. The objective is to make more efficient the network transmission of the corridor information and to ease the practical use of such data in power line inspection and forest fire prevention tasks.

Chapter 8: All the previously introduced methods for classification in point clouds assume a clean input that represents accurately the real world. However, the acquisition process often fails and the point clouds include noise. This noise must be filtered prior to using the point cloud. In this chapter, different types of noise that can be found in LiDAR point clouds are identified, and a methodology for their removal are proposed. Each type of noise is there filtered using a specific detector. Different detectors can be used in parallel when more than a type of noise is present in the point cloud. In this case, the final output is a combination of all the different filtering results. Finally, a new open benchmark of noise-affected point clouds is created, published and used to validate the proposed methodology.

Chapter 9: As a final contribution, in this chapter a lightweight viewer of point clouds is introduced. It was designed to address some needs of the daily research work. In the proposed viewer, camera controls have been implemented for the navigation towards the cloud. A LoD strategy for visualization based on a regular grid has been added. Levels of detail are generated per grid cell by applying a decimation factor which varies according to the screen area and the pitch angle. Finally, three selection modes have been integrated in the viewer for point edition purposes. Two of them, *rectangle* and *box*, are very common in other point cloud dedicated software. The third one, *centered box*, is a modified version of *box* which aims to better fit the selection to the object the user expects to be selected. The conducted experimentation looks for finding the best manner to adjust the grid cell size and the factors of reference screen area and pitch importance on the chosen level of detail strategy.

To close this document, a final chapter resumes the contributions and conclusions achieved during this research work and the future lines of work that have been opened from them. Moreover, the different use cases and commercial applications in which the different proposals of this work have been transferred are enumerated in the final chapter.

Chapter 2

Structuring, transmitting and efficiently displaying point datasets

Points are the most frequent type of vector data. They are normally expressed as a tuple of 2 or 3 dimensions expressing the spatial coordinates and, optionally, associated information of all kinds. They can symbolize geographical data, like cities, towns, mountains or rivers. It can also represent points of interest or current placements for people, animals and objects. They can even be data relative to measurements or active and passive sensors of all types, among multiple other uses. Individually, a single point does not normally say too much, but a set of multiple points can be a really powerful tool to understand and communicate the reality.

Figure 2.1: Glob3 Mobile web visualization of a dataset representing cities in the world. Displaying the raw dataset produces the undesirable cluttering effect.

In order to generate useful information from point data, the data should be processed, analyzed and displayed in their proper context. Without such an analysis, data has little value. Let it be an example of a point data set containing the populated places of the world, whose obvious potential use is for knowing their placements, names and sizes. But, as no analysis has been previously done, the raw data set is displayed as is on a map. Each point is represented as a red dot in the map, so you know where the cities are. The result of doing so can be seen in Figure 2.1.

As it can be seen, the whole of Europe is a red stain. You do not really know whether it is a single and continental-sized city or multiple cities. In case there are multiple cities, you do not know the amount of them. There is no context, no information. This phenomenon is called *cluttering* and must be avoided. Another factor is that, for such a representation of nothing, all the content of the point data set has been transferred to the viewer of your choice and then drawn in its screen, generating unnecessary costs in memory, processing and network usage. In order for this data to be useful, it should be shown resumed and aggregated at first, and then add detail on demand as the user explores the scene.

This chapter explores how to avoid cluttering and efficiently transmit and display a point marker data set with spatial information and two properties for name and size, creating a server-client architecture for the progressive transmission and visualization of the markers in mobile environments. In this work, already introduced in [2], level of detail strategies and cluttering reduction techniques such as sorting and clustering have been applied to generate a structured database out of the input data set. A client implementation of a level of detail scheme has also been developed to retrieve the proper information according to the point of view of the user. The client also includes an efficient rendering pipeline that has been developed in order to render these features as billboard markers.

2.1 Related work

This chapter focused in two different but related concepts regarding spatial data: cluttering reduction and progressive transmission. Both have been profoundly researched by several authors throughout the last decades.

Cluttering is an old problem common to most of the visualizations of multi-dimensional data in two dimensions, not only in mapping but also in plots, graphs and networks. This includes multi-variable data visualization, for which Peng, Ward, and Rundensteiner [3] analyses clutter in parallel-coordinate graphs, star glyphs, scatterplots and dimensional stacks; or spatial-temporal data, for which Shrestha, Zhu, and Zhu [4] proposed a new plot technique based on position lines and temporal points. In this topic, the work of Ellis and Dix [5] should be highlighted. They presented the problem of cluttering in a general manner, established a taxonomy to classify the different techniques to tackle this issue in base of aspects which they are focused on, and made selecting an appropri-

ate methodology easier to the reader according to their data and visualization target.

For the case of georeferenced data and mapping visualization, the proposed solutions to the cluttering problem generally rely on sorting and clustering techniques. Sorting solutions define a criteria to order the data from more to less important. This is seen in the work of Pombinho, Carmo, and Afonso [6], which defines a interest function by combining elements of the *degree of interest* function of Furnas [7] and the work of Keim and Kriegel [8]. Special cases of very concentrated points of similar importance are solved in [6] via grid-based aggregation functions. Regarding clustering, several methods have been introduced by Mahe and Broadfoot [9] for cluttering reduction in Google Maps API. Grid and distance based clustering have been reviewed in their work. Other proposals on clustering have been done by Delort [10], which proposed the use of hierarchical distance-based clustering; Lu, Chen, and Cheng [11], that combined distance-based clustering with the *Geotree* data structure to aggregate overlapped data, or Krızek [12], which used clustering with lines and points to better show the risk potential of a certain vessel path. Finally, other studies offer a comparative survey between different methods which aim to reduce cluttering in maps. Two examples are the work of Allison, Treves, and Redhead [13], which compares between a series of cluttering reduction methods based on ten different evaluation criteria; and the work of Korpi and Ahonen-Rainio [14], which presented a survey which focuses on how accurate is the interpretation that users make from data visualization, comparing cluster-grouped data and a heatmap.

On its part, progressive vector data transmission [15] is the common approach to send large volumes of spatial data to viewers in mobile and web environments throughout the Internet. It looks for initially sending the most relevant information and then adding detail on-demand without redundancy, so the volume of transmitted data is kept as low as possible. Progressive transmission techniques have been implemented for meshes (Rusinkiewicz and Levoy [16], Cheng [17], and Kim, Lee, and Kobbelt [18]) and raster data (Dekel and Goldberg [19]), as well as for punctual information. Although for point sets alternatives as neighbourhood operators [20] have been explored, the two common approaches are the conversion from points to TIN meshes for exploiting mesh transmission techniques [21] and the use of *level of detail* (LoD) data structures [22]. Inside LoD structures, binary trees (Gobbetti and Marton [23]) and, above all, octrees (Schnabel and Klein [24], Huang et al. [25], Kammerl et al. [26], and Smith, Petrova, and Schaefer [27]) are the most utilized ones for progressive transmission of point datasets.

Octrees are appreciated for local 3D point data sets, e.g. point clouds, because every new and more detailed level on them are generated in base to a single division on each of the three spatial coordinate axes, resulting in even node loads. However, when the point data sets have information at planetary scales, the height dimension loses relevance with respect to the latitude and longitude and many empty or poorly loaded nodes will be generated. This issue can be solved by using specific data structures for global geo-referenced data,

which are normally called *Geodesic Discrete Global Grid Systems* (G-DGGS) [28, 29] in the literature. The most known and used of the G-DGGS structures are the quadtree, due to its congruent, quadrangular 1-4 refinement, according to Mahdavi-Amiri, Alderson, and Samavati [30]. In this chapter, a variant of the classical quadtree is proposed for the transmission of punctual data.

Finally, the use of already existing software for spatial data transmission has also been considered. The best example is Geoserver [31], an open source, server-side software which allows its users to publish online geospatial datasets. Geoserver implements the open standards *Web Feature Service* (WFS), *Web Map Service* (WMS) and *Web Coverage Service* (WCS) from the *Open Geospatial Consortium*. Its wide compatibility with many kinds of data formats and its easy-to-use web interface have made it a reference on the GIS market.

However, critical disadvantages on its use for large data sets have been found. Requesting sorted feature data for large data sets and bounding sectors is excessively time-consuming, affecting the visualization experience. In addition, Geoserver does not use progressive vector data transmission. Data sent in coarse levels of detail are resent on finer levels, generating redundancy. The extensions for point clustering seem also limited. For instance, the PointStacker extension generates clusters of markers but they can only be displayed as a raster WMS layer. This hinders the implementation of custom actions on markers for user interaction. For these reasons, Geoserver has been discarded.

2.2 System overview

A custom client-server architecture is proposed instead for the transmission and visualization of large punctual datasets. It consists of a mobile app that allows the user to navigate a multiresolution earth model, displaying on real time the markers inside the visible area. The mobile client performs multiple network petitions to fetch marker data, depending on the camera point of view. The server maintains a database containing a representation of the entire dataset, in order to respond to the requests efficiently. Figure 2.2 shows the overall system architecture.

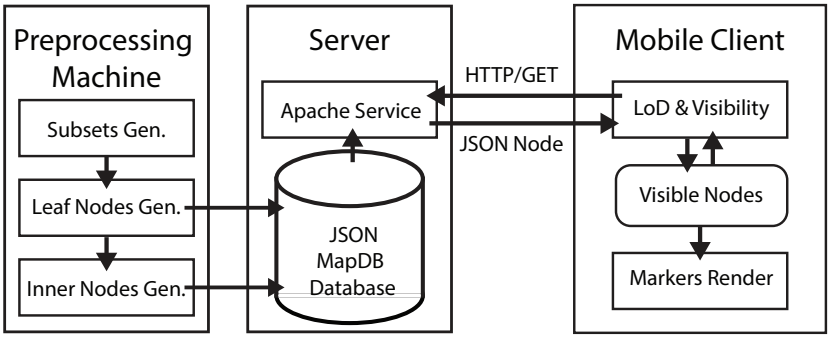


Figure 2.2: General client/server architecture of the proposed system.

The database is generated by running a preprocessing pipeline specifically developed for this purpose. The pipeline takes the point data set and creates a quadtree-like structure for the progressive streaming of the data. Nodes in the quadtree can be downloaded on-demand from a servlet for their visualization.

The mobile app is developed using the *Glob3 Mobile* framework [1, 32, 33, 34]. This open source engine allows the creation of realistic 3D virtual globe natively in web and mobile devices, assuring an efficient use of their graphic capabilities. The web version runs on Javascript and uses WebGL, so it is executable on standard HTML5 browsers. For iOS devices there is a C++ / Objective-C version, and a Java version is available for Android environments, both using OpenGL ES 2.0 as the graphic library. These native versions offer a higher performance on mobile devices than web versions running inside mobile web engines.

The Glob3 Mobile engine enables connecting to different data sources in order to obtain geographic information about a specific bounding sector of the map, including:

- **Raster map servers**, to obtain satellite photos or any other synthetic imagery.
- **Vector data servers**, which provide different geometries for georeferenced features.
- **Elevation servers**, which offer the cartographic data necessary to generate realistic 3D terrain.

This way, all the needed information is downloaded on real time while the user navigates through the 3D scenario, asking for more detailed data in those regions where the viewer approaches.

The vanilla version of the engine offers two different modes for displaying the engine: (i) requesting the whole dataset in a single petition from a network server, and (ii) fetching the data from an offline disk resource. However, none of these solutions is valid for large datasets, due to the scarcity of available memory in most mobile devices. Besides, the raw dataset lacks of any LoD strategy, which makes unfeasible its real-time rendering. Therefore, for the context of this proposal new functionalities have been added to the Glob3 Mobile engine, implementing a LoD strategy to render a high number of markers using online sources. This novel service makes all the needed requests automatically based on a point-of-view LoD scheme.

The following subsections exposes in detail the aspects of the preprocessing pipeline, the streaming service and the mobile viewer application.

2.3 Generation of a LoD based database

Storing data on the client side is not an option when large sets of point markers should be displayed in a mobile device. There are imitations regarding processing speed and memory which makes that option unfeasible. That generates the

need of a server/client architecture, which on its part requires preprocessing of the data for an appropiate transmission.

In the proposed architecture, a pipeline is developed for the generation of different levels of detail from the input data. Their content depends on two possible strategies:

- **Sorting**: every node in any level of detail contains only the N most relevant features.
- **Clustering**: some nodes also could contain clusters that represent several individual features.

The process implemented for such purposes has been split into three stages, performed by standalone Java tools, and its final product is subsequently streamed by the server using a progressive approach.

The threefold preprocessing is depicted in detail in the next subsections.

2.3.1 Splitting data in bounded subsets

The main goal of this stage is to transform a large, unordered input dataset into subsets. These subsets will correspond with a certain area of the globe and have a limitation on the number of features, making the problem easier to solve.

Initially, the dataset is sequentially read from disk, including every single feature inside it in a buffer which can hold a predetermined number of features. When the buffer becomes full, all its features are inserted into the nodes of a quadtree-like structure. If the number of features contained in a node becomes higher than a prefixed maximum size, that node is cleared and children nodes are created for the node. Then, the features of the former node are distributed among its children.

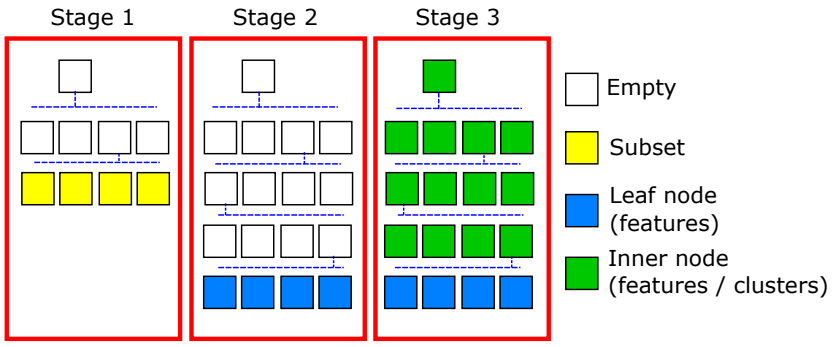


Figure 2.3: Preprocessing stages in the creation of levels of detail.

As splitting criteria for the nodes to be divided, we used the geographical sector related to the node in the following way:

- The root node covers a sector which contains the whole Earth.

- If the ratio between delta latitude and delta longitude is lower than 0.75 or bigger than 1.25, the node is splitted into two nodes of equal area.
- Otherwise, four children are created, splitting by the mid-latitude and the mid-longitude. Thus, each one covers a quarter of the area of its parent.

All changes in the quadtree as a result of an insertion are committed into a database before continuing with the next chunk of data. The process is repeated until all features are read. The result of this stage is a database containing a quadtree which holds the referred subsets. Visual representations of the database and the subsets can be seen in Figures 2.3 and 2.4.

In this stage, the values for *feature buffer size* and *maximum features per subset* parameters should be established in a way that helps avoid the overload due to I/O operations and thus accelerate the process. Experimentation introduced in Section 2.6.1 has been conducted to find the best settings for both parameters.

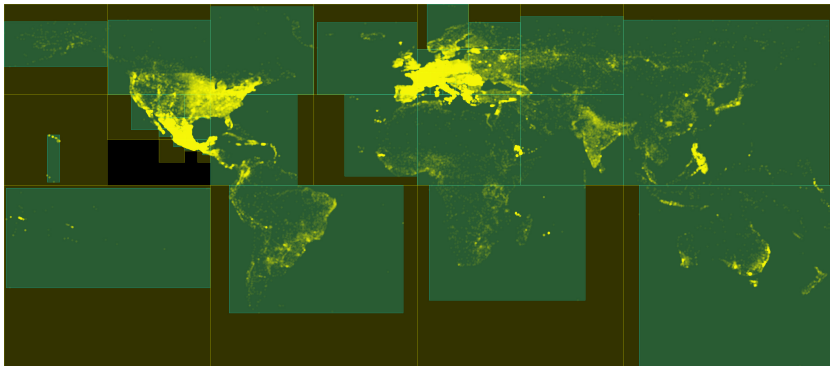


Figure 2.4: Subsets generated at stage 1 for the GeoNames dataset. Subset sector is painted on brown, minimum sector on green.

2.3.2 Creating leaf nodes

In the second stage, the features stored in the previously generated subsets are redistributed among deeper nodes with a lower capacity. This capacity is preassigned experimentally in order to avoid marker cluttering issues. The experimentation and the final settings can be found in Section 2.6.2.

For every subset from the first stage, a recursive algorithm is applied in which we take the features list of the subset node and, in case sorting is been used as strategy, the list is sorted based on the preferred criteria. For each node the number of features is compared with the established maximum. In case of a higher number, the node is split into its children, using the same method that in the first stage, distributing its features. This algorithm is then called recursively for each child node.

Otherwise, a filter based on Haversine distance between features is applied: If the ratio between that distance and the distance between the corner locations of the node bounding box is lower than a threshold, the node is split again. The reasons to use a distance filter and how to adjust the referred threshold are presented in Section 2.6.3.

The resulting tree stores all the features in its leaf nodes, ensuring a maximum number of features per node and a minimum distance between them.

2.3.3 Filling inner nodes in the quadtree

The third stage is devoted to fill the empty inner nodes of the quadtree using a bottom-up strategy. The tree is traversed from the deepest level to the root, looking for inner nodes. For each node found, the minimum bounding sector containing all the features in its subtree is computed. Subsequently, the LoD schemes are applied in base of the strategy of choice:

(a) Unfiltered clustering strategy (b) Filtered clustering strategy

Figure 2.5: Differences in cluster visualization between an unfiltered and a filtered strategy for the islands of Lanzarote (up) and Fuerteventura (down).

- **Sorting**: In case a sorting strategy was selected, the N most important features are extracted and saved in the inner node, with a maximum value of N determined by the Equation 2.1:

$$N_{max} = F\frac{1}{max(C, 2)} \tag{2.1}$$

where F is the total number of features stored in its children and C is the number of children. Note that the most prominent features according to the preferred criteria are selected disregarding their spacial proximity.

- **Clustering**: If a clustering strategy was chosen instead, the minimum sector for the inner node is split into four quadrants. For each quadrant containing more than one feature, a cluster is created, whose center is the average position of the elements. Finally, the cluster set is then checked using the distance-based filter described in the second stage. Clusters that are closer than allowed are joined into a single cluster to generate a clearer display. This can be shown in Figure 2.5.

The resulting feature tree is stored in the final database.

2.4 Server-client communication

The established structure consists in a REST servlet that accepts GET requests with one or two parameters from a client. Our servlet can contain several feature databases, each one called a *layer*.

Therefore, the first mandatory parameter in any request is the selected layer. The second one, which is called *features*, is optional and indicates that the client is only querying for features related to a given node. The request format for a given node and a list of desired properties is shown below these lines:

`http://<serverpath>/<layername>/features?node=<key>&properties=<property_name>|<property_name>`

The result of a request consists in a JSON array containing its features and/or clusters. In case the requested properties exist, they will be also included in the response.

2.5 GIS client implementation

The proposed system is meant to serve a map client application which eventually renders the stored features as geolocated markers for the end-user. In this work, such GIS compatible client has been implemented using Glob3 Mobile as virtual earth framework. The following subsections describe in detail the operation of such system.

2.5.1 Real time LoD during navigation

The rendering process has been implemented as a new subsystem within the Glob3 Mobile engine. The process starts by querying the server for the root node of the quadtree. For each created node, a recursive algorithm will check its visibility and screen-projected area.

Given a certain node, if the node is visible on the screen and its projected area fulfills predetermined conditions, the system queries the server for all its

features or clusters and draw markers for all of them. If the layer contains clustered levels of detail, the system also erases any cluster marker belonging to an ancestor of the visible node.

The LoD criteria determines that a node must be replaced when at least one of the following conditions is met:

- $abs(U_{lat} - L_{lat}) > 80^{\circ}$ or $abs(U_{lon} - L_{lon}) > 80^{\circ}$ where (U_{lat}, U_{lon}) and (L_{lat}, L_{lon}) refer to the upper and lower geographical coordinates covered by the node bounding box.
- The projected area of the node is greater than a fixed proportion of the screen area, measured in pixels. Experiments on how to adjust the screen area percentage were conducted and will be explained in Section 2.6.2.

Applying this process from the root node downwards, a list of visible nodes is obtained whose markers are rendered.

2.5.2 Generating marker icons

Another Glob3 Mobile subsystem is responsible for the generation of the marker icons, called *ImageBuilder*. To do so, it relies on the Canvas API provided by each platform.

ImageBuilder is a hierarchy of classes that implements an interface for providing images asynchronously. Such hierarchy can be seen in Figure 2.6.

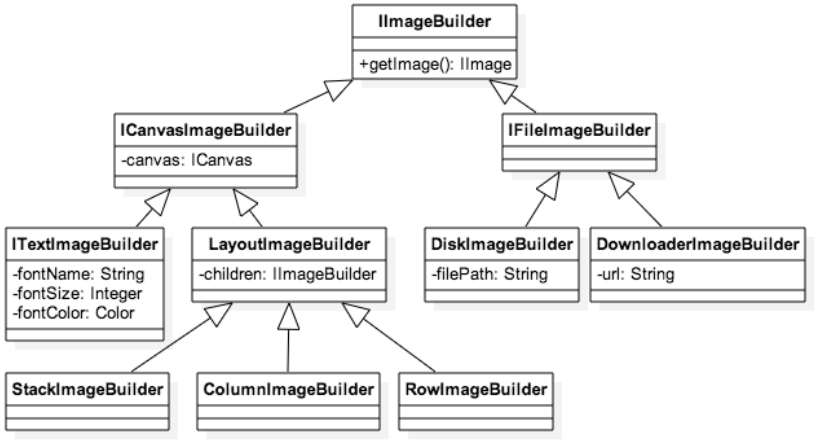


Figure 2.6: ImageBuilder hierarchy UML diagram.

Some markers may also rely on dynamic builders which update their output image when needed.

These *ImageBuilders* can be categorized based on their output. *Basic form builders* generate different geometrical shapes. *Layout builders* take the different outputs of other builders and combine them on a single image. *File builders* are

used for reading images from disk or from online sources. Finally, *Text builders* render text according to given font parameters.

Among them, the text builders are specially useful for markers, considering most of them include some text. An automatic tool also splits the text in multiline layouts, avoiding very elongated markers.

2.5.3 Shader management

Rendering a scene with many detailed and diverse spatial elements, like point markers, map imagery, terrain meshes or 3D models, requires sending enormous amounts of information. This information includes vertices, textures, indices, colors and illumination conditions, all of which contribute to the final rendered image. Moreover, if the scene contains a great number of these assets, careful management of the graphics information is absolutely necessary in order to avoid redundancies.

Both OpenGL ES and WebGL enable establishing explicitly the inputs and processes occurring on the Vertex and Fragment stages [35] of their pipelines, by creating programs in the *GLSL* language. However, most developers know very little about graphic programming. To ease the rendering process of point markers (and any other spatial element), a software design has been made for the Glob3 Mobile framework so the selection of a proper shader program for any given spatial element has been done without user intervention [1]. The management of its associated transactions and data storage is also performed in an automated manner.

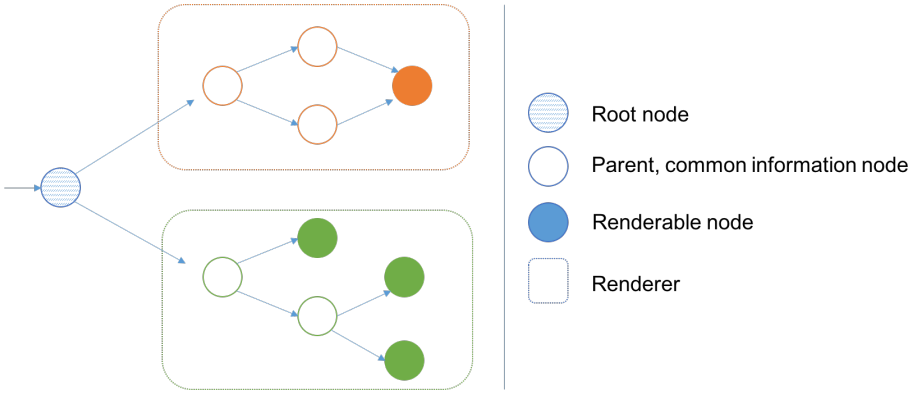


Figure 2.7: A directed acyclic graph to efficiently generate inputs for graphics pipeline from symbology. Nodes could be accessed via multiple paths. Parent nodes contain pipeline information common to many renderable nodes.

The first step is to create a proper representation of the symbology data. To do so, all the necessary rendering data are introduced in a *Directed Acyclic Graph* (DAG), like the one in Figure 2.7. Using this structure, the rendering settings and pipeline information presented as uniforms and attributes, which

are common to multiple renderable objects, are contained in the parent nodes of the DAG. This way, data redundancy is reduced. During the creation of a given frame, the DAG is traversed in depth, and all available information across the path to a leaf node is gathered. After that, the resulting data set is processed, generating all the necessary inputs for the graphics pipeline.

According to these inputs, a GPU program is selected from a shader library integrated in the framework. The system selects the shader that best matches the available rendering data for each DAG path according to the number of inputs. Finally, the re-sending of redundant information during the graphics data transfer is prevented by performing a value checking. Depending on the type and repetition of the symbology, this technique could save up to 80% of the transfers [35] and speedup the process, reusing the data and the program lookup in consecutive frames.

2.5.4 Billboarding

An efficient shader has been developed to render markers. The vertex shader takes the cartesian location of the mark from a single uniform and converts it to screen space coordinates with the *ModelView* matrix transformation, as can be seen in Figure 2.8. The marker corners are then generated through a view independent 2D translation, based on the texture coordinates.

```
void main() {
  //Model and perspective transformation
  gl_Position = uModelview * uBillboardPosition;

  //Screen space placing of billboard corners
  float fx = 2.0 * uTextureExtent.x / uViewPortExtent.x * gl_Position.w;
  float fy = 2.0 * uTextureExtent.y / uViewPortExtent.y * gl_Position.w;
  gl_Position.x += ((aTextureCoord.x - 0.5) - (uBillboardAnchor.x - 0.5)) * fx;
  gl_Position.y -= ((aTextureCoord.y - 0.5) - (uBillboardAnchor.y - 0.5)) * fy;

  //Texture coordinates sent to fragment
  TextureCoordOut = aTextureCoord;
}
```

Figure 2.8: Vertex shader applied to the marker billboards. Screen position of the corners is computed by the GLSL code.

The fragment shader performs a texture fetching operation using prefixed texture coordinates. This texture might be common to a large number of markers, therefore, the system sort the markers according to the texture they use. This way, a minimum number of OpenGL state changes is required.

The billboard generated using this technique has a single value of depth for all its area. For markers that are close to the ground or any other 3D model, using depth test normally would derive on *Z-Fighting* artifacts or partial coverage of the markers. The depth test should be disabled and the rendering of markers performed after the rendering of the 3D scene.

However, in some use cases it is important to determine whether the terrain occludes the markers. The calculation of the final visibility compares the depth

of the center of the mark with the actual depth of the frame, which on mobile platforms implies using depth rendering or ray shape intersection techniques.

2.6 Tests and practical results

In this section, several experiments are described to determine the optimum visualization parameters. To conduct these tests, the following machines were used:

- Processing machine: CPU: Intel i5 dual-core, 2.6Mhz, RAM: 8 GB LPDDR3, 1 TB HDD, OS: Mac OS.
- Mobile device: Motorola MotoG3, OS: Android 5.1.1. Map screen area: 720x720 pixels.

The chosen input dataset was *allCountries*, a dataset from Geonames.org which is freely available under a Creative Commons license. This database contains 10510732 records concerning places around the world, and both the dataset and its detailed format can be found at [36]. Apart from the position, each feature on this subset contains a related population, which is the selected parameter for the sorting strategy, and a name label, which is used in order to generate the point marker.

2.6.1 Subset size

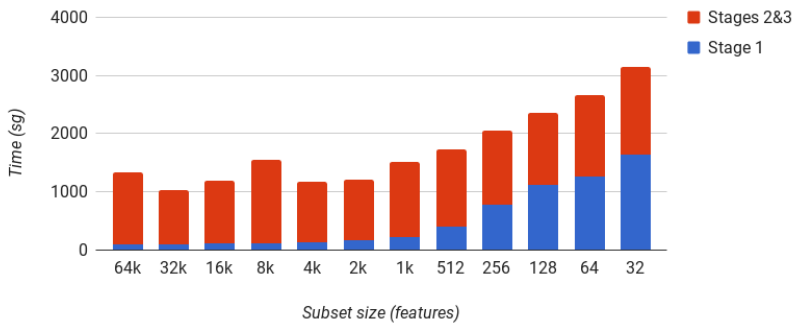


Figure 2.9: Execution times of preprocessing stages using different buffer sizes.

A first experiment was designed to find the proper subset sizes for the first stage which allows us to minimize overloads due to I/O operations and accelerate the process. The execution time for preprocessing has been measured using different buffer sizes between 32 and 65, 536 features. In order to perform mono-objective optimization for the subset size parameter, the other parameters have been fixed as clustering strategy and a node size of 32 in all the experiments for the second and third stages.

It should be noted that I/O operations could be found in all stages of pre-processing. Execution times are measured for all the process and not only for the Stage 1. The results are shown in Figure 2.9.

A value of 32k for the subset size has proven to be effective in order to accelerate the execution in all stages of the process. Therefore, it is selected as the final value for the given hardware configuration. Lower values increment the number of I/O operations for the three stages.

2.6.2 Node settings

There are two parameters in the final tree nodes that have a key influence on the number of features which will be shown on the screen during a map navigation. These are the maximum number of features per node and the split criteria applied to every node.

In order to adjust them, experiments have been designed in which trees have been generated where the maximum number of features per node varies between 4 and 20. At the same time, the projected area of each node was adjusted to fit from a 5% up to 50% of the screen area.

	Starting position	Final Position	Pitch
Flight 1	latitude: 5.35825 º longitude: -101.792515º height: 5764420 m.	latitude: 41.730985º longitude: -87.668136 height: 16029 m.	-45º
Flight 2	latitude: 48.06959º longitude: 7.204285º height: 9442414 m.	lat: 40.433391º lon: -3.618721º hgt: 32166 m.	-90º

Table 2.1: Settings of the Glob3 Mobile flight tests.

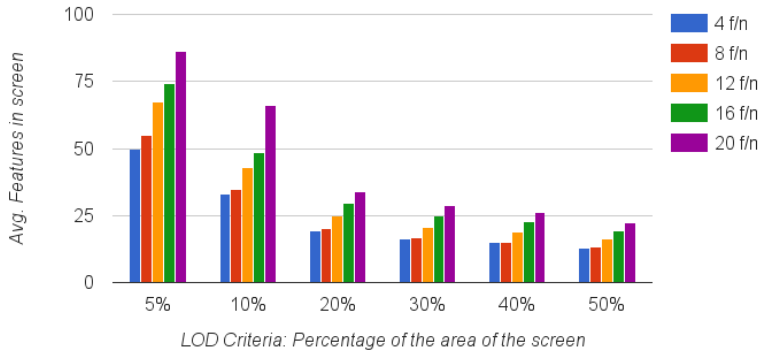


Figure 2.10: Features in screen obtained for Flight 1 after varying node capacities.

Two flight simulations have been made using all possible combinations of the

referred values. Both had a duration of 60 seconds, and their camera pitch and starting/ending positions can be seen in Table 2.1. The first flight set the view direction slightly towards the horizon line, in order to obtain 3D perspective views of the terrain. The second flight is similar to a 2D zooming, with the camera view direction always perpendicular to the terrain.

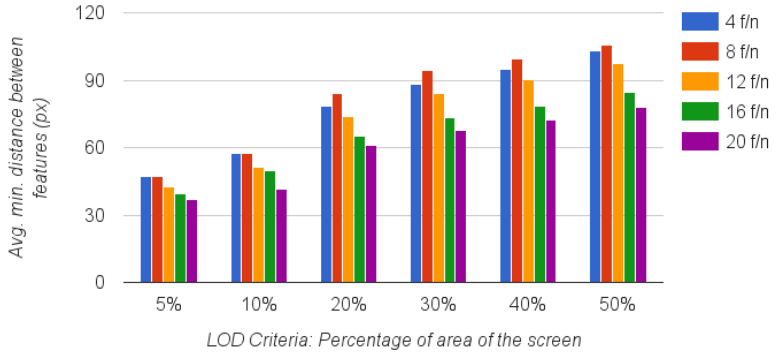


Figure 2.11: Average of min. distance between markers for Flight 1 after varying node capacities.

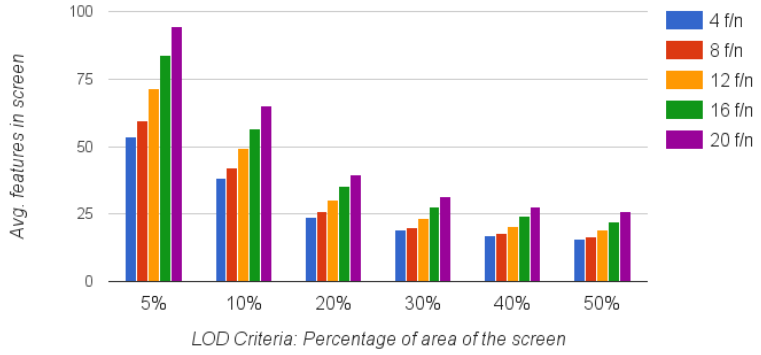


Figure 2.12: Features in screen obtained for Flight 2 after varying node capacities.

For each frame during the test flights, we saved data regarding the number of markers on screen and the average of the minimum distance in pixels between them. The results can be seen in Figures 2.10, 2.11, 2.12 and 2.13.

Our testing device has a screen area of $518,400$ pixels. In this dataset, the associated image for every marker is an elongated text label, that has an average area of 4,065 pixels. Thus, it would be possible to draw 127 non-occluding markers on this screen. However, a map full of markers is really hard to read. Several tests on user experience showed that the map usage is more comfortable

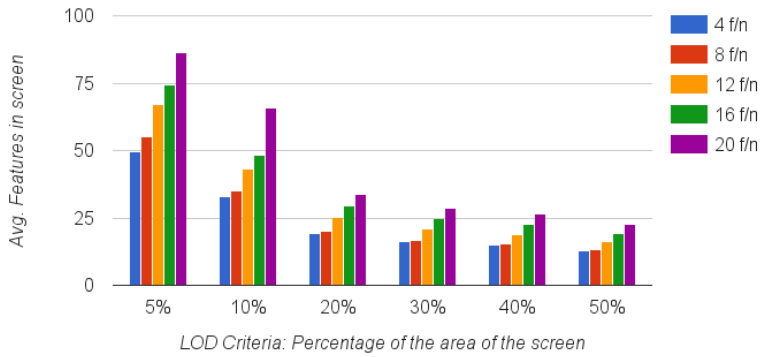
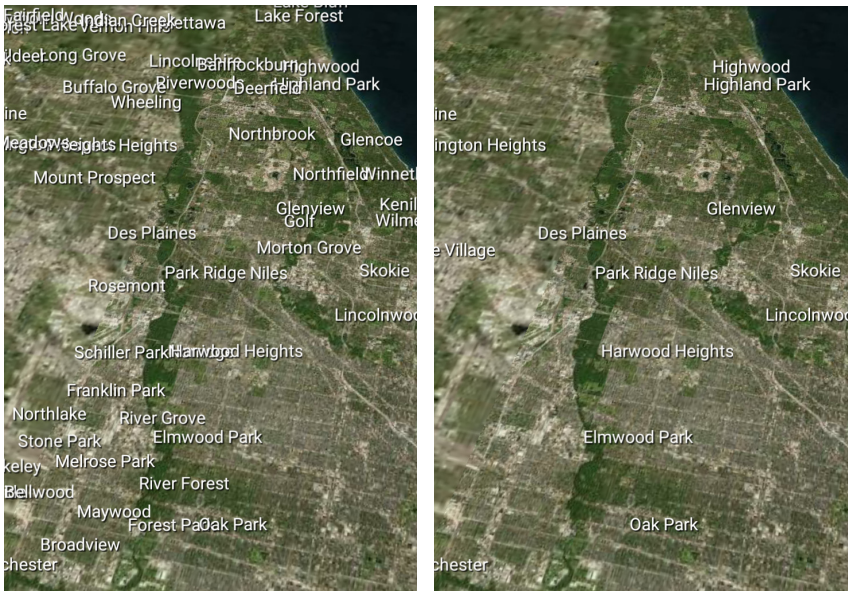


Figure 2.13: Average of min. distance between markers for Flight 2 after varying node capacities.

when around a 20% of the screen area is covered by markers, i. e., 25 markers using our mobile device.



(a) Proj. size: 10%, capacity: 12 (b) Proj. size: 30%, capacity: 8

Figure 2.14: Screenshots taken during the comparative for Flight 1 and varying parameters.

As the experiment shows, a LoD criteria that limits the projected size to 30% of the screen, combined with nodes with a maximum capacity of 8 features, offer

an average number of markers close to the recommended while maximizing the average of minimum distance between them. In Figure 2.14 it is shown how the information displayed for a certain area varies with different combinations of the parameters.

2.6.3 Distance-based filter

Limiting the number of visible markers does not mean that they are well distributed along the screen. Cluttering issues can be found when all features are concentrated in a certain area of its node. In order to avoid such issues, a distance-based filter were introduced in the pipeline.

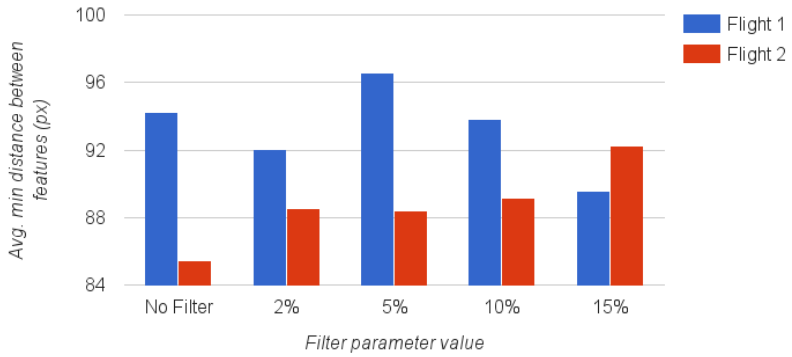


Figure 2.15: Comparison of minimum distances using different filtering parameters.

This filter has been adjusted by conducting tests over the same flight simulations indicated in Table 2.1. The previously selected parameter configurations are combined with the distance-based filter, varying its ratio from 2% up to 15%. The comparative results of these simulations can be appreciated in Figure 2.15.

The results have shown that the use of a filter improves the distribution of features on screen during the second flight for all tested parameter values, as seen in Figure 2.16. However, for the first flight the distribution of features was only enhanced with a ratio of 5%. Therefore, that value was chosen to be used in the filter due to its improvements in both flights.

2.7 Conclusions

The chapter explores the possibilities of the generation of levels of detail for extensive datasets of geolocated features, their performance on mobile devices, and their integration in a virtual globe. The creation of such levels is based on

(a) Unfiltered (b) Filtered

Figure 2.16: Differences between non filtered and filtered simulations. A 5% filter solves a cluttering issue between two of the markers in the area.

the well-known strategies of clustering and sorting, being the latter designed for progressive streaming.

A novel algorithm has been proposed in which the problem is split into three stages, producing a final tree structure. The features contained in this structure are associated to nodes that define the behavior of the view-based LoD algorithm present on the client. The implemented system allows mobile devices, such as smartphones and tablets, to request on run-time different degrees of detail of the dataset, providing a pleasant user experience.

The conducted tests demonstrated that the proposed algorithm is suitable for the fine-tuning of its defining parameters. Moreover, it is shown how these parameters impact on the overall preprocessing performance and the final visualization experience.

Chapter 3

Visualization of spatial underground data in multimodal applications

In the previous chapter, the importance of pre-processing the geo-referenced data and having an adequate streaming scheme which maintains the key information without cluttering the scene has been exposed. However, displaying data should not be done just for the sake of displaying. Data only becomes information when it is processed in an understandable way which allows the final user not only to interact but to make better decisions based on it. And this fact has a strong impact on important day-to-day tasks, e.g. managing the different infrastructures of a city, in particular utility networks for distribution of fresh or waste water, oil, gas or electricity, communication networks or even manholes.

Precise information and knowledge about these infrastructures is required for efficient urban planning and management [37, 38]. Traditionally, it has been managed using paper-based documents and maps. However, with the spread of information and communication technologies, digitization and visualization techniques are used in some cities [39, 40]. Appropriate visualization is of crucial use for utility networks planning, management and maintenance, essentially supporting the entire life cycle of the infrastructure components. These infrastructures can be graphically represented using different features, as line features to represent networks segments, points features to illustrate connections, substations or valves with spatial and non-spatial attributes. Visualizing these different types of features is a challenging process. Most recent research on spatial visualizations has focused on the above surface context, and traditional GIS were typically developed as 2D desktop GIS applications, which ignored the representation of underground features.

In the last two decades, a shift from traditional 2D desktop GIS to 3D and 4D desktop and mobile GIS applications has emerged where 3D/4D visualization

is used because it allows for a more effective representation of utility network features [41, 42]. Different approaches ranging from more traditional vertical profile visualizations to immersive visualization such as *Augmented Reality* (AR) and *Virtual Reality* (VR) have been developed. However, these applications did not find widespread usage until recently; mainly due to the lack of accessible and limited hardware options as well as suitable *Software Development Kits* (SDK) [43]. In 2017, the development and introduction of more performant consumer mobile hardware and native software support from both Google, with its ARCore SDK[1], and Apple, with its own ARKit SDK[2], allowed the creation of performant applications and optimal support of cameras and sensors already built-in in the mobile hardware [44].

The research and the development of such applications is still mainly focused on hardware and software implementation aspects and less on the visualization aspects [45, 46, 47, 48]. In this chapter, the main contribution is a research on optimal visualization methods for underground data, which has been done thanks to a partnership between the *University of Las Palmas de Gran Canaria* and the *European Institute for Energy Research* (EIFER) and has been already introduced in [49]. Three distinct strategies have been tested: the first is based on the use of transparencies to convey a sense of depth, the second relies on the generation of *ditch* meshes around the utility object and the third is a world-space deformation of the elevation model that exposes the underground elements. These strategies have been implemented in an immersive multimodal application, called *MultiVis*, for handheld iOS and Android devices. Furthermore, a comparative user experience analysis of different techniques aimed to the visualization of utility networks and other underground facilities are performed and evaluated. It includes a set of user evaluations for different parameters of these techniques, which gives us an insight on how the proposed methods affect the experience and usability for technical and non-technical users.

3.1 Related work

In the last twenty years, several studies have proposed solutions to accurately visualize subsurface data in VR or AR applications. A first AR prototype is introduced by Roberts et al. [50]. It relies on GPS sensors and visual tracking to overlay an underground utility network over a real scene. Then, the scene along with the overlay is shown in a *Head-Mounted Display* (HMD) device. Bane and Hollerer [51] proposed a *Tunnel Tool* visualization: data related to hidden and occluded parts of the scene are rendered inside a frustum, generating a tunnel effect in the final image. Avery, Sandor, and Thomas [52] applied an *edge overlay* effect, in which the outlines of visually distinct features on occluding surfaces are preserved to provide depth cues to achieve an *X-Ray vision* effect.

Around 2009, two different lines of research on how to visually represent underground data surged. The first one looks for estimating and showing the

[1] https://developers.google.com/ar/
[2] https://developer.apple.com/arkit/

positioning error of an underground network model. Such a technique is desirable for field technicians in order to avoid damaging hidden subsurface elements during maintenance tasks. On this topic, Su et al. [53] investigated the use of an uncertainty region around the pipe geometry. That region is visualized in the scene as a semi-opaque polygon. Li, Cai, and Kamat [54] empirically derived a proper size for that uncertainty region by comparing utility network plans with *Real-time kinematic global positioning systems* (RTK-GPS) and GPR sensor measurements made in the working place. Zhang et al. [55] compared the precision and perceived depth appearance obtained after placing the subsurface data in an AR app with computer vision matching techniques and sensor-based techniques. Finally, Scholtenhuis et al. [56] showed a fuzzy 3D-model of the utilities in an AR application for smart glasses and tablets. Models containing cylindrical halos were used to represent minimum, mean and maximum placement error margin.

The second strategy looks for improving how underground data are represented in an AR application. In this regard, Schall et al. [57] proposed an *excavation tool*, which simulates a hole on the ground and makes pipelines visible. Their work was later extended [37] so that the application could assist with the maintenance tasks of underground utility networks. They also compared their excavation tool with trench-like and shadow-like representations. Chen et al. [58] combined the *X-Ray vision* approach with aperture focus and context concepts to extract the depth order and mobile elements of a scenario. This information enables the generation of blending masks to draw the occluded objects. Finally, Zollmann et al. [59] compared the use of alpha-blending, edge ghosting and image-based ghosting techniques for representing subsurface objects in images.

Most underground-related applications are thought to be useful only in the context of AR and designed for geological purposes (Lee, Suh, and Park [60]) or management of power and water underground utilities [54, 56, 37, 61]. However, multimodal applications, for which the work of Santana et al. [62] is a good example, are applications that can feature multiple environments, including virtual globes and VR or AR viewers, to display geo-referenced data. The work of Santana et al. [62] shows that integrating underground visualization in this type of applications is still an open problem. In this chapter, a contribution to that field is done by introducing a visualization scheme of subsurface objects that remains useful for all of the aforementioned integrated view modes. The proposal is then illustrated with a use case in which appropriate underground data and AR, VR and virtual globe technologies are applied.

3.2 About the geo-referenced underground data sample

The visualization of subsurface infrastructures is a topic which fits in the areas of urban planning and Smart Cities. It must be taken into account that in those

contexts, underground data is not only analyzed and displayed in screen, but it is commonly integrated as a part of larger urban models that include buildings, pump stations or reservoirs (Delmastro, Lavagno, and Schranz [63], Li et al. [64]. This implies that the data sample for this study should be generated according to industry standards.

Several standards for representing data related to utilities and underground infrastructures have been developed, such as the INSPIRE Generic Network Model ([65]), the ISO standard *Industry Foundation Classes* (IFC) ([66]) or the ESRI Geometric Network model ([67]). However, most of them lack of explicit control over all of the necessary fundamental aspects required to fully model the physical, functional and semantic properties of arbitrary utility networks in a three-dimensional context (Kutzner and Kolbe [68]). Therefore, using one of these data models as the basis for a subsurface feature visualization app may therefore lead to conceptual and technical limitations. This risk increases as the application escalates.

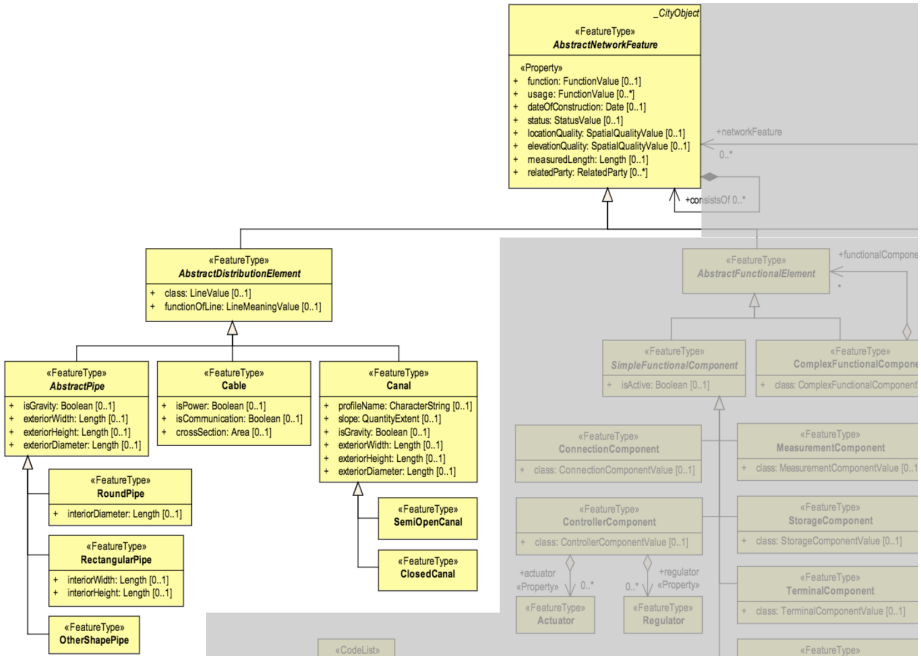


Figure 3.1: A section of the UML diagram of the Utility Network ADE Network component from the repository of Kutzner [69]. The highlighted area depicts the elements used for rendering and visualization.

CityGML is another standard for the representation and exchange of semantic 3D city and landscape models which is open and promoted by the *Open Geospatial Consortium* (OGC). Its data model is based on the ISO 19100 standards family and it is implemented as an application schema for the *Geography Markup Language* (GML) [70], which is also from the OGC. Researchers, devel-

opers and users of the CityGML standard started creating *Application Domain Extensions* (ADE), which enable the extension of the CityGML standard for the purpose of modelling urban objects that belong to a specific theme. The *Utility Network ADE* is one of those thematic extensions. It offers new urban objects and properties so utility networks and infrastructures can be modelled with ease. While still in development, it has been shown to be mature enough to model the constituent features of a real urban multi-network system, as well as functional and connective relationships within and between networks (Duijn [71] , Boates, Agugiaro, and Nichersu [72], Boates [73] , Den Duijn, Agugiaro, and Zlatanova [74]).

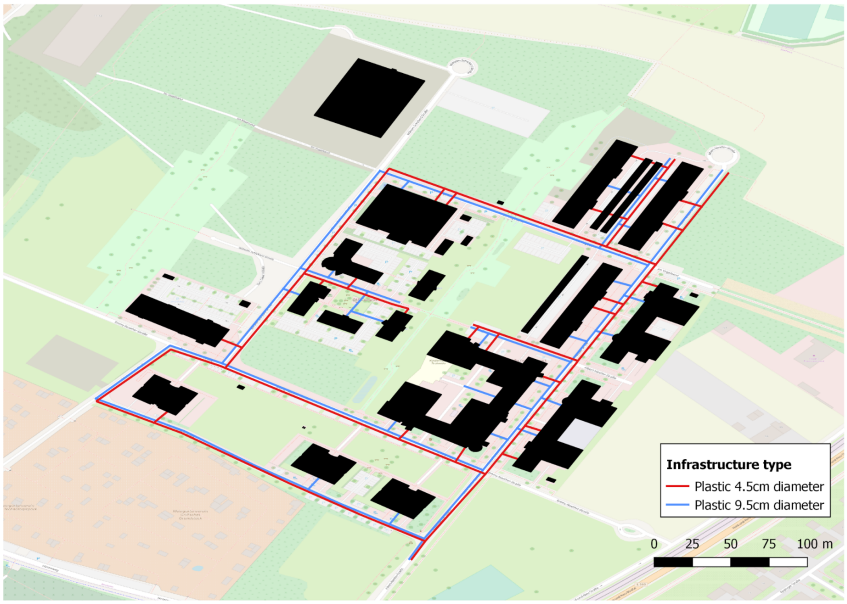


Figure 3.2: Spatial extent of the study area with data curated from OpenStreetMap and a utility network in the CityGML Utility Network ADE standard.

The Utility Network ADE data model for CityGML uses inheritance and hierarchical relationships to make common properties shared between features, while each feature defines its own unique properties. For example, a *RoundPipe* element and a *RectangularPipe* element are both children of the *AbstractPipe* element, and so they share properties implemented by the *AbstractPipe* element, such as the material type, the intended function or the year of construction. However, the *RoundPipe* feature implements a diameter property, whereas the *RectangularPipe* feature implements properties relative to width and height (Figure 3.1). These inheritance-based relationships allow the storage of individual features in a manner which lends itself well to a standardisation

of visualization processes.

For the work described in this chapter, the Utility Network ADE was used to model a sample multi-network comprised of different kinds of pipes placed in a district from the German city of Karlsruhe called *Technologiepark*. For modelling purposes, the *RoundPipe* model is used due to it being the most vastly used on civil infrastructures. Figure 3.2 shows the utility network data consisting of utility pipes of different diameters, depths and types.

Using this data model resulted in a simple data sample for rendering the properties of different kinds of features in different networks, while ensuring that subsequent progresses on rendering abilities can be tested simply by expanding the data sample. Expanding this data sample is, as discussed above, straight forward and any use case of a visualization application can likely be addressed. Furthermore, the direct link to the CityGML core data model supports future integration with smart city applications.

3.3 The MultiVis mobile application

Figure 3.3: User interface layout of *MultiVis* for iOS and the three implemented visualization modes.

The visualization techniques for underground data that will be introduced in this chapter should be tested on a multimodal application. To do so, there are two possible options to choose: to create such an application from scratch or modifying an already existent application. From those, it was decided to adapt the *MultiVis* application, which is presented in detail in the works of Santana et al. [62] and Wendel, Santana, and Simons [75]. *MultiVis* focuses on a holistic approach that implements a seamless transition between a traditional virtual map view to VR and AR modes in a single mobile application. This is particularly interesting to experts and decision makers as it provides them with

means to explore results on-site through AR or VR. These two visualization techniques have been combined with traditional maps for a better overview and strategic planning capabilities (Wendel, Santana, and Simons [75]). With respect to the content, it loads a geo-referenced city model which can contain buildings and points of interest over its geographic area, for which satellital images and *digital elevation models* (DEM) are also loaded.

MultiVis has been implemented using the *Glob3 Mobile* framework ([62], [76]). This engine is chosen due to it being highly configurable in terms of user navigation and level of detail (LoD) strategies. Moreover, Glob3 Mobile is open-source, which makes easier the work of modifying some aspects of *MultiVis* to introduce novel underground visualization techniques. The Glob3 Mobile API provides native performance on its three target platforms (iOS, Android, HTML5). 3D graphics are supported by the Khronos Group APIs *OpenGL ES 2.0*, on portable devices, and *WebGL* , the web counterpart of OpenGL, on the HTML5 version. Due to the multi-platform nature of the Glob3 Mobile, API portability to Android or HTML5 is possible. Figure 3.3 shows the different visualization modes that were adopted from Santana et al. [62] and ported to Android.

3.4 Geometrical modelling and rendering

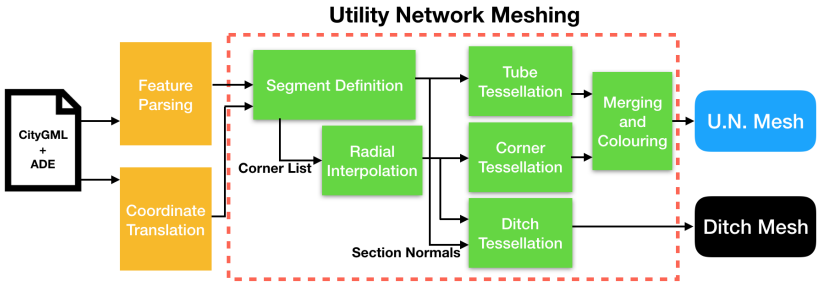


Figure 3.4: Utility network mesh generation process. RoundPipes are extracted from the CityGML model (orange) for meshing (green). The process generates a list of straight segments and rounded corners that are tessellated independently. The final product is a set of tubular shapes signalizing the pipes (Utility Network Mesh) and surrounding black terrain trenches (Ditch Mesh).

This section focused firstly in the tessellation process of the pipe network data, which will give two meshes as a result: one representing the actual round pipes belonging to the network, and another one which represent *ditches*. Those ditches, which are introduced in detail the following subsections, envelope the pipes and are thought to give extra depth cues to the user. Then, the multi-pass rendering process which stacks both meshes in the image is also explained. Figure 3.4 describes the different states and sub-stages of this process.

3.4.1 Meshing pipes from utility network models

Commonly, underground round pipes are characterized as a linked chain of straight segments and a radius, r. This geometry is presented in a CityGML LoD2 model accompanied by metadata regarding the inner and outer materials of the pipes. As a precondition to the tessellation process, the model must be previously cleaned of consecutive segments in the same direction, which are merged beforehand. Each one of these round pipes is visually represented by a tubular mesh of its outer surface.

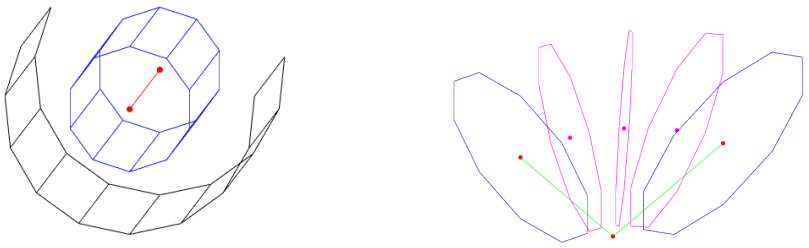


Figure 3.5: Generation of cylinders from a segment (left, blue) and rounded corners between two cylinders (right, magenta).

The first step consists in refining the corners of the pipe polyline, which normally shows sharp corners instead of smooth turns. To do so, each segment in the set is shortened at both ends by the same length, r, which is the radius of the cylinder. A cross-section of the tube is generated at each segment end by rotating a point at a distance r in the direction of the normal of the segment. The point is then rotated n times $(360/n)^\circ$, with n being the desired number of vertices per segment, as it could be seen in Figure 3.5 at the left.

During the smoothing of each bend, the rotation needed to bring each end of a segment to the start of the next is computed. The pivot point of such rotation is computed as the intersection of both segment normals. Once the rotation between segment ends is computed, m new intermediate covers are generated by rotating $\gamma/(m-1)$ (being γ the angle of the joint) the end cross-section of the first segment. This is best shown in Figure 3.5 , right. Finally, the tubular surface is subsequently generated by connecting the vertices of all the computed cross-sections via triangles. The final mesh is stored as a triangle strip to improve the rendering performance.

3.4.2 Strategies for visualization of underground utilities

The meshed model generated in the previous section has been rendered following four different strategies, so they can be compared in terms of user experience. Those strategies, which are summarized in Figure 3.6, are variable static, alpha-blending, ditches and an excavation tool.

(a) Static. (b) α-blending. (c) Ditches. (d) Excavation.

Figure 3.6: Tested methods for visualization of underground utilities.

Variable α-blending

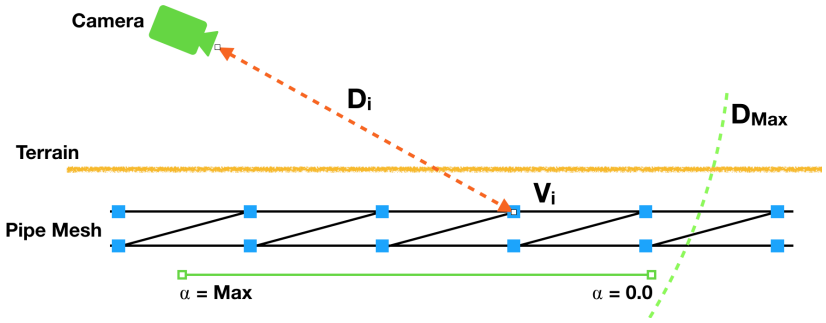


Figure 3.7: Pipe transparency is a function from the distance to the camera to the pipe mesh, computed at the vertices (V_i). D_{Max} is the maximum distance at which the transparency value (α) should be 0.

The use of transparencies to indicate that an object is below the surface is widely extended upon from previous works. A simple manner of this effect is to apply a single value of transparency (α) to the whole object, regardless of how deep or how far from the user viewpoint it is. Zollmann et al. [59] considered the option of applying different transparency values for underground elements, but did not implement the technique, focusing on ghostings instead. Moreover, which function should be used to assign each in order to convey the sensation of depth is another issue to be considered.

In this regard, making the function dependent on the distance to the viewer, D_i, is proposed. It is applied to each pipe vertex closer than a maximum

allowed distance D_{max}, as depicted in Figure 3.7. Considering D_{max}, D_i can be remapped into the range $[0, 1]$. Let the result of the remapping be d_i. This allows for α values to be assigned according to Expression 3.1:

$$\alpha_i = 0.5 \cdot (1 - f(d_i)) \tag{3.1}$$

in which $f(d_i)$ can be any function which returns a value in the range $[0, 1]$ for any given d_i. Since the choice of $f(d_i)$ will affect the final visualization, several alternatives have been explored. Table 3.1 summarizes the different alternatives and their corresponding equations. Figure 3.8 shows the variation of these equation for ranges of d_i, between 0 and 1.

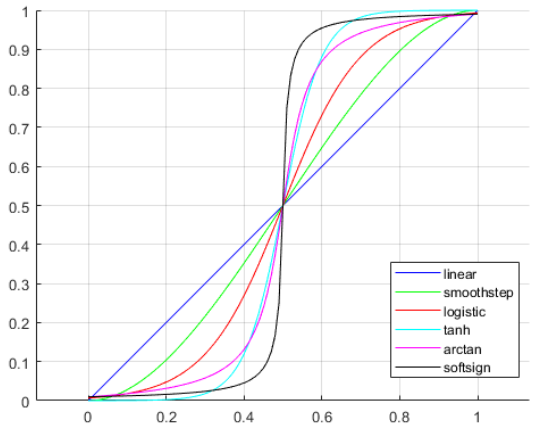


Figure 3.8: Superposition of the different tested expressions, with values of d_i mapped into the range $[0, 1]$.

$f(d_i)$	**Expression**
Fixed	0
Linear	d_i
Smoothstep [77]	$3d_i^2 - 2d_i^3$
Logistic [78]	$1/(1 + e^{-s_i})$, where $s_i = 10d_i - 5$
tanh	$0.5 + 0.5 \cdot tanh(s_i)$, where $s_i = 10d_i - 5$
arctan	$\frac{1}{3} \cdot (arctan(s_i + 1.5)$, where $s_i = 20d_i - 10$
Softsign [79]	$0.5 \cdot (\frac{s_i}{1+\lvert s_i \rvert} + 1)$, where $s_i = 100d_i - 50$

Table 3.1: Different distance-based α subfunctions.

The transparency of each rendered fragment of the model is computed every frame. However, in order to improve efficiency, the results are stored in graphics memory, so calculations are only done on changes of the virtual camera position.

Once all values are calculated, the color information per vertex, including α, is sent to a shader engine to apply the final appearance of each feature.

Terrain ditches

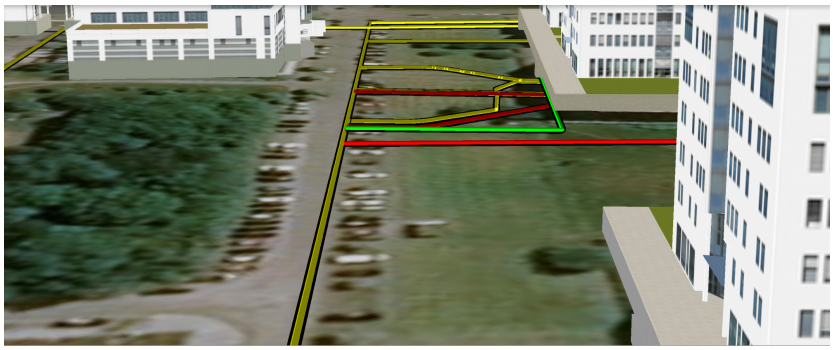


Figure 3.9: Pipes and their ditches in a 3D scenario simulating a street.

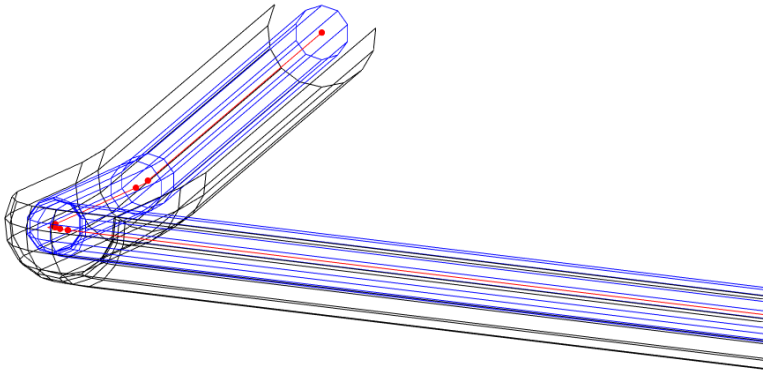


Figure 3.10: Generation of a ditch model (black) surrounding the pipe (blue) generated for an underground network object (red).

As an alternative to using transparencies, the creation of a second geometry model to add an underground context is proposed. This model, called *ditch*, is semi-cylindrical and surrounds the lower half of the pipe. The ditch is rendered in the scenario with a flat and dark color, so the pipe seems to be placed inside and below the ground plane, as seen in Figure 3.9.

The generation of the ditch model is analogous to that of the pipe, creating semi-cylinders and rounded corners based on the line segments, as well as a

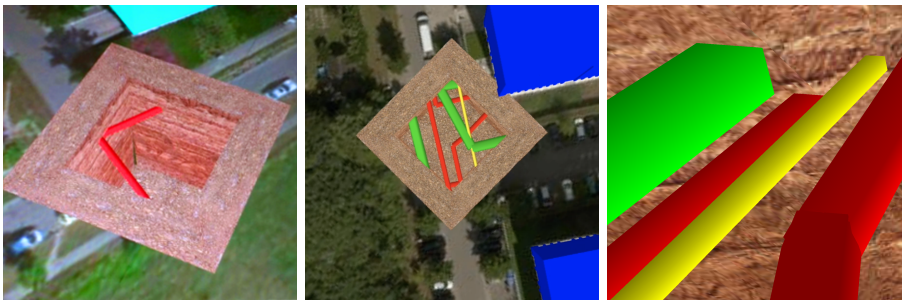
radius. This allows the creation of both meshes following a parallel process, as it is appreciated in Figure 3.10, saving computational resources.

Excavation tool

The last rendering strategy considered is to extend the proposal of an excavator tool, which was done for AR by Schall et al. [57], so it can be used not only in AR-based applications, but also in applications based on a virtual globe. The placement of the hole is dependent on user interface events.

For maps, the hole generation is triggered when the user long-presses a screen pixel, which is then counter-projected to the scenario. If the outgoing ray intersects a position of the terrain, it is used as pivot for the hole generation. In a VR environment, the hole is generated at a fixed distance in front of the user. The hole is then stationary, while the user can walk, take a look and inspect the revealed underground elements.

To take full advantage of the three-dimensional nature of this kind of scene, a rectangular area of the DEM is modified in our implementation, so its height is N meters deeper than the shallower point in that area. In order to preserve the continuity of the terrain, *skirts* [80] are generated whenever necessary. Finally, a textured mesh is created in the edges of the hole region to simulate the walls of the hole. This can be seen in Figure 3.11.



(a) 3D view. (b) 2D view. (c) Inside the hole.

Figure 3.11: Underground network visualization using the implemented excavator tool extension [37] for virtual globes.

Although depth cues are improved for map and virtual reality by using this technique, some issues must be addressed before using it in a multimodal application. For instance, a *level of detail* (LoD) scheme becomes necessary. On one hand, for large scale rendering, the LoD may be coarse enough for the hole visualization to be imperceptible. A possible and straightforward strategy is to make the region affected by the hole larger as the level of detail becomes coarser. At the same time, a LoD strategy should also be implemented to generate new hole elevation models for these cases. Another possibility is to enable user interaction on the *excavation*, allowing it to move and change its

size according to gestures in the screen.

3.4.3 Rendering of underground elements

The MultiVis mobile application, for which the integration of underground visualization is being discussed in this chapter, has been developed using the Glob3Mobile framework. This makes possible to use the automatic shader selection and the DAG graph introduced for rendering point markers in Chapter 2 in order to render the underground symbols and elements.

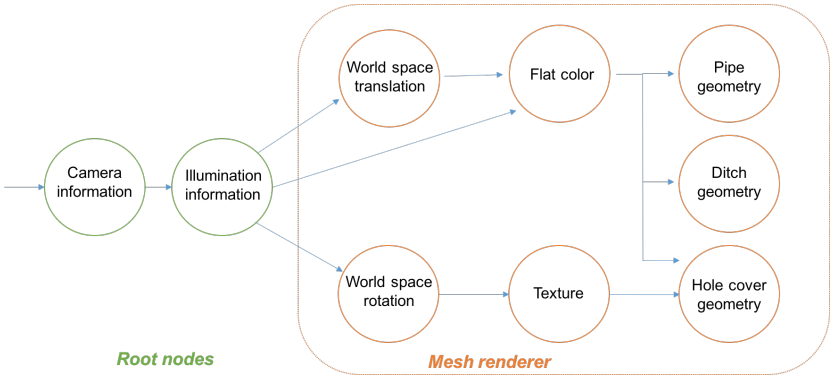


Figure 3.12: Direct acyclic graph for an example where 3 geometry symbols (pipe, ditch and cover) require 7 symbol instances.

Each one of the different initial branches in the DAG directly corresponds to a certain manager component, a *renderer* [81], which takes care of a specific kind of symbology. As an example, the *PlanetRenderer* is the branch with information about terrain and map imagery, the *MarkRenderer* processes 2D markers, the *MeshRenderer* deals with vertices, indexes, textures and other things needed to generate meshes like the underground pipes, etc. An example on how it will work for three geometry symbols representing a pipe, a ditch and a hole wall is introduced in Figure 3.12. During the rendering, *PlanetRenderer*, the module in charge of terrain model generation and rendering, is always processed first. The content included in the rest of the renderers are then processed in a user-defined order.

A limitation of this scheme is exposed when underground elements are included in the scene. If a depth test is performed, underground elements will not be shown in screen. If depth test is disabled for those particular elements, they will be rendered on top of every above-ground object, losing all depth cues and potentially confusing the users. This is the starting point for the alpha-blending strategy. For ditches, a multi-pass rendering scheme [82], also called layered rendering, was also implemented on the Glob3 Mobile engine.

To achieve the desired effect of terrain transparency, the abovementioned multi-pass rendering strategy renders different layers of the scenario with inde-

pendent depth buffers. Renderers containing the underground symbology are set as second-pass renderers. In case there are over-surface elements that could eventually occlude the underground network (e.g: building models), their renderers should also be marked as second-pass renderers.

At the time of rendering, the planet and first-pass content are drawn first. After that, the *Depth Buffer* is cleared. This enables the use of the depth test with underground information. Finally, the elements contained in second-pass renderers are drawn over the same *FrameBuffer* in the pre-established order.

3.5 Experimentation and user survey

To compare the behaviour of the different proposed strategies and determine the one which results in the most convenient visualization, a survey has been conducted with users applying a methodology similar to the one introduced in the work of Mirauda et al. [61]. The users are required to navigate through the *MultiVis* application in a 3D scenario, which includes representations of objects over the surface (buildings, trees, sensors, etc) and objects below the surface. Underground objects will be represented using one of the proposed strategies.

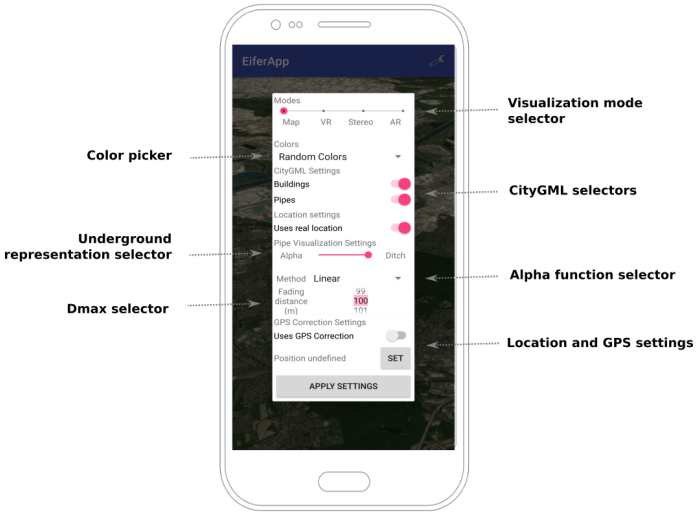


Figure 3.13: User interface layout for the survey in the Android platform. Underground-related selectors are added to the control panel.

The survey participants are asked whether the pipe network looks like it is placed below the surface or not, for all the different $f(d_i)$ functions tested for the α-blending technique, as well as for the static, ditch and excavation strategies. Responses are encoded so that their answers fit into a 5-point Likert scale. A value of 1 indicates that the participant felt like the feature does not look like it is underground at all. A value of 5 indicates that the participant

felt like the feature was certainly underground. Additionally, users were invited to experiment with different D_{max} values for the function of their choice and indicate the value which, in their opinion, made the visualization more useful. Extra controls have been included in the *MultiVis* interface for the survey, and they are shown in Figure 3.13.

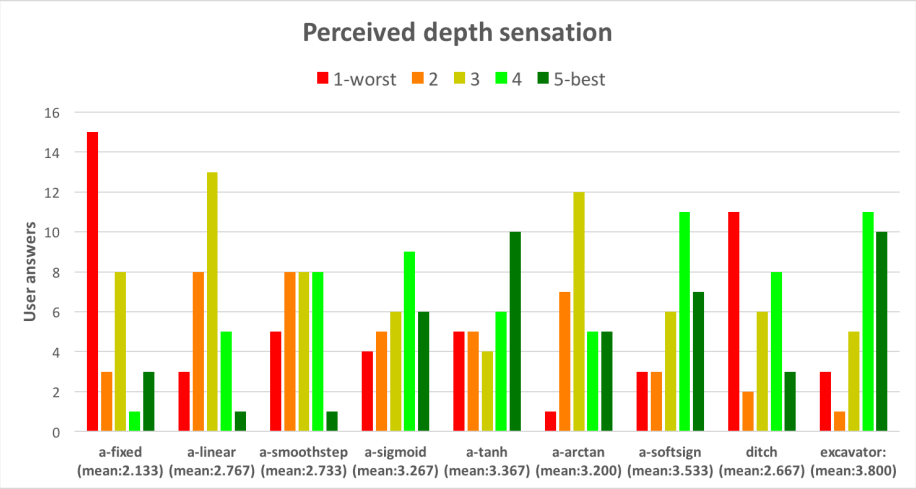


Figure 3.14: Answers of the survey participants per method.

The collaboration of 30 participants were interviewed for the study. All genders and ages between 16 and 75 are represented in the sample. 17 people from the sample declared they had previous technical experience. The answers of this questionnaire, organized by method, are introduced in Figure 3.14.

	People	Mean	Median	Standard deviation
Technical users	17	3.235	4	1.393
Non technical users	13	1.923	1	1.256

Table 3.2: Variability on the validation of ditch technique between users with and without previous technical experience.

The results of the survey reflect that the use of new techniques for underground visualization improves the depth cues obtained from using a static α blending. From the explored methods, the one which was best received by the survey participants was the excavation tool, with an average rating of 3.8 and a median of 4. From the variable α functions, survey participants tended to prefer the *softsign* function (average rating 3.53, median 4). Other options, like *sigmoid* and *tanh*, were also appreciated, but there was more variability of perceived effectiveness among the test subjects. The ditch technique was, on average and according to the survey participants, the least effective method, again with an average rating of 2.67, a standard deviation of 1.47 and a high

variability in the responses. With respect to the ditch methodology, a trend was also observed in the responses.

As Table 3.2 shows, users without technical experience had more difficulty in understanding the ditch technique and tended to interpret what they are seeing as over-surface structures, while most of the technical users found it effective. Considering that tools which deal with underground utility information are normally intended for people who have technical experience and work in the field, the ditch technique could still be a possibility to present the underground information in a multimodal application for which the target user group is one with significant technical experience. The excavation tool and the variable alpha blending appear to be the most generally accessible methods. However, these results suggest that a more thorough investigation should be conducted on how the previous technical information of users affects their comprehension of a 3D scene showing 3D subsurface network elements.

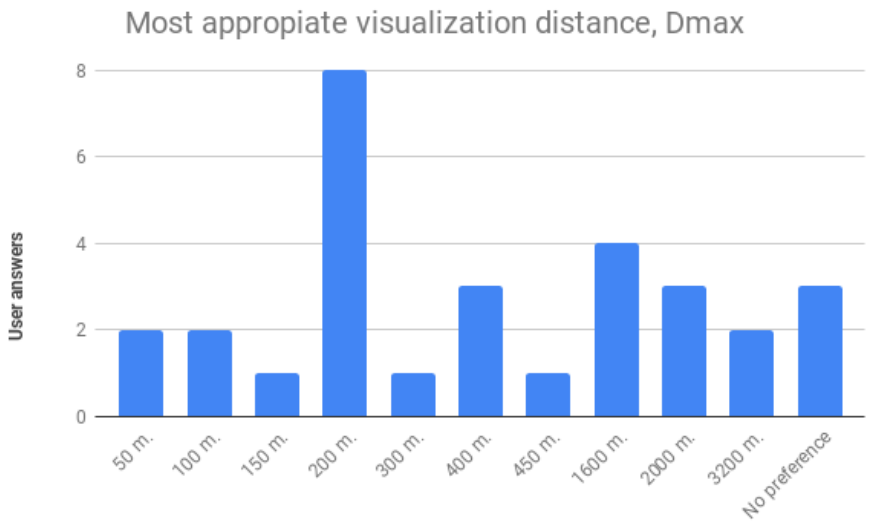


Figure 3.15: Distribution of preferred D_{max} distances.

With respect to the most appropriate D_{max} , the mean of the responses was 844.44 m., with a standard deviation of 972.84 m. However, almost two thirds of the users preferred a distance between 100 and 500 m, as it could be seen in Figure 3.15, which makes the median value, 300 m, an acceptable initial setting for D_{max}. The mode value, 200 m., is also a good initial setting for D_{max} in the *MultiVis* app, due to its disproportionate representation in the overall responses. Although, several users indicated they preferred to maintain a general view of the utilities in the area, and therefore chose a D_{max} larger than a kilometer. This fact suggests that a multimodal application which wants to take advantage of a variable α blending method for visualization should offer

this option as a user-adjustable parameter.

3.6 Conclusions

In this chapter, several alternatives for the visualization of underground structures have been discussed for its use in a multimodal application. An alpha blending strategy has been considered in which the viewer-object distance is mapped to the transparency of each feature in different ways to give depth cues to the user. A set of mapping functions where implemented and analyzed for such a purpose. Furthermore, a novel technique for representation of subsurface utilities, named "ditches", has been proposed. Finally, an extension for multimodal applications of the excavation tool of Schall et al. [57] was further developed. These techniques have been implemented as an extension of the *MultiVis* app [62], a proof of concept for the integration of CityGML spatial data and the Glob3 Mobile library for 3D data visualization to test whether the CityGML data standard is suitable for multimodal visualizations.

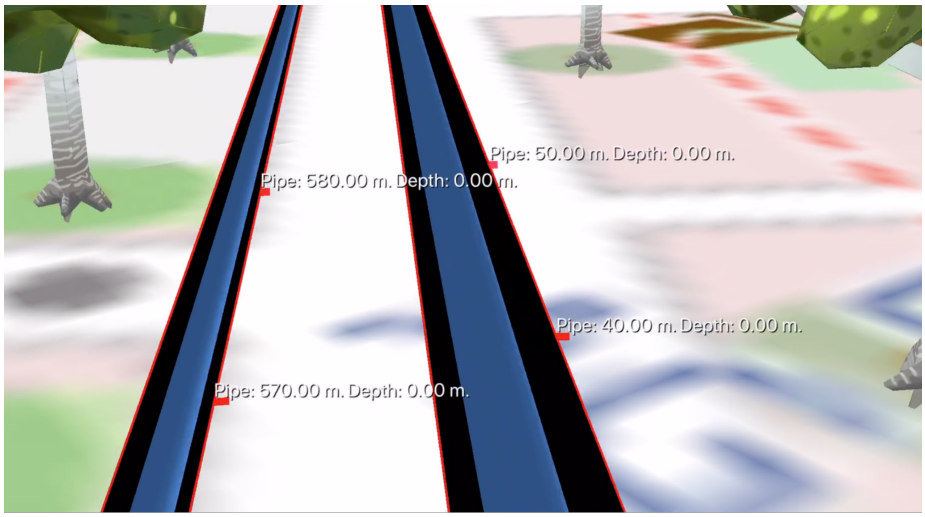


Figure 3.16: Proof of concept of ditch highlighted with borders to enhance depth perception. Evenly spaced tick marks along utilities also provide further information to the user.

The behaviour of these techniques were validated by conducting a user experience survey with 30 participants in which all the visualization techniques are demonstrated to represent different underground water pipe networks in the case study located in the city of Karlsruhe, Germany. The results reflect that all techniques are more effective than the reference method, a static transparency blending. In general, users found more effective depth cues while using the excavator tool extension. The distance function which achieves better visualization

results for variable transparency blending was found to be the softsign function method. This method was also rated as the overall second best rated technique. Most participants preferred a maximum distance of subsurface utility network feature visualization between 100 and 500 meters. Finally, a pattern in the ratings of the ditch technique suggested that the previous knowledge and experience of the users may affect their comprehension of the visualization of subsurface utility networks.

However, the prototype and the techniques introduced in this chapter still have limitations to render some underground network elements. Namely, the connection of underground utilities with buildings and other above-surface structures has to be factored into the rendering pipeline. Finer detailed models would also be of use for photorealistic applications, which would have to be produced by the tessellation system. Moreover, only CityGML LoD2 models have been used due to the lack of higher detailed city models. Those models could be too coarse for realistic representations, especially in the VR mode, where the user has a complete immersive experience. In the same way, for use cases as the AR that require high positional accuracy levels, GPS positioning may lack the required accuracy.

Figure 3.17: Proof of concept of a *cartoonish* hole styling with user interaction for the excavation tool.

These facts open future lines of work which involve working with visual odometry tools and beaconing devices to overcome this issue and provide higher positional accuracy of the app. Improvements in the general appearance of the ditches and holes, like the proofs of concept shown in Figures 3.16 and 3.17, can be studied in order to see it they improve the understanding of the scene. Additional user interactions can be integrated to better adapt the excavation tool to the needs of multimodal applications, and further user experience sur-

veys should be conducted in order to determine possible enhancements on both visualization and usability.

Besides these further technical implementations, it will be necessary to establish and extend current cartographic rules for the representation of underground features. From the literature review and the current research and commercial applications it is extracted that most applications that show underground infrastructure are lacking on the visualization side. Current cartographic guidelines and visual variables are lacking the support for underground infrastructure visualization. Existing cartographic principles should be adopted to the usage and visualization of underground infrastructure in outdoor environments in immersive ways. Although Becker and König [83] already presented first representation strategies for utility network data, they are still lacking visualization guidelines for utility networks in immersive environments. The establishment of such new rules will be beneficial for further developers to create better and more intuitive visualizations and applications.

Chapter 4

Modelling buildings from open vector data

In the previous chapters, two different proposals have been introduced which involve point data and line data, respectively. This chapter introduces the third main type of vector data: the polygon. Unlike lines, polygons always form closed circuits, connecting at least three points starting and ending in the same points. Polygons can be found in country, region and municipality borders, perimeters of lakes, seas or forests, cadastre data or geo-referenced 3D models, among others. The latter two ones play a major role in the development of smart city applications.

Traditionally, 3D city models were purely based on graphic and geometrical models and developed only for visualization purposes. However, the generation of such models has changed in the last years as semantic aspects have been progressively included for analysis purposes. Standards for data modeling as CityGML [70] have been created to add thematic properties, taxonomies and feature aggregation to the graphical appearance. This way, different items (e.g. building installation) and data (e.g. building energy characteristics) can be integrated within a single framework at different *Levels of Detail* (LoD) [84].

However, for smart city modelling the requirements are different than for visualization. Smart city applications require geometrically correct surfaces, which are not always achieved when the model is generated via photogrammetric techniques [85]. Errors can be produced due to wrong definition of normal vectors, lack of planarity in the building surfaces or polygon nesting in a single surface [86]. Some of these problems require a significant amount of time to be fixed and thus the generation of simplified building models (CityGML LoD1 and LoD2) are preferred as they fulfill the energy analysis requirements.

The *Light detection and ranging* (LiDAR) [87] technology can ease the automatic generation of such simplified models. Sensors based in this technology emit a laser pulse and receive an echo response from the objects in the targeted area. The time between the emission and that incoming echo and the direc-

tion of the laser beam are used to obtain the position of the object in which the laser rebounds. The accumulation of these positions generate a point cloud that represents accurately the area near the sensor. LIDAR devices can be differentiated in two categories: waveform sensors, which offer the whole laser return, and discrete sensors, which capture only a certain number of returns. This number varies depending on the device: most models provide between 3 and 5 returns. The generated data is normally presented as point clouds, whose points include, at least, accurate X-Y-Z values, an intensity value and its associated return number. Some sensors also add extra information about each point, like RGB color values. A frequently used format to represent these point clouds is the Laser File Format (LAS), whose specifications can be found in [88].

In this chapter, a novel methodology is proposed to generate standardized 3D building models out of open LiDAR data, so the resulting models are suitable for analysis and modelling in smart city applications and not limited to any geographic region. The methodology is capable to differentiate between five categories of buildings according to their rooftops and propose a CityGML LoD2 model for the building based on the data and the chosen category. This research has been done thanks to a partnership between the *University of Las Palmas de Gran Canaria* and the *European Institute for Energy Research* (EIFER) and has been already introduced in [89]. It addresses two current issues in the generation of 3D building models from open LiDAR data sets: (1) the need for a more precise feature extraction of rooftops from LiDAR point clouds, and (2) the development of novel line generalization algorithms that can be used to automatically generate geometrically and semantically correct 3D models for the integration, analysis and modelling of smart cities applications and processes.

4.1 Related work

During the last decade, many authors have worked on different methodologies to reconstruct 3D building models from LIDAR point clouds. Their approaches are usually classified into two categories: data-driven and model-driven [90].

On one hand, model-driven methods try to reconstruct building models by adjusting well-known primitives, represented by a number of parameters, to a subset of data from the point cloud. This kind of methods have the advantage of being robust and easy to compute, but they still have challenges where structures are quite complex or do not match well with the predefined models. An example can be seen in the work of Henn et al. [91], where plane primitives are extracted by using RANSAC and the selection of the best model relies on a *Support Vector Machine* (SVM). Yang and Förstner [92] used *Minimum Description Length* (MDL) instead of SVM to determine the best possible model from the primitives. Zheng and Weng [93] decomposed complex footprints into non-intersecting, quadrangular blocks and calculated different parameters for every block. A decision tree uses then the parameters to classify every block into one of 7 possible roof categories. Zhang et al. [94] combined LiDAR data and aerial imagery to generate a cost function that help optimize the geometric primitives

that compose a roof. In the work of Kada and McKinley [95], complex footprints are also decomposed and their normals calculated. Then rules are established to discriminate the rooftop into 5 possible categories according to its major planes. And more recently, Castagno and Atkins [96] used the output of a convolutional neural network as input for SVM and decision tree classifiers for 8 types of roof models.

On the other hand, data-driven methods aim to find features, such as planes or line boundaries, from the raw dataset and then aggregate them to generate the building model. Normally, those methods are more accurate and work on any kind of structures, but have a higher computational cost.

Data-driven works differ on how to process points and generalize outlines. Several works [97, 98, 99, 100] use region growing to segment points and find candidate roof areas. Clustering techniques are applied for similar reasons in [101, 102, 103, 104]. Wang et al. [105] used a voxel-based algorithm to segment buildings and extract roof points. In the work of Awrangjeb and Fraser [106], building masks were generated to extract points belonging to each building. A genetic algorithm is used in the work of Pahlavani, Amini Amirkolaee, and Bigdeli [107] to select the best features from a set to find the building areas. The study of Yang et al. [108] takes advantage of the scale-space theory to generate a graph of roof features and generate diverse LoD representations of each roof. Yan et al. [109] separated non-ground points and presents a modified version of the snake algorithm to fit the surfaces. Outlines are extracted after the generation of triangles from points in [110, 111, 112]. Awrangjeb [113] presented an algorithm to better extract edges and corners and thus generalize better a complex building footprint. Finally, Wu et al. [114] used a graph-theory-based contour tree to represent the topology structure of buildings. Building parts are separated analyzing topology relationships and final models are reconstructed via bivariate graph matching process.

Some other works also combined both approaches, as the one of Fan, Yao, and Fu [115], which regularize all kinds of non-flat rooftops as sets of gabled roofs and looks for roof ridges for extraction. Many other relevant studies that deal with 3D reconstruction of buildings from LiDAR point clouds are included in the survey of Wang, Peethambaran, and Chen [116].

Despite the large amount of studies that have been already done in relation to modelling buildings, it is still possible to find some research gaps. One of them is that most of the methods used for segmentation of rooftops in previous works are either dependent on a predefined number of clusters (k-means), too heavy in terms of computational cost (region growing) or they were proposed for detection of curved buildings [102]. There is still space for proposing a solution which does not require to set the number of clusters and can be used in a larger amount of cases. A second one is the fact that that regularizing a footprint is still an open problem. Common line simplification algorithms, as Douglas-Peucker [117], usually depend on a non-intuitive parameter which can remove critical points when not well set and thus the regularized line will not respect the original shape of the surface. Other algorithms, such as the one of Zhang, Yan, and Chen [118], look for detecting the two principal directions of the

buildings. That implicitly assumes the buildings always have a quadrangular shape and thus all angles are of 90º. Finally, snake-based algorithms either require knowledge about the expected shape or could result in an inaccurate footprint shape. There is still room to make improvements, such as proposing an algorithm which can generate footprints out of multiple building shapes with none or at least a short and intuitive interaction of the user.

The approach which is introduced in this chapter tries to overcome the research gaps and take advantage of both model and data driven approaches. It relies on clustering to find candidate roof surfaces, but using anisotropic filtering and agglomerative clustering techniques, unlike other clustering works. A new corner-based algorithm is introduced to extract footprints.And additionally, the methodology makes use of MLESAC [119] to find plane primitives and apply rules to categorize the rooftops.

4.2 About the data sets

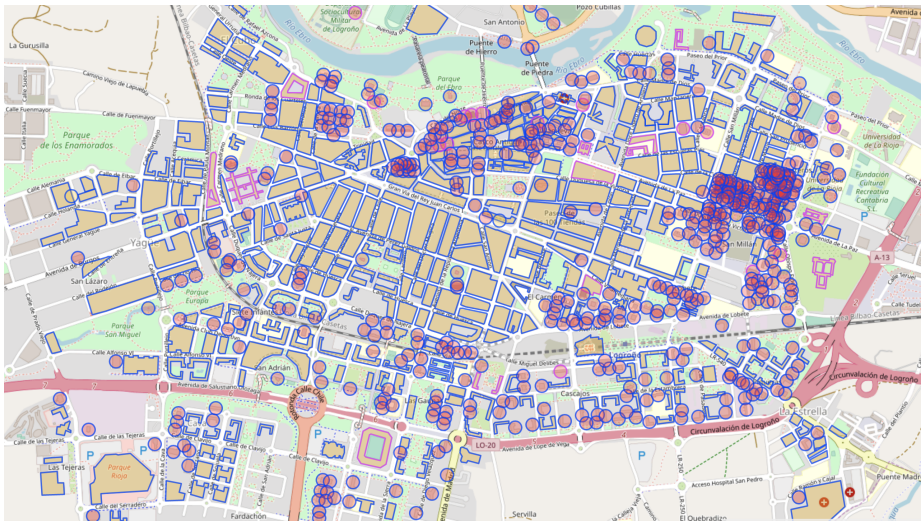


Figure 4.1: Map showing the studied area in Logroño and the available OSM footprints.

For this work, free and open LIDAR point clouds and building footprint data for the generation of the 3D city models are required. It was chosen to use an open point cloud data from the Spanish Geographical Institute representing the city of Logroño. The cloud, which can be found at [120], has 12086959 points and covers an area of 2×2 km. with a minimum density of 2 points per m^2. The bounding box of the chosen area is $[42.4511, -2.4649; 42.4690, -2.4404]$. This area was chosen because of it having a wide variety of rooftops belonging to old and modern buildings in a relatively large extension, for which their cor-

responding footprints are available online. An image representing the selected area can be seen in Figure 4.1.

Likewise, open footprint polygon datasets can be found in several web mapping services, such as OSM, as well as in many geo-referenced data portals belonging to local and national mapping agencies. For this study, a dataset containing 454 footprints from the same 2 km. area of the city of Logroño has been downloaded and processed using the Overpass API [121]. Those footprints represent 1261 individual buildings with a minimum area of 30 m^2. For the quality assessment of the methodology proposed in this chapter, a ground truth of roof categories was manually crafted for each one of the buildings.

4.3 The proposed pipeline

In this chapter, a pipeline for the generation of LoD1 and LoD2 CityGML models from open LiDAR point clouds is proposed, whose stages are introduced in Figure 4.2.

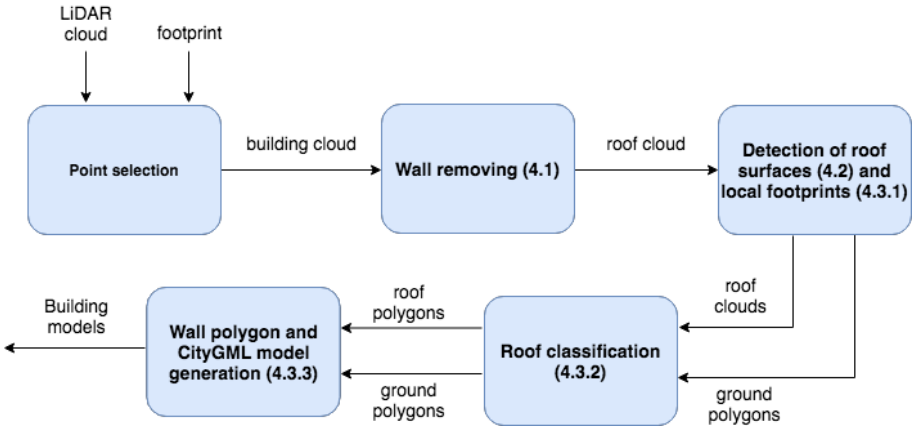


Figure 4.2: Architecture of the proposed pipeline for city model generation from LiDAR data.

As an overlook of the complete proposed pipeline, the process can be decomposed on the following tasks:

- Segmentation of the point cloud and selection of the points of a certain building, given a LIDAR point cloud and a building footprint.
- Discrimination of wall and roof points. Wall points are then dismissed.
- Grouping of roof points into different roof surfaces belonging to different buildings inside the footprint, when necessary. In that case, smaller footprints for each cluster are automatically generated.

- Searching of planes and extraction of features and intersections between planes. That information is used to categorize the rooftop. 5 categories are defined to do so: *flat*, *shed*, *hipped*, *pyramidal* and *complex*.
- Generation of roof polygons and wall polygons to complete the building model.

The stages of the pipeline are thoroughly explained in the following subsections.

4.3.1 Removing wall points

Given all the points inside of a footprint, discriminating which points belong to vertical walls can be achieved by categorizing the horizontality of their associated normals. To do so, the normal vector of each point is computed according to their 8 nearest neighbors by applying the method of Hoppe et al. [122].

After that, the verticality θ_V of the plane which best fits each point is assessed, using Expression 4.1, where N refers to the normal and Z is the upwards direction.

$$\theta_V = \arccos(|N \cdot Z|) \tag{4.1}$$

It is expected that a point belonging to a wall has a θ_V angle near $\frac{\pi}{2}$ (90°), so any θ_V in the range $[\frac{\pi}{2} - \omega, \frac{\pi}{2} + \omega]$ is considered to belong to a wall. The value for ω has been set experimentally and can be found in Section 4.4.

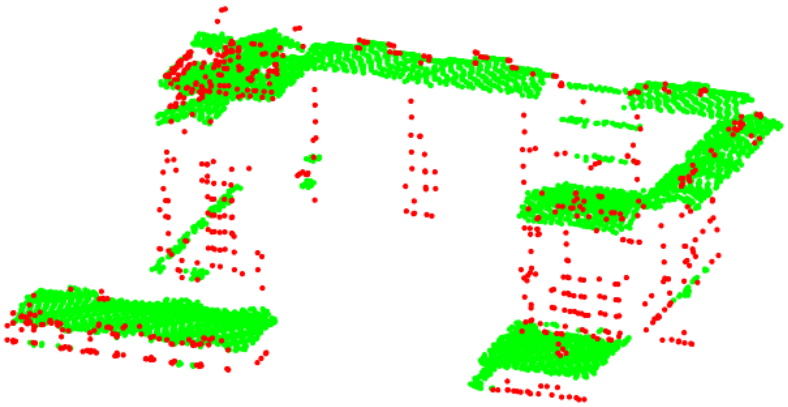


Figure 4.3: Discrimination of wall and noisy points (red).

The remaining roof points are then filtered to simplify the ulterior generation process. Given the normal and mean position of a point and its neighbours, a container plane is established for that wall patch. Points too distant from their

corresponding planes are considered outliers due to noise or unnecessary details that will be not shown on the final model.

This point-patch distance can be computed by solving Expression 4.2, for each point (x_0, y_0, z_0) and the plane $a \cdot x + b \cdot y + c \cdot z = d$:

$$D_p = \frac{a \cdot x_0 + b \cdot y_0 + c \cdot z_0 + d}{\sqrt{a^2 + b^2 + c^2}} \tag{4.2}$$

By means of D_p , the "noisy" points are also removed from the dataset when $|D_p| < \varepsilon_1$ m. A point will be considered *wall point* and removed if it has passed one of the two tests. In Figure 4.3, it is possible to see a result of this stage.

4.3.2 Recognizing different roof surfaces

The resulting non-wall points belong to one or multiple roof surfaces, depending on the number of adjacent buildings inside the footprint. The remaining points also may include little ground areas that should be removed. A clustering algorithm should be applied in order to segment the point cloud into different surfaces. In the choice it should be taken into account that a certain number of groups cannot be expected, and also the fact that some urban clouds lack of enough density could make harder to detect the correct surfaces.

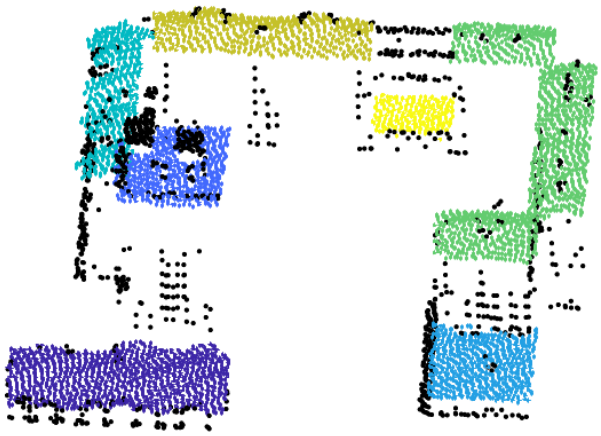


Figure 4.4: Different roof surfaces, shown in different colours, found for the example of Figure 4.3. Black points represent walls, ground or noise.

To overcome these issues, a local transformation is applied to the point cloud prior to the agglomerative clustering algorithm [123]. The anisotropic scaling transformation exaggerates differences in the z coordinate and diminish the differences in x and y coordinates, and it is applied to all the non-wall points. This is shown in Expression 4.3, where f_{xy} and f_z are the factors to transform the point cloud. f_{xy} depends on the point cloud density and should be a number

between 0 and 1. On the contrary f_z always be greater than 1 for exaggeration. The final sets of both values in this work can be seen in Section 4.4 as well.

$$p'_i = p_i \odot [f_{xy}, f_{xy}, fz]' \tag{4.3}$$

Finally, the agglomerative clustering algorithm is run over the transformed cloud. Euclidean distance is used to aggregate points into a group. A cutoff ε_2 representing the maximum allowable distance to agglomerate points and clusters should also be defined. An example of how different roof surfaces could be found using this stage is introduced in Figure 4.4.

4.3.3 Classifying the roofs and generating the building model

Each roof surface detected in the previous stage of the pipeline represents a building. For every building, a CityGML, LoD2 building model are generated. A building model requires a ground polygon, a variable number of roof polygons (depending on the type of roof) and a variable number of wall polygons. The goal is to generate a mesh of minimal complexity while still conserving the shape of the building.

Depending on the number of different building surfaces detected inside a footprint, it can be necessary to extract and generalize a ground polygon for each one as a first step. That only happens when more than a building surface is detected in the local point cloud for the input footprint area and so several building models should be created. Then, the type of roof will be estimated from the planes it consists of and their intersections, and different roof polygons will be generated according to it. Finally, wall polygons are generated between the ground polygon vertices and the roof polygon ones, as an extruded polygon with the footprint as the base.

The ground footprint: a corner-based polygon simplification algorithm

When multiple building surfaces are detected, new ground polygons should be created for all of them, as the original footprint represent the united set of buildings and not each individual building. A first approximation on the matter is to generate a unique 2D α-shape [124]. The algorithm looks for finding the minimum largest allowed edge (the α value), which encloses the roof surface on a single polygonal boundary. The boundary created with this technique has much detail and generates small and irregular lines, that should be simplified so the representation remains close to the actual building. To do so, a novel corner-based polygon simplification algorithm is proposed. It removes needless vertices by using the following strategy:

1. For each boundary point, a window of m points and the two polylines before and after each point p_i, $ll_i = [p - m, \ldots, p - 1]$ and $rl_i = [p + 1, \ldots, p + m]$, are defined. Considering the polygon as a circular list, point

indices are modular. Given this, we can define the left and right difference vectors, v_l and v_r, as in the following Expressions 4.4 and 4.5:

$$v_l = \sum_{1}^{m} \left(ll_i - p_{i]} \right) \tag{4.4}$$

$$v_r = \sum_{1}^{m} \left(rl_i - p_i \right) \tag{4.5}$$

And an angle β_p between the v_l and v_r vectors is computed as in Expression 4.6:

$$\beta_p = \pi - \arccos\left((v_l \cdot v_r') \div |v_l| \div |v_r| \right) \tag{4.6}$$

2. The β_p computed for all the vertices can be expressed as a function $f(p_i)$. Therefore, the local maxima of that function can be found. This can be appreciated in Figure 4.5. All the peak points whose β_p value is greater than a salience threshold θ_C considered corner candidates. θ_C can be adjusted according to the shape of the buildings in the set and the user needs.

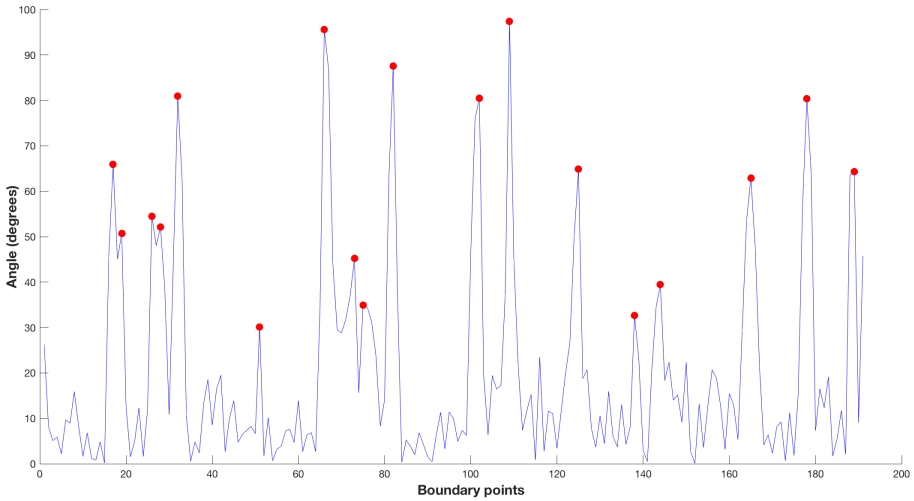


Figure 4.5: Corner candidates chosen from the polygon vertices of the green rooftop from Figure 4.4, based in their β_p values and $\theta_C = 30°$.

3. For all the points between two corner candidates, including them, the line that fits them the best is computed. The intersections between all the lines are then found, and the corner positions are updated with the ones of these intersections. This process results in the final footprint of the building. The final result for the example of Figure 4.5 after applying this correction can be seen in Figure 4.6.

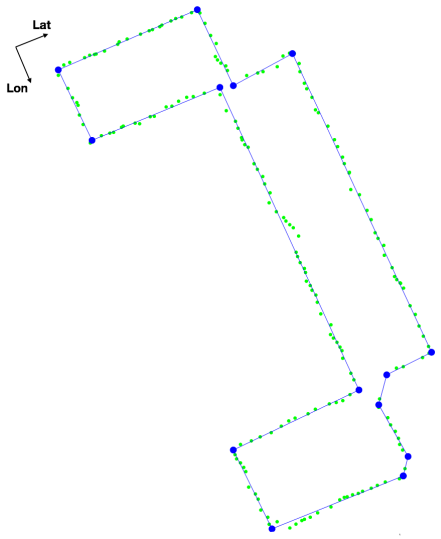


Figure 4.6: Final building ground footprint for the green rooftop in Figure 4.4 (blue line), against its initial boundary generated using 2D α-shapes (green points).

Identifying the roof category

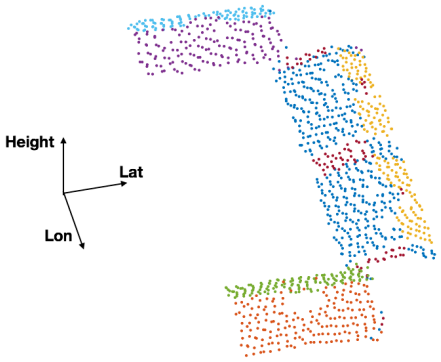


Figure 4.7: Plane extraction for the complex roof surface sample from Figures 4.4, 4.5 and 4.6.

In order to determine the category of the roof and thus create the necessary roof mesh, it is required to find all the planes within the surface. To do so, MLESAC [119], a variant of the RANSAC algorithm, is used. Some parameters of MLESAC can be configured, being the most important one the error margin, ε_3, to select planes, which is set experimentally. As an example, the plane extraction done for the roof surface sample used in Figures 4.4, 4.5 and 4.6 can

be seen in Figure 4.7.

Once the roof planes are defined, a set of several rules categorize the rooftop:

1. There is a unique plane, or one of the planes occupies more than 70% of the rooftop area: the rooftop is classified as *flat* or *shed*. When the difference in height between all the points in the plane is below 1.5 m., the category is *flat*. In that case, the final simplified building mesh is the extruded footprint polygon with the mean z of the rooftop points as height. Otherwise, the category is *shed*. In this case, the z values for the roof polygon vertices equals the height of the closest point in the roof cloud.

2. There are two intersecting planes and together they occupy at least 60% of the rooftop area: the rooftop is considered *hipped*. In this case, two roof polygons are created in the following manner:

   (a) An extruded polygon *ep* is generated in an analogous manner to the flat case. The z components are set to the lowest height in the roof surface.

   (b) The intersection line between the two planes is calculated. From all points close to the line, the two in both ends of the line, h_1 and h_2, are included in the polygon, between their closest two vertices. The z component of the two new vertices are assigned the highest height in the roof surface.

   (c) The final rooftop polygons are extracted from the footprint by dividing *ep* in two using the line vertices as pivots: $r_1 = [h_1, \ldots, h_2]$ and $r_2 = [h_2, \ldots, h_1]$.

3. There are n planes that mutually intersect and together occupy at least 90% of the rooftop area: the rooftop will be considered *pyramidal*. In this case, n roof polygons are created as a triangle fan:

   (a) A base polygon is the top of the extruded polygonal footprint.

   (b) The peak of the rooftop is the intersection point of all the roof planes, which can be assumed the highest point of the roof that belongs to all planes. The roof polygons consist of a pair of consecutive base vertices and the top position.

4. In any other case, the roof surface is categorized as *complex*. In this case, a sub-roof surface is created for any detected plane, using the methodology introduced for the ground polygon. It looks for ensuring the generating of a result regardless of the shape of the footprint, so triangular or circle shapes can be processed. For the generation of each sub-roof polygon, the footprint, the points inside each plane and the intersections between planes are considered. When there are points of the two involved planes close to a given intersection line, the polygons for both planes share vertices in order to avoid possible holes in the generated roof.

Generating wall surfaces

A wall surface polygon consists of a pair of consecutive ground surface vertices, g_1 and g_2, and at least two rooftop polygon vertices, t_1 and t_2, the closest to the ground surface vertices.

There is a distinction between the *hipped* roof case and the general case. In the general case, the wall polygon is generated following the order: $[g_1, t_1, t_2, g_2, g_1]$. For the special *hipped* case, two of the walls are include one of the line intersection points, so the polygon will be generated in the order: $[g_1, t_1, h, t_2, g_2, g_1]$. The remaining walls in the *hipped* cases will be created as in the general case.

4.4 Validation of the pipeline

This section introduces the final settings of each parameter of the methodology and the experimentation conducted to validate its behaviour.

4.4.1 Adjustments of the pipeline parameters

The proposed methodology depends on seven input parameters, ω, ε_1, f_{xy}, f_z, ε_2, θ_C and ε_3. They refer respectively to vertical angle offset, outlier removal, anisotropic factors, clustering cutoff, minimum expected angle for footprint corners, and plane adjusting error margin. For replication purposes, the final values of all the parameters during the experimentation are provided in Table 4.1. Most of them have been chosen after a mono-objective optimization with multiple tested values. For that cases, the range of options and the step between each tested value is also provided.

Name	Type	Range	Step	Value
ω	Mono-objective optimization	0:30	1	8 (º)
ε_1	Mono-objective optimization	0.05 : 0.25	0.05	0.2 (m)
f_{xy}	Mono-objective optimization	{0.1,0.2,0.25,0.33, 0.5,0.66,0.75}	-	0.25
f_z	Mono-objective optimization	1:25 : 4	0.25	2
ε_2	Mono-objective optimization	0.5 : 3	0.5	1
θ_C	Semantic	-	-	30 (º)
ε_3	Mono-objective optimization	0.05 : 0.25	0.05	0.1 (m)

Table 4.1: Configuration settings of the different parameters of the pipeline.

4.4.2 Tests and practical results

The proposed methodology has been run over the LiDAR dataset described in Section 4.2 in order to generate a city model in CityGML format of the city of Logroño. The main goal of the experimentation is to determine whether the

pipeline is able to properly classify each type of roof and generate a model for them.

		Predicted				
		Flat	**Shed**	**Hipped**	**Pyramidal**	**Complex**
Real	**Flat**	372	0	4	0	1
	Shed	0	52	2	0	1
	Hipped	9	1	223	8	5
	Pyramidal	1	0	2	45	1
	Complex	5	0	14	25	490

Table 4.2: Confusion matrix of roof surfaces.

A confusion matrix is generated out of the comparison between the ground truth and the output of the process and introduced in Table 4.2. Additionally, five quality assessments have been computed and analyzed: the *overall accuracy* of the method and four class-wise scores: *recall*, *precision*, *F1* and *IoU*. Per class, the achieved results in all the scores are presented in Table 4.3.

	Flat	**Shed**	**Hipped**	**Pyramidal**	**Complex**
Recall	98.7 %	94.5%	90.7%	91.8%	91.8%
Precision	96.1 %	98.1%	91.0%	57.7%	98.4%
F1 score	0.974	0.963	0.908	0.709	0.950
IoU score	0.949	0.929	0.832	0.549	0.904

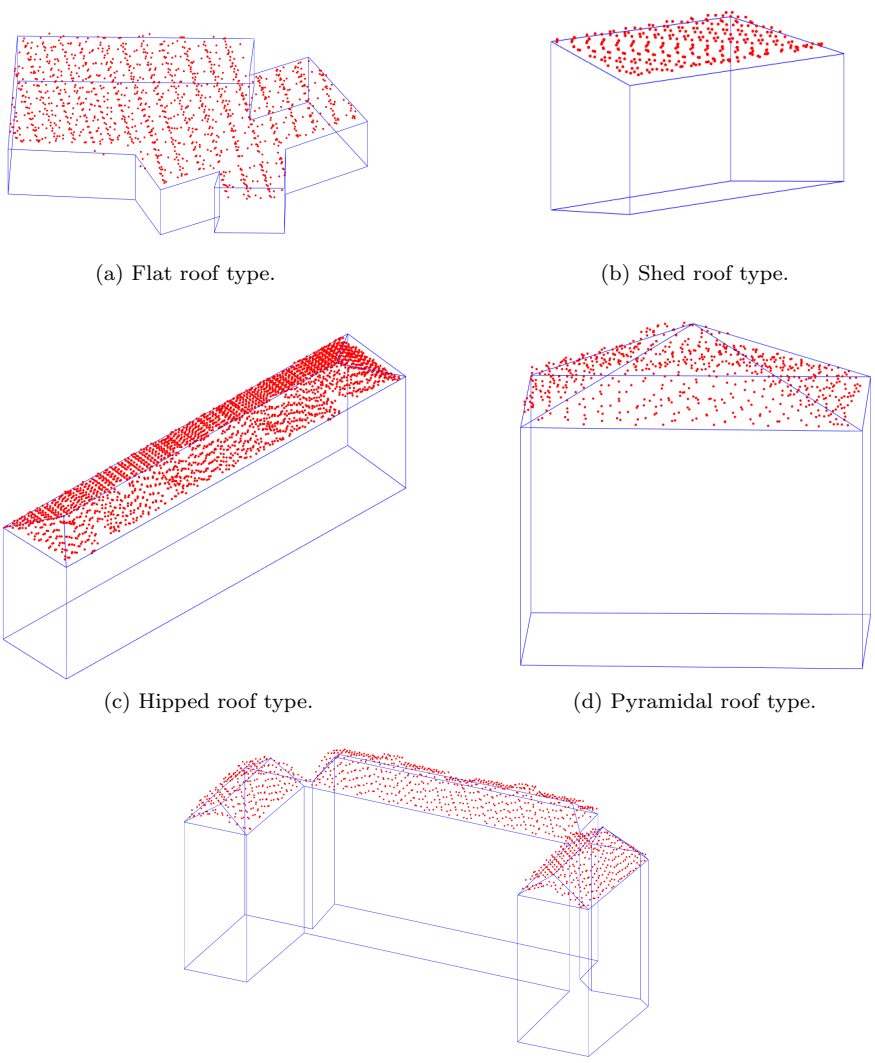
Table 4.3: Quality assessment of roof classification per class.

The obtained results are promising in general, showing an overall accuracy of 93.74%. Per category, the pipeline behaves properly for detection of flat and shed buildings, with F1 scores over 96% in both cases. For hipped class, the performance is a bit worse but still competitive, being the recall, precision and F1 scores around 91%. Additionally, the most correct predictions are achieved for the complex class (98.4%) for a F1 score of 95.0%, which demonstrates the robustness of the solution. However, there are still issues to be solved regarding pyramidal roof classification. Although most of the pyramidal roofs are found correctly (45/49, 91.8% of completeness), the criteria which filter pyramidal roofs are not sufficiently strict, resulting in a small but appreciable number (25/534) of complex roofs misclassified as pyramidal.

In Figure 4.8, the CityGML model generation for a sample of each roof class is shown, compared with its corresponding building point cloud.

Additionally, an integration test was done by loading the output, which contains the generated CityGML models, in FZKViewer[1]. By using this publicly available software, it is checked whether the pipeline results are correctly expressed in the standards. This allows to demonstrate the usability of the final

[1] `https://www.iai.kit.edu/english/1648.php`



(a) Flat roof type.

(b) Shed roof type.

(c) Hipped roof type.

(d) Pyramidal roof type.

(e) Complex roof type.

Figure 4.8: Meshing of final building models (blue) against their associated point clouds (red).

result by the community. A sample of the visualization of the obtained results can be seen in Figure 4.9.

As a final observation, it is worth noting that the size of the final city model is 21 Mb. This implies a reduction to a 8.64% of the original size of the point

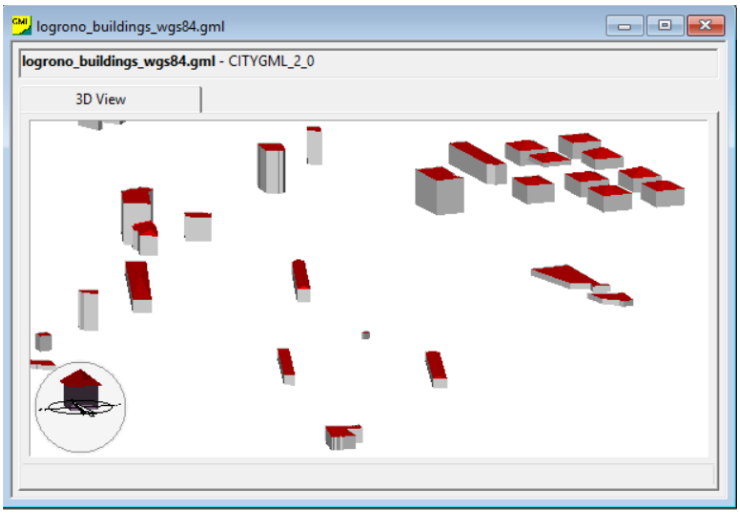


Figure 4.9: Subset of buildings from the city of Logroño, generated using the proposed methodology and visualized using the FZK Viewer for CityGML models.

cloud (245 Mb). Combined with the use of well-known standards, it makes the data more suitable for their transmission and visualization online in any device.

4.5 Conclusions

In this chapter, a methodology which combines model and data driven algorithms is proposed to extract 3D city models from LiDAR point clouds and OSM polygons. It starts filtering out wall points and checking whether the OSM polygon envelopes one or several buildings. The latter case is addressed by using a novel corner-based algorithm to generate a footprint for each individual construction. Plane extraction is performed from the rooftop points and a rule-based system is proposed to categorize the rooftop in base of the extracted planes and their intersections.

The pipeline behaviour has been analyzed using open data as input. A LiDAR point cloud dataset from the Spanish government that contains 12.1 million points and has a spatial resolution of 2 points per m^2 is combined with 454 OSM polygons which enveloped 1261 buildings in the same area. Promising results have been achieved, with F1 scores superior to 95% for flat, shed and complex rooftop predictions, and a 91% in the same quality score for the hipped category. However, a limitation is found in the detection of pyramidal roofs. Low precision (57.7%) is observed due to false pyramidal positives in complex building. The generation of CityGML LoD2 models is also tested. The 245 Mb input point cloud is translated into a 21 MB model. The final output followed the standards and is properly visualized in a popular open-source software for

semantic data visualization.

Two lines of work are open for future research. One of them involves improving the rule system to better differentiate pyramidal and complex buildings and apply the proposed method in other datasets. Using only point clouds from the open LiDAR repository from the Spanish government, the potential of city model generation has the scale of an entire country. Additionally, other benchmarks and sources of different densities and countries can be explored as well. The other one is focused in comparing the resulting model with the input data to ensure the topological correctness of the output. Having accurate building models is key in order to exploit the city model in smart city and energy oriented applications. Those applications can include simulations of heating and cooling needs, energy consumption, solar radiation and potential for photovoltaic energy generation. Accurate models can also be used for visualization purposes in virtual globe applications, augmented reality and virtual reality, which demonstrates the utility of the research line opened in this chapter.

Chapter 5

Ground filtering and DEM generation from point clouds

As mentioned in the previous chapter, LiDAR is a remote sensing technology whose use has become widespread in the last few years for a variety of tasks. These tasks go from the 3D city modeling [116] to land cover detection and management [125], including others as power line management [126], aerosol gas detection [127] or autonomous driving [128].

Most of these tasks require an automatic and reliable detection of objects in the point cloud, which has led to the appearance of several algorithms for classification and generation of *Digital Elevation Models* (DEM) from LiDAR point clouds. However, as it is stated in the literature review survey from Chen, Gao, and Devereux [129], there is still no perfect methodology capable of dealing with all the possible scenarios (rural, urban, mountainous, etc) at once. Hence, the problem is still open for new solutions and improvements. This is easily visible by checking the results from commercial software used by companies throughout the globe to perform their tasks. Most of the ground filtering algorithms implemented in those software rely on parameters that are hard to adjust and could produce severe misclassifications, as the one seen in Figure 5.1. This fact forces subsequent undesirable human interventions to refine the results and properly complete the task.

In this chapter, a novel algorithm for ground filtering, called *Patch decision tree*, is introduced to minimize this kind of misclassification errors. It adds a clustering routine for minimum local points, whose result is a set of patches, to the classic multiscale analysis methodologies. Features are then extracted for each generated patch and it is determined whether it belongs to the ground or not by using a decision tree. The final classification can be later used in an extension of this work, presented in [130], as input for the generation of DEM in the form of triangular regular meshes.

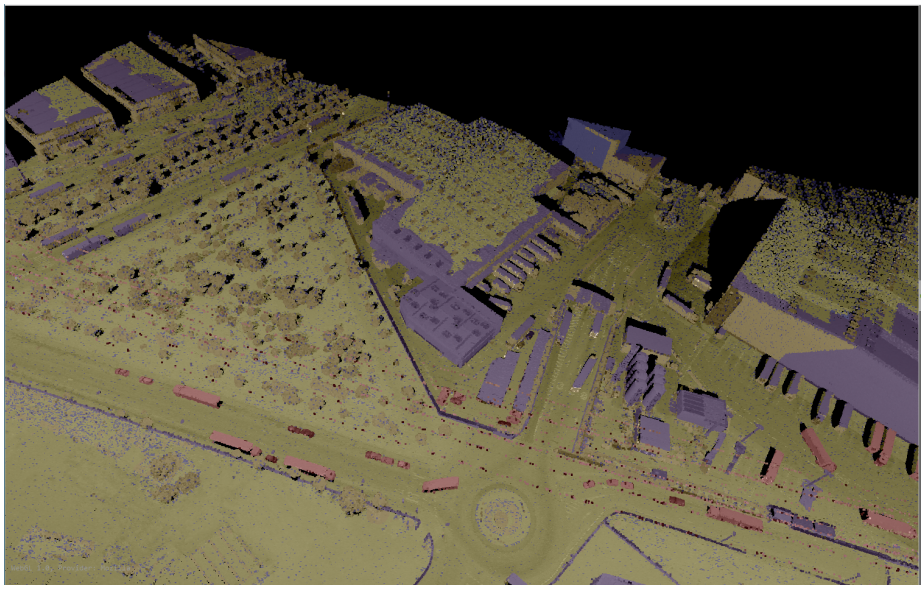


Figure 5.1: A classification of ground (yellow) and building (purple) generated using the commercial software TerraSolid. The majority of tools of this kind use window-based algorithms, which results in an inaccurate classification when the window size is not properly selected.

5.1 Related work

During the last years, a great number of authors have proposed different methodologies for classification of ground points and generation of digital terrain models from LiDAR data. Some of them become widely used and have commercial implementations, like the *Robust Filtering* algorithm of Kraus and Pfeifer [131] or the TIN adaptive filtering of Axelsson [132]. Although there are exceptions, most of the classical ground detection algorithms perform two steps [133]: converting from a raw point cloud to a regular grid, which usually requires an interpolation technique, and filtering the non-ground points from the cloud.

In order to convert from point cloud to grid, multiple techniques have been used. Linear deterministic and non-deterministic interpolators for this task have been analyzed in the work of Anderson, Thompson, and Austin [134], including the *Inverse Distance Weighting* (IDW) [135], the *Linear Least Squares* method [136] and the *Kriging* method [137]. Adaptations of morphological filters have been applied for this task in the work of Chen et al. [138] and surface fitting functions have been used in the work of Okagawa [139].

In the filtering part, the studies can be split into several categories, from which slope detection, *triangular irregular networks* (TIN), morphological filters and multiscale analysis are the most widely used. Examples of the morphological category are the work of Zhang et al. [140], which defined 3D morphological

operators of erosion and dilation, and the study of Li et al. [141], which introduced an improved top-hat filter which takes plane inclination into account to better distinguish between buildings and hill-like accidents. Morphological opening operations were combined with multi-scale windows to extract the rooftops in the work of Li, Sun, and Yan [142]. The *Elevation Threshold with Expanding Window* (ETEW) algorithm from Zhang and Whitman [143] and the three-windowed height threshold comparison from Rashidi and Rastiveis [144] are also good examples of multi-scale filtering methods. Inside the slope category, studies like the one of Sithole and Vosselman [145] can be found. They created a local slope operator based on a point-slope surface, from which the gradient is computed and a cutoff plane per point is generated. This cutoff plane is then compared against the cloud to filter non-ground points. The slope filtering method is improved for classification of gentle sloped urban scenes by Susaki [146]. Finally, examples of TIN-based works applied to LiDAR filtering are the one of Uysal and Polat [147], which directly applied the previous algorithm of Axelsson [132] to the raw point cloud and validate its behaviour with LiDAR data, or the study of Quan et al. [148], which used TIN to detect the borders of buildings and then applied region growing to filter the rooftop points.

Recently, the proposed approaches evolved to exploit the capabilities of machine learning for the detection of multiple classes in LiDAR point clouds, including ground. On this matter, Gu, Wang, and Xie [149] created the kernel-based MKSRC algorithm to detect four classes of urban elements: tree, building, wire and ground. Niemeyer et al. [150] used a 2-layered conditional random field approach which allows classification and segmentation of objects. Winiwarter and Mandlburger [151] adapted the PointNet++ cloud for detection of 9 classes from LiDAR point clouds, including two regarding ground: *impervious surface* and *grass*. With the same objective, Zhao, Pang, and Wang [152] and Yang et al. [153] presented different convolutional neural networks which exploit LiDAR and textural information for accurate prediction of different ground and non-ground elements. Wang et al. [154] introduced a deep neural network which uses directly the 3D features instead of making a previous conversion point-grid. And finally, Zhang et al. [155] proposed a graph convolutional network which considers the spatial relationships between ground and non-ground point for detection of ground in heavily forested scenarios.

Many other works with strategies different than those exposed here could be also highlighted, like the one of Bretar and Chehata [156] that uses Kalman filters and a Bayesian framework to generate DTMs from LiDAR. However, it is best to refer instead to the survey on all the ground classification techniques done by Chen, Gao, and Devereux [129] to go deeper on this field of work.

5.2 The patch decision tree pipeline

A pipeline is proposed to filter non-ground points in LiDAR point clouds. The pipeline architecture can be seen in Figure 5.2. It is composed of four stages: the dimensionality reducer, which converts the point cloud into patches and grids,

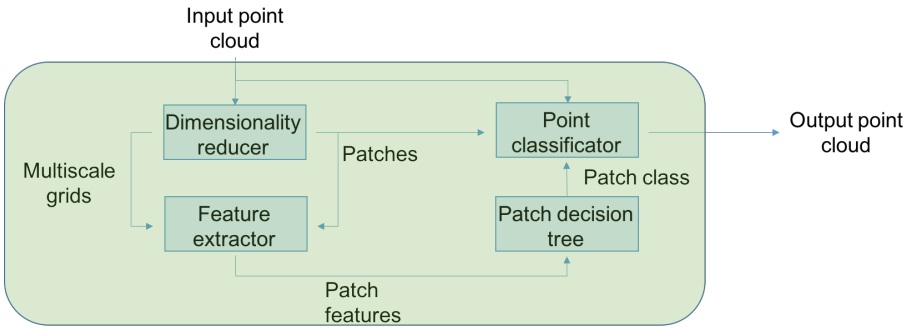


Figure 5.2: Architecture of the Patch Decision Tree methodology.

which are more efficient in terms of processing costs; the feature extractor; the decision tree for classification of patches; and the point classificator, which translates from patch classification to the final classification. Those stages are explained in detail in the following subsections.

5.2.1 Dimensionality problem mitigation: patches and multiscale grids

Point clouds normally have different average point densities and, potentially, tens or hundreds of millions of points. To overcome this problem, its dimensionality must be reduced, and this is achieved by associating each point to a cell in a rectangular grid according to its X and Y positions. A grid cell represents a prism with square base (normally 1 m.$\times$ 1 m.) and infinite height. From this association, two products are generated: the *patches* and minimum height multiscale grids.

Patches

From the input point cloud, a subset is defined containing all the points P whose height P_z matches the minimum height for their associated cell. Hence, at least one point per cell is inserted into the subset, and more than one for those cells associated with a fully flat surface.

For that subset, anisotropic agglomerative clustering [123] is applied. In order to stretch the surface, anisotropic factors are defined in their X and Y coordinates. The same is done in order to exaggerate the Z coordinate and thus highlight any possible step-like accident in the surface at the moment of grouping. Steps in the surface is a common sign of a human structure. The anisotropic scale factors applied are $S = [S_{xy}, S_{xy}, S_z] = [0.5, 0.5, 3]$. The sets of points resulting from this operation are called patches and can be appreciated in Figure 5.3.

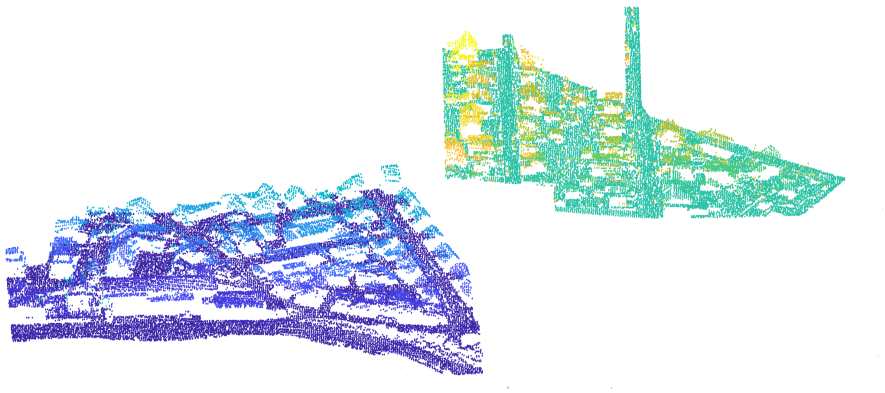


Figure 5.3: Generated patches for the ISPRS WG II/4 benchmark - Vaihingen.

Multiscale minimum height grids

To generate this product, the minimum height grid H [157] is first obtained from the cloud. Derived from it, three grids H_1, H_2 and H_3 are computed, whose values per cell are the minimum value of three windows w_1, w_2 and w_3 .Those windows are centered in the same cell of H and have sizes of $[n_1 \cdot 2+1, n_1 \cdot 2+1]$, $[n_2 \cdot 2+1, n_2 \cdot 2+1]$ and $[n_3 \cdot 2+1, n_3 \cdot 2+1]$, respectively. This is analogous to what is exposed in the work of Rashidi and Rastiveis [144], with the sole difference that now the window sizes, n_1, n_2 and n_3, can stay fixed instead of being tuned depending on the elements of the cloud. Those sizes are thought to capture short, medium and large sized objects and have been set as $n_1 = 2$, $n_2 = 4$ and $n_3 = 6$, respectively.

5.2.2 Feature generation

This stage computes different features and attributes for each one of the generated patches in order to determine whether the patch represents a ground area or other elements, like rooftops, cars or very forested areas where ground is not found during the scanning process. They are detailed after these lines:

- **Area**

  The area of the patch is considered to be the area of the 2D α-shape [124] polygon computed from the patch points, using the critical α. It is expected that a patch whose area is larger than a certain threshold (e.g. 10000 m^2 for a common football stadium) always represents the ground.

- **Amplitude**

  It is defined as the maximum height difference between all the points in the patch. As a patch only has minimum local height points, and a rooftop surface generally has an amplitude of less than 2 floors (around

8-12 meters height), it is to be expected that patches whose amplitude is larger than a certain amount α_2 represent the ground.

- **Height step ratios**

  The height step ratios RH_1, RH_2 and RH_3 are computed from the multiscale minimum height grids, $H1$, H_2 and H_3. The process for the computation of RH_1 is here introduced, being the ones for RH_2 and RH_3 analogous.

  Given a point P in the patch, which is associated with a cell $[i, j]$ in the grids, it is tested if their height P_z fulfills $P_z > H_1^{ij} + \theta_1$. The value of θ_1 is a minimum height threshold, introduced in [144], to consider a given point as non-ground, with the sole difference that it becomes invariant in this proposal, unlike in the referred study. The height step ratio RH_1 is then defined as the ratio of the number of points that pass this condition, divided by the number of points of the patch. It is expected that a value close to 1 for this feature augments the probability of the patch to represent a non-ground element.

  The thresholds θ_1, θ_2 and θ_3 are defined according to the goal of each scale grid: detecting small, medium and large objects, in the following manner: $\theta_1 = 1.75$m., $\theta_2 = 3$m, $\theta_3 = 5$m.

- **Covariance descriptors: planarity and eigen-entropy**

  The covariance features are extracted from a point neighbourhood by calculating the covariance matrix of their coordinates and extracting its eigenvalues, $\lambda_1 >= \lambda_2 >= \lambda_3$. These eigenvalues are widely used in the state of the art [152] to gather information from the point neighbourhood. Eight characteristics are commonly used [158]: eigen-sum, linearity, planarity, sphericity, anisotropy, omnivariance, eigen-entropy and omnivariance. From those, two of them are interesting for analyzing patches: planarity and eigenentropy.

  The planarity F_i of a point, given its neighbourhood and the covariance eigenvalues, is defined as in the Expression 5.1:

$$F_i = \frac{\left(\lambda_2^i - \lambda_3^i\right)}{\lambda_1^i} \tag{5.1}$$

  On its part, the eigen-entropy E_i of a point can be computed as in the Expression 5.2:

$$E_i = -\sum_{n=1}^{3} \lambda_n \cdot \log \lambda_n \tag{5.2}$$

  A decision on which is considered as the point neighbourhood should be then made. Some authors [158] suggest cylindrical shapes, using a preset amount of meters to create the neighbourhood. This is indeed useful when

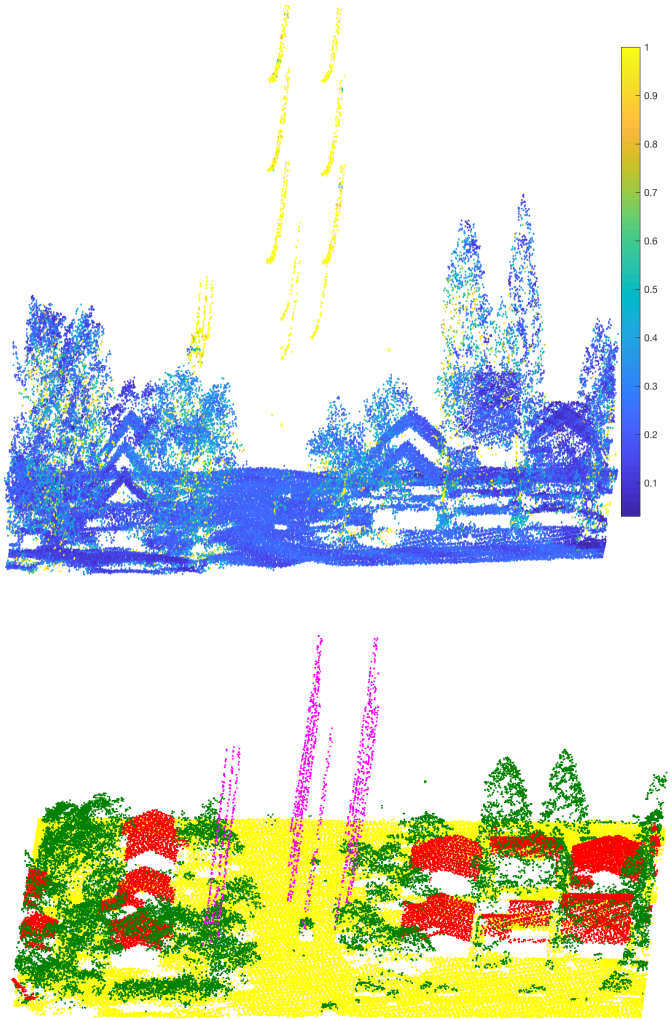


Figure 5.4: Normalized punctual eigenentrophy (top) versus the classification (bottom) for an urban point cloud benchmark. Bluish colors representing low values eigenentropy are dominant between ground points and human constructions on a minor scale. On the contrary, trees and wires have higher values of entropy.

the whole cloud is analyzed, but patches are composed only of points with minimum local height, so a 3D volume-based neighbourhood loses sense. Another option is to consider the whole patch as the point neighbourhood. However, as the patch area grows, the points inside the patch have less

information and hence the information obtained with the covariance features become less relevant. The last strategy considered was to use the closest N points on the patch as the neighbourhood of the point. Eigenvalues obtained in this manner are dependent on the distance between the selected points, but inside the patches a minimum density of 1 point per square meter is guaranteed and higher for really flat areas, so this effect could become interesting for detection of regular patches representing areas with obvious human intervention. Due to this, the closest points strategy was selected, using $N = 8$ points.

To obtain the planarity and eigen-entropy features per patch, the averages of all punctual values for these features are computed. To save computational costs, these features are only computed for patches with an area in the range expected for a building. High planarity values in a patch are expected to be a cue of human structures. On the contrary, low values of entropy, as it can be seen in Figure 5.4, are expected to be a cue for detection of ground patches.

- **Ratio of presence of safe non-ground points**

  To calculate this feature, safe non-ground points should be, from those composing the original point cloud, the ones whose height P_z are higher than the value of H for their associated cell plus a fixed amount in meters. In this work, that amount is preset to 0.5 m. For a given grid cell, the ratio of presence of safe non-ground points is defined as the number of such points divided by the total number of points in the cell.

  This feature is thought to help differentiate between rooftops and cultivation terraces: it is more probable for a patch to represent terrain if there are many points representing other objects over it, something not so common in a rooftop.

5.2.3 Decision tree for patch classification

Once the features are extracted, the classification of patches as terrain is computed by using the decision tree in Figure 5.5.

The area and amplitude features enable taking first and coarse decisions to quickly categorize ground patches in an obvious manner. The height step ratios RH_n split the remaining patches between those which contain no salient points with respect to the surrounding areas, those which are fully salient and those which contain salient and non-salient points.

Patches in which all the points are salient are categorized as representing non-ground areas. Optionally, they could be divided into rooftop patches and patches representing other elements by using the planarity feature. Patches without any salient point are categorized as ground. Finally, mixed patches are categorized using planarity, eigen-entropy and clear non-ground presence ratio: low planarity, low entropy or high non-ground ratio are strong indicators of a ground area, so a patch presenting any of those is classified as ground.

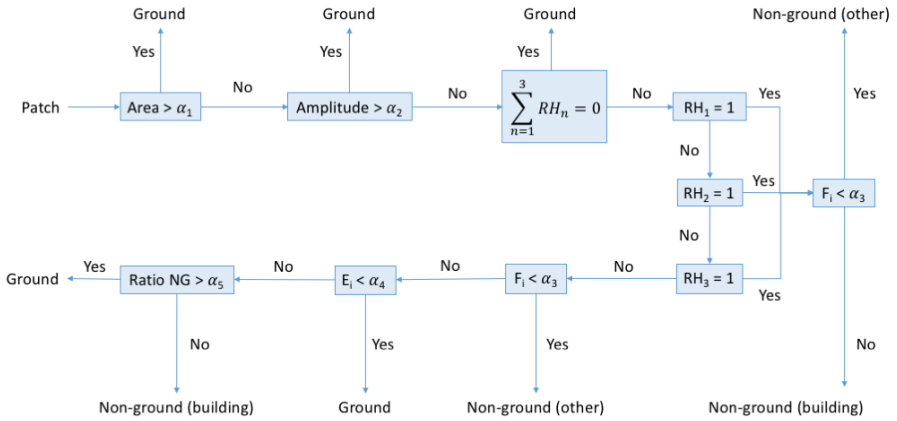


Figure 5.5: Decision tree for the terrain patch classification. Different features are calculated for each patch and sequentially tested to determine whether it represents ground or not.

5.2.4 Final point classification

The final stage transforms the patch classification into point classification. All points with a relative height lower than 0.5 m. and associated with cells covered by ground patches are considered belonging to the ground (e.g. low grass). The rest of the points in the cloud become non-ground points.

Optionally, it is possible to classify as building any point whose associated cell is covered by a non-ground patch with area larger than 15 m^2 (the approximate area of a room) and high planarity. However, this class is not considered for validation in this study.

5.3 Test and practical results

In this section, the tests conducted to validate the behaviour of the proposed methodology are described in detail, along with the configuration settings and a description of the test scenarios. All the experimentation has been conducted in Matlab 2018b.

5.3.1 About the experimental data

Two different benchmark point clouds have been used to test this scenario.

The first one is the well-known ISPRS commission Working Group II / 4 benchmark on urban classification, 3D Building Reconstruction and Semantic

Labeling. This benchmark is composed of several scenarios containing categorized point clouds and RGB texture information. As this work deals with 3D semantic labeling, the chosen scenario for the test is the one representing the city of Vaihingen an der Enz (Germany). For this scenario, the ISPRS commission maintains a record of several algorithms that have used the cloud for validation tasks and their performing results [159].

The cloud contains around 411000 points, distributed non-uniformly around an area of 400×400 m^2 with an average spatial resolution of 4 points per m^2. It was first introduced in the work of Niemeyer, Rottensteiner, and Soergel [160]. The cloud includes a ground truth in which nine classes are considered: *impervious surface*, *low vegetation/grass*, *shrub*, *tree*, *car*, *fence*, *facade*, *roof* and *power line*. As this work focuses only on the ground classification, the nine classes have been reconverted into only two: *ground*, which includes the former impervious and grass classes; and *non-ground*, which is composed of the remaining seven classes.

The second one is the *"dataset 1"* shared in the work of Gu, Wang, and Xie [149], and downloaded from the OpenTopography website [161]. The cloud, which represents an urban area, has a spatial resolution of around 13 points per m^2. It contains a ground truth that is composed of four classes: ground, building, tree, and powerline. For the means of this study, the latter three are comprised into the *non-ground* group.

The reported results of all the reference algorithms have been adapted to this binary classification for a fair comparison with our proposal.

5.3.2 Configuration settings

The decision tree presented in this study depends on five thresholds α_1 , α_2, α_3, α_4, α_5 related respectively to area, amplitude, planarity, eigen-entropy and non-ground ratio. For replication purposes, the configuration settings of all the parameters, which are the same for both clouds, are shown in Table 5.1.

Parameter	Setting type	Value	Range	Step
Area - α_1	Semantic	10000 (m^2)	-	-
Amplitude - α_2	Semantic	15 (m)	0.1-0.9	0.1
Planarity - α_3	Mono-objective optimization	0.5	-	-
Eigen-entropy - α_4	Semantic	0	-	-
Non-ground ratio - α_5	Mono-objective optimization	1.5	1-20	0.5

Table 5.1: Configuration settings for the system. Range and step of tested parameters are offered for those parameters which were evaluated via mono-objective optimization.

5.3.3 Quality assessment and discussion

To validate the behavior of the proposed pipeline, a test has been performed in which the proposed patch decision tree pipeline is run against the two scenarios described before for ground classification. From this classification and the provided ground truths of each scenario, four quality measures have been computed per class: *recall*, *precision*, *F1 score* and *IoU score*. Details on how to compute these scores are provided in the Annex I.

For each scenario, reference methods from the literature have also been selected for comparison. Seven algorithms from the list provided by the ISPRS [159] have been used as reference methods for the Vaihingen benchmark: Niemeyer et al. [150], Blomley et al., Wang et al. [154],Winiwarter and Mandlburger [151] , Cvirn et al., Zhao, Pang, and Wang [152], and Yang et al. [153]. For all of them, a reference paper or at least a short memory on how the method works is provided, which allows us to classify them according to the amount of information they used:

- **Only punctual information:** Niemeyer et al., Blomley et al., Wang et al., Winiwarter et al.
- **Punctual and imagery information:** Cvirn et al., Zhao et al., Yang et al.

	Winiwarter		**Yang**		**Zhao**		**Cvirn**	
	G	N	G	N	G	N	G	N
G	188310	12366	194847	5829	198001	2675	191577	9099
N	11002	200064	13543	197523	4776	206290	13006	198060
Rec.	0.938	0.948	0.971	0.936	0.987	0.977	0.955	0.938
Prec.	0.945	0.942	0.935	0.971	0.976	0.987	0.936	0.956
F1	0.942	0.945	0.953	0.953	0.982	0.964	0.945	0.947
IoU	0.890	0.895	0.910	0.911	0.964	0.965	0.897	0.900
	Patch tree		**Blomley**		**Niemeyer**		**Wang**	
	G	**N**	**G**	**N**	**G**	**N**	**G**	**N**
G	191726	8950	164128	36548	180411	20265	190491	10185
N	14211	196835	5743	205323	6410	204656	18609	192458
Rec.	0.955	0.933	0.818	0.973	0.899	0.969	0.949	0.912
Prec.	0.931	0.957	0.966	0.849	0.966	0.910	0.911	0.950
F1	0.943	0.944	0.886	0.907	0.931	0.939	0.929	0.930
IoU	0.892	0.895	0.795	0.829	0.871	0.885	0.868	0.869

Table 5.2: Confusion matrices and quality scores for the proposed methodology and seven reference methods against the Vaihingen benchmark. G and N stands for ground and non ground, respectively.

The patch decision tree falls into the category of algorithms that use only punctual information as input. The final results obtained from the classification

for the Vaihingen scenario can be seen in Figure 5.6 and their comparison with the reference methods can be seen in Table 5.2.

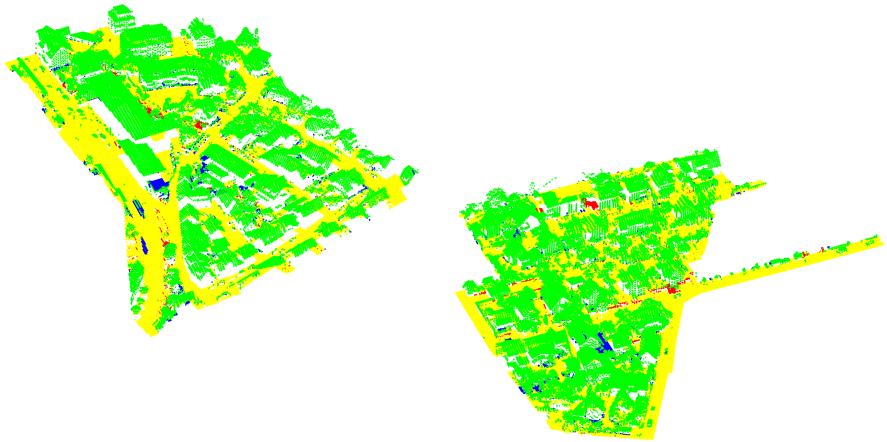


Figure 5.6: Obtained results using the patch decision tree over the Vaihingen point cloud dataset. Yellow and green stands for correct ground and non-ground prediction. Blue represents incorrect ground prediction, and red means incorrect non-ground prediction.

Regarding the second benchmark, the only reference method is the kernel-based MKSRC proposal from Gu, Wang, and Xie [149], for which a Matlab implementation is available. The results of the test against this scenario are shown in Table 5.3.

	MKSRC		**Patch Tree**	
	Ground	Non-ground	Ground	Non-ground
Ground	55175	1941	56995	121
Non-ground	1557	43075	1725	42907
Recall.	0.966	0.965	0.997	0.961
Precision	0.973	0.957	0.971	0.997
F1-Score	0.969	0.961	0.984	0.979
IoU	0.940	0.924	0.969	0.959

Table 5.3: Quality assessment of the algorithm applied over the "dataset 1" point cloud. The "MKSRC" methodology from Gu et al. is used as a reference method.

From the given results it is observed that the proposed patch decision tree is a promising solution for classification of ground elements in LiDAR point clouds. A completeness of 95.5% and 99.7% in ground classification have been respectively achieved for each benchmark, and the quality scores reflect that

the method is competitive when compared with all the solutions based only on punctual information. This is best explained in Figure 5.7.

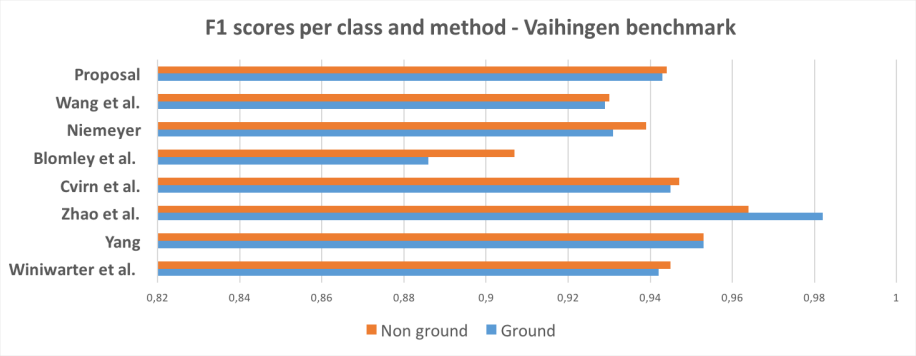


Figure 5.7: A F1-score comparison between different ground classification algorithms over the Vaihingen benchmark.

By looking at the F1 score achieved by each method for ground and non ground classification, it is clear that the patch decision tree works better than the methods of Niemeyer, Wang and Blomley and quite similar to the Winiwarter et al. method in the Vaihingen scenario. (94.3 % versus 94.2 % for ground classification, but 94.4% versus 94.5% for non-ground classification). It is also better than the MKSRC algorithm in the second benchmark (98.4% versus 96.9% in ground, which indicates that the algorithm is robust enough to deal with different point cloud densities and scenarios.

However, the solutions using the extra RGB input still perform better than our proposal. The method of Cvirn et al. performs slightly better than the patch decision tree (F1 scores of 94.3% versus 94.5% for ground classification, 94.4% versus 94.7% for non-ground), and the deep neural network proposals of Zhao et al. and Yang et al. are clearly superior. This opens an interesting question on whether the RGB extra input could make our schema better and how to apply it if possible. However, it should not be forgotten that for many tasks, like inspection and management of facilities or creation of DEMs and 3D city models, the acquisition of the data is the heaviest process in terms of time, space and economical costs, and many companies may rely solely on a unique source of data, generally point clouds. In these cases, the patch decision tree proposal can add value as it can accurately classify the ground using only the point cloud as input.

5.4 From patches to DEMs of progressive detail

One of the main applications of the methodology proposed in this chapter for ground filtering in LiDAR is the generation of *Digital Elevation Models* (DEM). Such DEMs are derivative products of high definition, with spatial resolutions

of few centimeters and very large extents. These data can be saved in vector or raster formats and be used for geographic applications and simulations. Additionally, they can be incorporated into 3D visualizations in the context of scenarios and virtual globes.

Loading highly detailed DEMs for visualization purposes comes with issues regarding computation, memory and network costs [33]. This problem is usually addressed with the generation of multiple representations in progressive level of detail of the elevation model [22]. Such representations allow to keep controlled the above-mentioned requirements for any point of view, depending only on a LoD test [76] which acts on a point of view change.

Three lines of work are predominant for LoD generation in DEM. The first one is based on TIN, which have been already introduced in the related work section. TIN have been used to create multi-resolution representation by several authors [162, 163, 164] for different mesh sizes. However, as the proposed *Patch Decision Tree* does not generate a TIN structure for filtering, it seems unpractical for the scope of this work. A second line is the generation of *Triangular Regular Networks* (TRN) [165, 166, 167], a mesh type in which all vertices are aligned in a regular grid. Techniques of this type are easier to implement but lack of the flexibility of TINs for representing changes in the elevation. An attempt to mix the advantages of both methods comes with the third line of work: the *Right Triangulated Irregular Networks* (RTIN). In RTIN, the vertices are aligned following a regular grid structure, as well as in TRN, but some triangles are removed from the meshes when unnecessary (e.g: no changes in the elevation). However, this kind of meshes offer no advantage when used as raster information in GPU [34].

In this section, a new methodology for the generation of multi-resolution TRN-based DEMs from terrain points is exposed. It looks for minimizing the visual differences between each representation. The method, which has been already introduced in [34, 130], estimates the height values of each vertex in a TRN of a resolution given by the user. To do so, it relies on solving a linear system of equations which relate each input point with triangle vertices on the output mesh.

The method starts by identifying a 2D triangle T in the mesh which envelopes a given input point P. Suppose a TRN is desired with a resolution $[m, n]$ in which the system coordinate origin is centered in (X_0, Y_0). In this supposed mesh, the minimum difference in the X and Y coordinates between two adjacent vertices can be expressed as Δ_X and Δ_Y, respectively. If the coordinates of the input point (P_x, P_y) are also related to the origin (X_0, Y_0), it is straightforward to find the mesh indices (X_G, Y_G) of the triangle T. It can be done with the following expressions:

$$X_G = \frac{(P_x - X_0) \pmod{\Delta_X}}{\Delta_X}$$

$$Y_G = \frac{(P_Y - Y_0) \pmod{\Delta_Y}}{\Delta_Y}$$

These indices are shared by two triangles in the mesh. Finding the correct triangle is done by applying the test $X_G > Y_G$. The superior triangle is selected when the test is passed, and the inferior triangle is selected otherwise. With T properly identified, the barycentric coordinates of P in the triangle T, $\lambda_P = (\lambda_i, \lambda_j, \lambda_k$ can be computed in linear time [168].

Having this in mind, the goal is to find the height coordinates Z for all the vertices of the mesh so the height differences are minimized with respect of the input model. Going back to the P example, a punctual error function E_p which takes into account the point height P_Z, the barycentric coordinates of the point λP and the vertex heights Z_i, Z_j, Z_k for each vertex in the triangle T can be expressed as:

$$E_p = P_z - (\lambda_i \cdot Z_i + \lambda_j \cdot Z_j + \lambda_k \cdot Z_k)$$

When applied to all the input points, this punctual function can be transformed into a global error function E expressed as the average of all the vertex errors. Such errors can be obtained via linear system of equations:

$$E = \mu(||A \cdot Z - B||)$$

where A is a large and sparse matrix with a number of rows equal to the number of input points and a number of columns similar to the vertices in the output TRN, $m * n$. In each row of the matrix A, the barycentric coordinates of its related point are included in the columns which refer to the involved vertices. On its part, B is a vector which includes all the point heights P_Z.

Finding a solution for Z which minimizes such an error can be done by using different methods, from which the least squares based *LSQR* algorithm [169] is recommended. LSQR admits non-squared matrices and it is particularly efficient for sparse matrices. By using LSQR, an initial approximation to Z is found via gradient descent. However, it is possible that most of the global error obtained is concentrated into few vertices with high vertical errors, which is also undesirable. This leads to the proposal of a novel iterative stage for refining the LSQR solution.

For each iteration, the contributions Q of all the mesh vertices are computed as $Q = A^T \cdot E$. For all vertices Z_i which have a contribution $Q_i > 0$, a local change of Z_i is proposed following the rule $Z'_i = Z_i - \alpha \cdot Q_i$. The local change is applied to Z_i if and only if it reduces the global mesh error. α stands for an adjustable factor which ranges between 0 and 1. However, it was observed during the experimentation that $\alpha = 0.7$ offers good results. Iterations must continue while $E - E' > \varepsilon$, being ε an adjustable tolerance and E' the new mesh global error obtained after applying the local changes. This iterative stage generates the first LoD representation for the elevation model. Subsequent, coarser representations can be obtained using this LoD representations to speed up the process, as TRN vertices can be considered as points for the method input.

There is a final issue to be considered before using this technique with LiDAR inputs: how A grows in size against potentially large amounts of input points

and mesh resolutions. As it is required to solve a linear system of equations and then run the refining stage, memory consumption and execution times increase exponentially when this methodology is applied on the whole point cloud at once. To tackle this drawback, a solution is to divide both the input and the mesh in individual tiles of reduced size and solved each tile independently. In this manner, parallel computing keeps the execution times controlled. It should be taken into account that each tile should have a frame of a number f of vertices width from adjacent tiles. The main reason for this is to ensure that the tile results are invariant with respect to the results for the full input. Experimentally, it was observed that a factor $f = 10$ is enough to guarantee it.

5.5 Conclusions

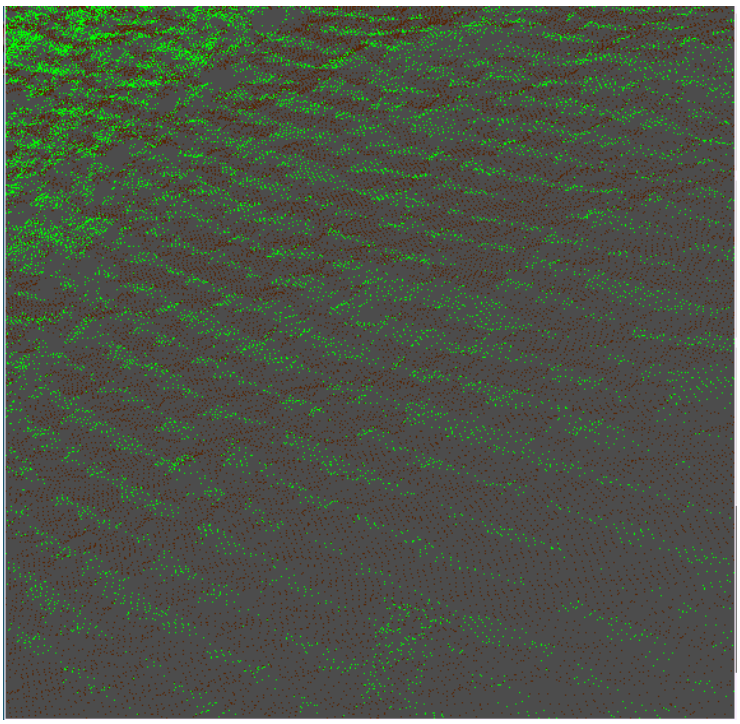


Figure 5.8: Linear artifacts (green) detected after using the algorithm in a mountainous scenario with low vegetation. Slope information could contribute to removing such misclassification.

In this study, a novel algorithm has been proposed for separation between ground and non-ground points based on patches: clusters of minimum local height. The algorithm gathers different features of geometrical, covariance or multiscale analysis origin for each patch and assigns each patch a tag using a

decision tree. Patch classification is then converted into the final cloud classification.

The methodology validation has been conducted against two well-known benchmarks for which reference results from state-of-the-art algorithms are available. The Patch Decision Tree has been competitive in both scenarios, with results of 95.5% in completeness and 94.3% in the F1 quality score for the Vaihingen an der Enz cloud; and 99.7% in completeness with 98.4% in F1-score for the *dataset1* of Gu et al. These results are better than the ones obtained for the reference methods that consider the point cloud as the sole input. Moreover, the terrain classification obtained this manner can be used as input for the generation of regular multi-resolution digital elevation models.

However, the algorithm still has some drawbacks to be solved and three main lines of potential improvement have been identified in order to improve the accuracy of the algorithm. One of them is the integration of other sources of information on the patch decision tree whenever they are available. A second one is to add slope cues in the last step to correct minor artifacts that could happen in the final stage when defining which points from each cell are considered non-ground. The current condition is a simple height rule, which is useful for most point clouds, but could be not good enough for really complex terrain, as seen in Figure 8. By adding slope information to the last step, it is expected that most of these misclassifications can be removed. The third possible line is a further research on how to differentiate subclasses related to ground (road, low grass, bare terrain, etc).

Chapter 6

Detection of urban objects in vehicle-borne point clouds

In the previous chapter about ground, it was exposed the need of a proper ground classification in LiDAR to ease several tasks, which include urban-related topics. Those topics include simulations and 3D modeling [116], autonomous driving and pedestrian safety [128] or electrical inspections [126], among others. Many of these tasks can also benefit of algorithms that identify several types of urban objects in LiDAR clouds.

At this point, it is already known that LiDAR information can be given in full waveform or discrete returns, and can include extra information regarding intensity or color [87]. Additionally, it could be differentiated according to the device in which the sensor has been installed: *terrestrial* (TLS) or *mobile* (MLS) borne (called generically *vehicle-borne* in this document), and *airborne* (ALS). Airborne point clouds are coarser but more regular in terms of their point density, while vehicle-borne point clouds are much more dense but the density decays with the distance to the acquisition vehicle [170]. The higher density makes the latter more suitable to segment urban scenes.

From vehicle-borne LiDAR point clouds, multiple urban object classes have been successfully extracted by numerous researchers, including roads, natural ground, poles, traffic signs, buildings and façades, cars, pedestrians, and trees [171]. However, it is generally needed to combine different techniques for multi-class classification of these objects and both accuracy and time consumption are normally affected. This fact makes multiclass urban classification an open problem prone to new proposals and improvements.

In this chapter, the novel unsupervised methodology P4UCC for the classification of five classes of urban objects in LiDAR point clouds is introduced. It initially extracts the ground areas from minimum height point clusters and the estimated sensor position and inclination. After that, hierarchical clustering

and local covariance-based feature descriptors are used to extract car and pole objects from the clouds. Finally, a recursive algorithm extracts vertical planes from the remaining clusters and decides whether the group represents building or vegetation. A quality assessment is performed to validate the methodology over the Street3D point cloud benchmark, demonstrating its suitability for urban classification.

6.1 Related work

The topic of urban classification in LiDAR point clouds has been previously explored by many authors and can be categorized by the features to be extracted, the classes the work covered, as in this case, whether they made use of supervised or unsupervised learning techniques.

Supervised machine learning is the most frequent option and, inside it, multiple techniques have been explored. The combination of weak classifiers and decision trees with boosting algorithms is used in the work of Gao and Li [172]. *Random Forests* (RF) were proposed for similar tasks in urban classification by Chehata, Guo, and Mallet [173]. in 2009. Other authors took advantage of RF as well, e.g. Fukano and Masuda [174], who combined them with an analysis of laser strips to detect poles; Weinmann et al. [175], who used them along with *Conditional Random Fields* (CRF) and studied appropriate neighbourhoods for the generation of features to train; or Wang et al. [176], who analysed 2D projections of the cloud with RF and then established pixel comparisons and voting frameworks for classification.

Another common approach is super-voxelization, which can be applied to both supervised and non-supervised techniques. This technique divides the cloud in voxels and then grouped them in base to the available features by applying the *Voxel Cloud Connectivity Segmentation* (VCCS) [177] algorithm. VCCS results have been used to train RF [178] or CRF [150] classifiers for multi-class urban classification of point clouds. Other alternative approaches were the kernel-based machines, with the work of Gu, Wang, and Xie [149] as a representative example; or the votation frameworks, as the one proposed by Velizhev, Shapovalov, and Schindler [179] for car and pole shape detection.

Finally, *Neural Networks* (NN) are also heavily considered for automated urban classification from point clouds. This branch divides into the architectures of convolutional neural networks for pixel and voxel classification, in which the works of Zhao, Pang, and Wang [152] and Yang et al. [153] should be highlighted, and the point-oriented neural networks, in which a prominent reference can be found in PointNet [180]. This network allows to determine whether a point cloud represents a given object. An evolution of this network, PointNet++, is also introduced by Qi et al. [181] and then adapted by Winiwarter et al. Winiwarter and Mandlburger [151] for urban object classification. Additionally, networks as the DNNSP from Wang et al. [154] enabled direct point classification instead of shape recognition from a point set. Zaboli et al. [182] compared the behaviour of a NN architecture against other methodologies like RF, support

vector machines and perceptrons, demonstrating close results between RF and NN in a benchmark dataset.

Unsupervised techniques are less explored but there are still some relevant results in the state-of-the-art. Rodríguez-Cuenca et al. [183] used hierarchical clustering and anomaly detection algorithms to differentiate poles and trees in urban point clouds. El-Halawany and Lichti [184] calculated 2D density of neighboring points and applied RANSAC line fitting for pole detection, and Sirmacek and Lindenbergh [185] presented a probabilistic algorithm for filtering of vegetation points in urban clouds. In this context, the proposal exposed in this chapter is an attempt to perform multi-class classification by using unsupervised techniques, where the above-mentioned works discriminated the input data in only two groups.

6.2 The Progressive 4-staged Urban Cloud classifier

The *Progressive 4-staged Urban Cloud Classifier* (P4UCC) is described in this section to solve multiclass urban classification. It is a four-staged pipeline that aims to progressively classify ground, car, pole, building, and vegetation points in vehicle-borne point clouds, using only geometric information. Its general architecture can be seen in Figure 6.1.

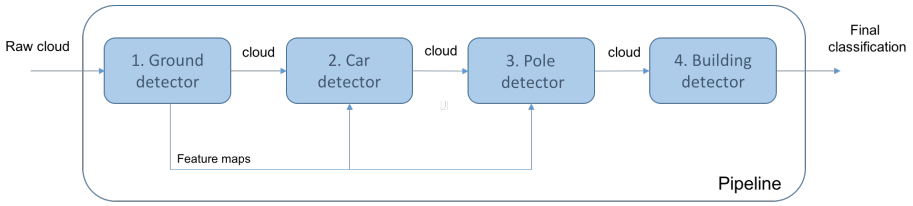


Figure 6.1: Multi-stage architecture of the P4UCC method.

Each stage aims to identify only a certain class. The first one is dedicated to ground point detection and also generates pixel-wise features which are useful for subsequent stages. Stage 2 looks for the detection of car points, Stage 3 filters pole points and Stage 4 applies a recursive algorithm for extraction of wall planes. The presence of wall planes helps to determine the class of the remaining points between building (those in a wall plane) and vegetation (the rest). The details about each stage can be seen in the following subsections.

6.2.1 Ground detector

Ground detection starts by dividing the cloud horizontally in a grid with squared cells of 1 m^2. Cell features are computed using the points in each cell: minimum height, H [157]; point accumulation, A; and geometric features such as omnivariance, O; and eigensum, E [158]. Minimum height is used to select the

ground candidates: all points in the cloud with height lower than $H_{ij} + \epsilon_1$. Those points are then grouped using a hierarchical approach [123], euclidean distance and cutoff of ϵ_2.

Cluster centroids and their inclination angles with respect to the horizontal plane of the LiDAR sensor are then computed. In case the sensor position is unknown and the point cloud has been taken with a TLS system, the mean of the point positions for the cell with the maximum A_{ij} can be used as approximation. This can be seen in Figure 6.2. Such reasoning can be done only due to the nature of terrestrial point clouds, which are dense in the proximities of the sensor position and progressively coarser in relation with the square of the distance [170]. In airborne point clouds this is not possible due to all points being the flight height far from the sensor as a minimum. For MLS systems, it is also not possible as the final point cloud is a combination of all the acquisitions of the LiDAR sensor during the vehicle movement. Hence, the real sensor positions and timestamps are needed for them. The point clusters deemed ground are:

- Clusters with an inclination angle, α, lower than the one of the vehicle in which the sensor is mounted.
- The cluster with the largest amount of points.

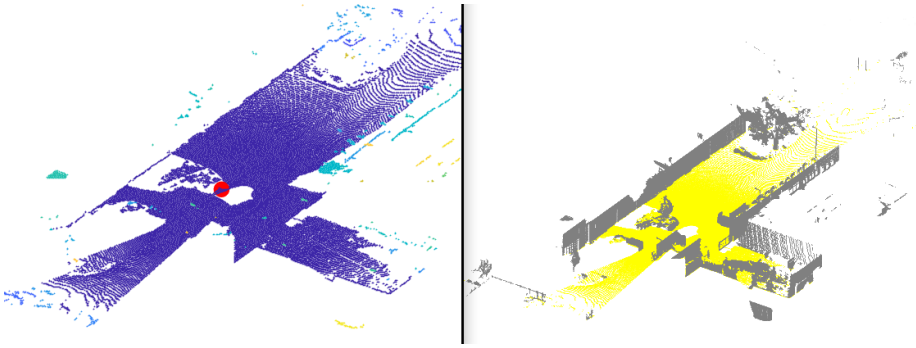


Figure 6.2: Minimum height clusters and sensor position (left figure, sensor in red) vs final ground areas (right, yellow).

The use of sensor position and inclination thus allows to heavily simplify the decision tree for ground patch prediction presented in Chapter 5 in case vehicle-borne point clouds are being used.

6.2.2 Car detector

Points from cells with $E_{ij} < \epsilon_3$ are considered for car detection. These points are grouped using the same hierarchical clusters of Stage 1. An area descriptor is calculated as $\Delta X \cdot \Delta Y$. A planarity descriptor [152] is also computed per cluster. A cluster is considered to represent a car when its area is shorter than

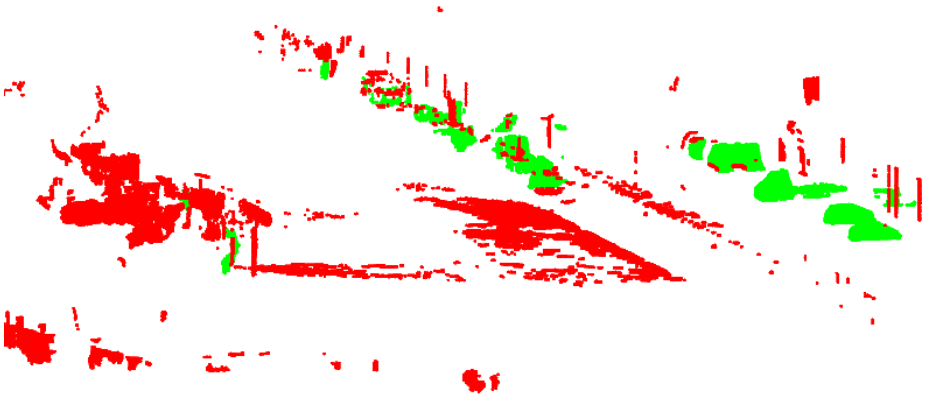


Figure 6.3: Car clusters (green) chosen from the points considered for car detection.

40 m^2 and the planarity is greater than ϵ_4. Examples of this can be seen in Figure 6.3.

6.2.3 Pole detector

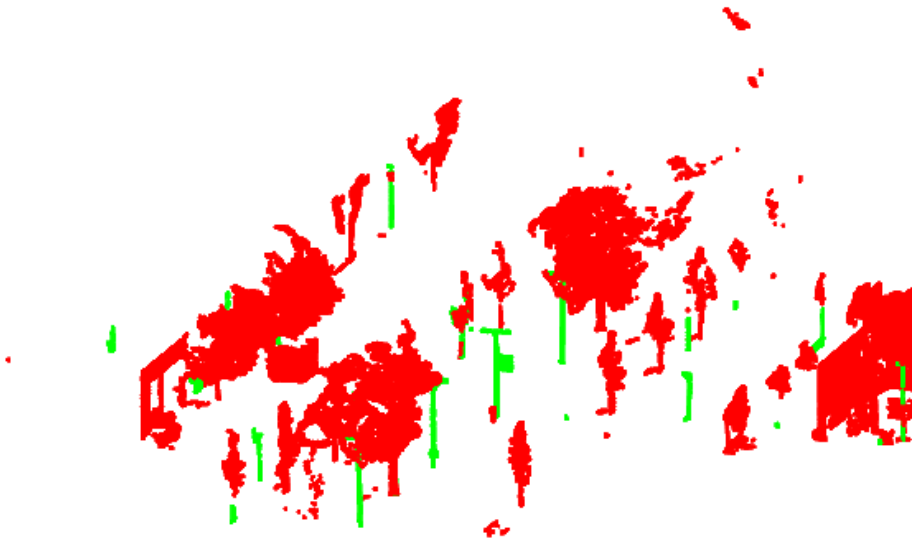


Figure 6.4: Pole clusters (green) chosen from a set of the points considered for pole detection.

Points from cells with $O_{ij} < \epsilon_5$ are considered for pole detection. They are clustered, again, using the same hierarchical clustering algorithm from Stages 1

and 2 with the same conditions.

Per cluster, the following features are the computed: curvature change [158], dominant component, minimum relative height and deviation of the XY component values. Point clusters are classified as poles, as shown in Figure 6.4, when the dominant component is Z and the following conditions are fulfilled:

- The curvature change is lower than ϵ_6.
- The relative minimum height is lower than 1 m.
- The deviation of the XY point coordinates is lower than 1 in both X and Y components.

6.2.4 Building detector

The remaining points are grouped as well. A recursive algorithm for the extraction of vertical planes based in MLESAC [119] is applied to each cluster. The algorithm progressively looks for planes with an error margin of ϵ_7. Based on the normal of the plane, the plane points are selected as vertical or not, and then are removed from the process. The procedure stops when planes can no longer be extracted.

Clusters with a majority of points in a vertical plane and with $\Delta Z > 2.5$ m. are considered to represent buildings. Otherwise, they are classified as vegetation.

6.3 Tests and experimental results

In this section the validation process for the pipeline is exposed. It includes details about the testing dataset, the configuration settings and the quality assessment performed over the proposed algorithms. All the experimentation has been conducted in Matlab 2018b.

6.3.1 About the experimental data

In this study, the benchmark *Street3D* of the SHREC 2020 Conference, track 3 [186] was used. It is composed of 80 point clouds which cover grids of 50x50 m. with sizes between 2 and 5 million points. They were acquired by the Cyclomedia Technology company using a Velodyne HDL-32E sensor mounted in a car. The point cloud set represents urban scenarios including objects of the five above-mentioned classes, plus extra *undefined* points which are not considered for evaluation. A ground truth for each cloud was manually generated specifically for the conference track.

The point cloud set came divided into groups of training (60 files) and test (20 files) in order to allow the comparison of different solutions in the challenge that could make use of supervised machine learning approaches. This schema is kept for validation regardless of whether the pipeline does not use supervised

learning, so the results can be compared with the solutions of other competitors in the challenge.

In the test set, 68328274 meaningful points have been found, from which a 57.45% are classified as ground; a 23.50% are classified as building; a 0.83% of the points belong to the pole class, a 2.33% are classified as car and the final 15.89% as vegetation.

6.3.2 Configuration settings

The pipeline depends on seven thresholds $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5, \epsilon_6, \epsilon_7$ related respectively to height offset, clustering cutoff, eigensum, planarity, omnivariance, curvature change and maximum error point-plane. Additionally, an inclination angle, α should be estimated in absence of vehicle data.

Two parameter setting methods have been performed: *semantic* for those with a clear meaning, and a mono-objective optimization for the rest, using a range of possible values of the target parameter. For replication purposes, the configuration settings of all the parameters are shown in Table 6.1.

Name	Stage	Range	Step	Value
ϵ_1	1	Semantic		0.25 (m)
ϵ_2	1	Semantic		0.5 (m)
ϵ_1	1	Semantic		1.5 (^{o})
ϵ_3	2	0.25:3	0.25	1.25
ϵ_4	2	0.1:0.9	0.1	0.2
ϵ_5	3	0.05:0.5	0.05	0.2
ϵ_6	4	0.01:0.1	0.01	0.01
ϵ_7	4	0.05:0.2	0.05	0.05 (m)

Table 6.1: Relation of parameter settings.

6.3.3 Quality assessment and discussion

In order to validate the behaviour of the proposed pipeline, a test was conducted in which the pipeline is run against the 20 point clouds of the test dataset and their results are compared against the provided ground truth. A quality assessment is then performed, using an intraclass score: *intersection over union* (IoU), and two general scores: *overall accuracy* and mean of intraclass IoUs. In Annex I, details about the computation of those scores are provided.

P4UCC is compared against the result of four external methods: the PointNet ++ of Qi et al. [181], which serves as a reference due to it being well known and popular amongst the community, and three participants in the conference. Although the details of each competitor method can be found in the conference joint paper [186], a short summary of each one is included in the following lines for better readability and comparison of ideas:

- **Method1 (Duque-Arias et al.)**: It trains a 2D model from height, distance sensor-point and point normals using a variation of the neural network architecture U-Net. The mentioned features are previously converted into 2D using spherical projection. A final layer with a *k nearest neighbours* (KNN) classifier is applied to offer the final point predictions from the network output.

- **Method2 (Ku and Veltkamp)**: It voxelizes and reduces the point cloud in order to create a graph of fixed-size. A model is trained using a residual-graph attentional convolutional network for 5-class prediction of the graph. The final point prediction is performed from the network prediction via KNN classifier.

- **Method3 (Akadas and Gangisetty)**: It is an adaptation of the Randlanet deep learning architecture for semantic segmentation [187] so it considers the XYZ coordinates as point features. Points are subsampled in grids prior to training and final predictions are determined from the network output, again, by using KNN search.

	Pointnet++	**P4UCC**	**Method1**	**Method2**	**Method3**
Ground	0.9226	0.9646	0.9646	0.9357	0.9810
Building	0.7303	0.8435	0.8316	0.8523	0.9366
Pole	0.3274	0.4018	0.2752	0.2548	0.6179
Car	0.3512	0.6051	0.4993	0.4377	0.8392
Vegetation	0.7840	0.7975	0.7941	0.8352	0.9455
Overall acc.	0.9016	0.9413	0.9389	0.9310	0.9783
Mean IoU	0.6231	0.7225	0.6730	0.6631	0.8640

Table 6.2: Achieved results of the experiment: quality scores. Class scores are all IoU.

The obtained results, which can be seen in Table 6.2, are really promising. P4UCC outperforms the reference method PointNet++ for the prediction of all the classes. It also achieves the second best general performance of all the included algorithms, with an overall accuracy of 94.13% and a mean IoU of 76.06%, superior to the neural network approaches and only behind the deep learning approach. Analyzing the behaviour, the algorithm performs the second best for ground, pole and car classes, and third best for vegetation and building class. This exposes the competitiveness of P4UCC for multiclass classification. Figure 6.5 shows the output of the process for one of the point clouds in the benchmark.

However, there are some flaws in the methodology. Low IoUs of 40.18% and 60.51% have been achieved for pole and car classes due to an appreciable amount of false positives in classes with a low amount of points. False car predictions have been found mostly in points belonging to the ground, which possibly indicates that the criteria to decide whether a point is ground or not

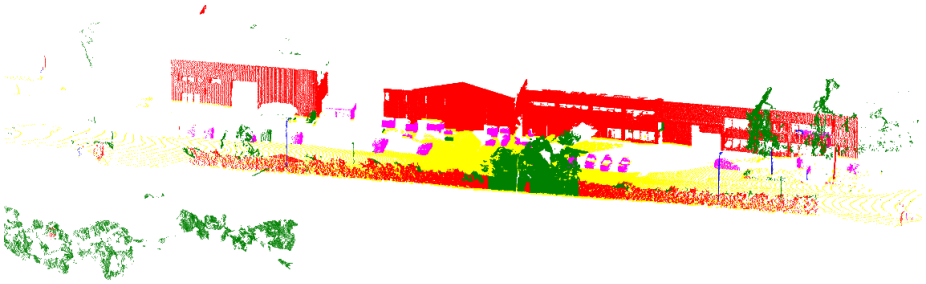


Figure 6.5: Final result in 5D4KVPG4 cloud. Building, pole, vegetation, car and ground are represented with red, blue, green, magenta and yellow colors, respectively.

should be improved in the proximities of a car. False pole predictions have been found in building points, and vice versa, which probably indicates issues regarding the clustering process. In areas with less point density, a vertical cluster with pole-like descriptors can be formed out of building points. The opposite case has been observed in places with high point densities and close proximity between the pole and a wall, resulting in a single cluster containing the pole and the wall and considered building.

Another interesting observation comes from the achieved results of the competitor methods. The two neural network approaches tend to generalize really well one or two of the classes while giving poor results on new examples of the remaining three. This is, for example, the case of the Method 1, which specializes in ground detection and is not quite recomendable for the detection of poles and cars. On the contrary, Method 2 segments better buildings and trees. In the progressive unsupervised detection presented in this chapter, the initial segmentation of one class favours the latter detection of the remaining others, thus gaining in regularity at the expense of the individual class accuracy. As for the deep learning approach, it outperforms both the reference method and all the proposals, including P4UCC, for classification of all the point categories. However, its results for pole prediction are still prone to improvement, which opens a question on whether a progressive pipeline of two-class deep learners can benefit the detection of the most problematic urban objects.

6.4 Conclusions and future work

In this chapter, a novel pipeline has been developed in this work to classify five relevant classes, ground, building, tree, car and pole, in urban vehicle-borne LiDAR point clouds. The pipeline uses only position information and starts by extracting the ground points, defining clusters of points with minimum local height and calculating the inclination angle with respect to the estimated sensor

position. Hierarchical clustering is then utilized to separate clusters of non-ground points and local covariance descriptors computed for the identification of cars and poles. The final step is a MLESAC-based recursive algorithm that extracts vertical planes and determines the presence of buildings and trees from the remaining points.

The results of a quality assessment performed over a benchmark dataset demonstrate promising results. The proposed P4UCC pipeline outperforms the reference method PointNet++ for urban classification and achieves better overall accuracy (94.13%) and mean IoU (72.25%) than other two proposed neural networks, being the second best proposal for the benchmark data only to a deep learning approach. The conclusion extracted from the results of the validation process is that the main advantage of P4UCC over those methods is precisely to progressively extract the classes with one-class detectors instead of using a single architecture for classifying the whole cloud.

Additionally, a simple manner to heavily improve the classification results of P4UCC is to integrate the strengths of such supervised methodologies in the current pipeline schema. Future lines of work will look for the inclusion of new features and models to improve each individual detector, combining the current proposal with some of the ideas exposed in the supervised algorithms.

Chapter 7

Modelling power corridors from LIDAR data

Nowadays, the world population has an always increasing dependence on energy. This makes the management of power lines an issue of special importance for power companies. Service could be interrupted due to damages caused by the fall of objects, generally trees, over the power line. In other cases, the change of tension suffered by a pylon due to damaged conductors can make collapse an entire section of the line. Moreover, forest fires could start as a result of these events, producing another kind of inconveniences on the people. That was the case of the fires of the island of Gran Canaria in mid-August of 2019, in which the contact of a power line with nearby trees started a fire that results in more than 9200 hectares of pine forest and crops burned and 10000 people evacuated from their homes.

For the power line inspection and management tasks, the use of LIDAR technology has become very frequent. In this chapter, a new pipeline for the classification of power structures in discrete LIDAR point clouds, as well as the 3D modelling of pylons and wires, is proposed. The chapter is divided into two parts: an initial one who describes a methodology to find a power corridor in a LIDAR point cloud and make an initial classification the corridor points into wire and pylon points, which was firstly introduced in [157]; and a second one introduced in [126] which separates the corridor points into 5 more detailed categories: pylon, insulator strain, common conductor, shield conductor, and chain conductor. The conductor points are segmented into single entities and a 3D catenary model is generated for each of them. The pylon points are also segmented into individual pylon structures and a vectorial schema is generated from them.

Extensive testing with multiple point clouds to fit the parameters and comparisons against state-of-the-art techniques are also introduced in this chapter to validate the behaviour of the proposed technique.

7.1 Related work

A great amount of previous works deal with classification of power lines in LiDAR point clouds and characterization of the lines as vector data models. In the field of power lines, the papers focused on extraction of wire models and classification of the corridors. Different methodologies for classification of corridor scenes are given in works as the one of Liu et al. [188], which applied the Hough transform on a subset of the point cloud after conducting an analysis on skewness and kurtosis of the data. Jwa et al. [189] introduced the Voxel-based Piecewise Line Detector method. Kim and Sohn used an RF classifier to identify five classes of points, including wire and pylon [190], and then integrated it into a *Multiple Classifier System* [191]. Liang et al. [192] looked for the orientation of the power line and then clustered the candidate points in single lines. Ritter and Benger[193] used eigenvectors and tensor fields to reconstruct the different lines.

In the last five years, new proposals on classification of the corridors have appeared. This is the case of the work of Guo et al.[194], that used a Joint Boost classifier instead of an RF to identify the same categories. Zhu and Hyyppä [195] introduced an algorithm which uses height data to identify low-voltage lines in forested areas. Gu et al. [149] looked for heterogeneous features in the data and generated kernels from them. Those kernels are then used to classify points into 5 classes. Finally, Awrangjeb and Islam [196] defined an image mask to extract pylon points and then analyzed the topological relation between pylon candidates and wire lines to remove false positives. Other proposals are also available in an extensive survey done by Matikainen et al. [197]

Other group of works are focused not only on classifying but also on extracting models of the different conductors in the scene. On this line, the work of McLaughlin [198] takes into account elliptical neighbourhoods for each point and proposed to segment the points relying on the computation of the matrix of covariance of data. Then, local models for each wire are generated. Guo et al. [199] extended their previous work [194] to extract wire models by analyzing similarities and the distribution of points between pylons and then applying the RANSAC technique. Cheng et al. [200] partitioned the cloud in voxels and applied the *Autoclust* algorithm to cluster individual conductors. Each conductor is then modelled using a polynom. Finally, Guan et al. [201] extracted power infrastructures from vehicle-borne points and filtered them using the Hough transform and distance-based clustering. Models per each detected conductor are then generated as a horizontal line and a two-dimensional catenary equation.

All the mentioned studies only consider the classification of pylons and conductors without detecting any shorter elements inside of these classes. However, there are several subtypes of pylon in the real world, like suspension and anchor pylons, which differ in how they can manage forces and how the conductors are attached to them. The conductors can also be split into different subcategories: shield conductors, common phase conductors and chains. Moreover, a small piece of the pylon called *insulator strain*, which allow wires to hang from the pylon without losses of energy and it is easy to misclassify as a wire, should be

also detected, removed from the wire model and considered into a pylon model for inspection purposes. This is still an open problem, in which the work of Arastounia and Lichti [202] can be highlighted. It aims to detect those insulator strains, but it does detect them only on electrical substations. Instead, the proposal explored in this chapter look for those elements along the whole power line corridor.

7.2 Finding the power corridor

In this section, a methodology is described to accurately detect the power lines from the LIDAR data. It started by performing a statistical analysis on the data, in order to find patterns related to pylons and wires. Once patterns have been found, a two-staged pipeline is proposed, with an initial stage which performs an image-based selection of candidate areas, and a final stage which filters these areas to remove possible false positives. The statistical analysis and both stages of the initial pipeline are presented in detail in the following subsections.

7.2.1 Research dataset description

A dataset with 68 different point clouds have been used in this study. They have a number of points which ranges between 1.5 and 2.5 million per cloud and an average point density of 25.5 points per square meter. The dataset clouds represent different sections of a power corridor placed in Spain, whose total length is of 80 km. and its average width is of 80 m. They have been acquired using a Riegl VUX-1LR sensor and a IMAR IMU-FSAS-NG inertial unit, mounted in a helicopter. Scanning flights were parallel to the power line and around 300m. over the ground, with an average speed of 40 knots. This acquisition scheme allows to minimize the number of strips and, with a proper calibration of the system, no appreciable overlapping errors are found between them. Noise removal has been then applied in all the acquired point clouds.

The point data include power line points as well as points representing terrain, all kinds of vegetation, buildings, roads and water masses. A hand-made classification has been performed on the set and used as a ground-truth for pylon/wire classification. The files have been divided into three different sets:

- **Training set (20 files):** Used to conduct previous analyses on the data.
- **Validation set (23 files):** Used as a sandbox to adjust the parameters.
- **Test set (25 files):** Used to conduct the experimentation and obtain the final results.

7.2.2 Data analysis

Prior to the development of any method, it is important to analyze the available data to find what makes all the possible elements in the point cloud different

from each other. The focus is put on the three variables that are included in the minimum version of LAS point clouds: height, intensity and return value. The height value can be used in terms of distance between the point and the underlying ground to determine where a power line could be found. Power lines are supported by pylons with diverse shapes and heights. Most pylons for high voltage lines have heights between 10 and 55 meters [203]. That also implies that wires can be found at heights of at least 8 meters.

To find conclusions from the intensity of the return, maximum, minimum and mean intensity values have been extracted from the twenty files of the training set. The results are presented here as a resume in Table 7.1.

	Terrain	**Vegetation**	**Buildings**	**Pylons**	**Wires**
Min	35	37	44	35	0
Max	5843	5843	4774	4819	4752
Mean	3070	2995	2415	2050	1370
Std	91.12	203.8	506.9	60.56	34.88

Table 7.1: Resume of intensity analysis

From the analysis of the set can be extracted that all categories present similar minimum and maximum values, meaning that the category of a single point cannot be inferred from its intensity information. However, mean and standard deviation for the whole set of category points allow to make some differentiation. Particularly, groups for power line components have lower and less diverse returns than the rest of the categories. As an example, the ratio of intensities between the mean for wires and the global maximum is 0.23. For the case of pylons, the same ratio has a value of 0.34. This fact enables us to differentiate the two categories based on the mean intensity value of the cluster.

	Terrain	**Vegetation**	**Buildings**	**Pylons**	**Wires**
1 (mean)	82.66%	80.76%	75.07%	67.75%	90.73%
1 (std)	5.78	6.83	26.82	5.20	5.74
2 (mean)	14.75%	16.42%	14.13%	26.80%	8.59%
2 (std)	4.11	5.08	6.35	2.42	4.46
3 (mean)	2.22%	2.46%	2.96%	0.70%	0.69%
3 (std)	1.46	1.53	4.55	2.27	1.18
4 (mean)	0.32%	0.31%	2.96%	0.70%	0.07%
4(std)	0.26	0.25	8.88	0.80	0.17
5(mean)	0.04%	0.03%	2.90%	0.08%	0.01%
5(std)	91.12	203.8	506.9	60.56	34.88

Table 7.2: Return distributions per category in the training set. First column expresses the return number.

The third variable, the return number, cannot be applied to single points either. However, it is expected that in regions with more than a possible return,

the nearest point has a minor return value. Moreover, high human-made objects (like pylons or buildings) are expected to have a face full of first return points and other one with more points obtained from last returns, since the LIDAR device will be opposed only to a given side of the object. An analysis on the distribution of returns for the training set has been performed, and a summary can be read in Table 7.2.

From the summary it can be extracted that, for the pylon category, standard deviations from the mean of first and second returns are lower than in the rest of the groups. The percentage of first return points are quite low compared with the same percentage in other categories. These facts can be used to filter clusters of points that represent pylons.

Other remarkable fact is a high percentage of first return points in the wire category. Combined with the information about height and intensity, this could help identify clusters of points as wires.

7.2.3 Image-based classification

This stage proposes the generation of images in which every pixel represents a squared prism-shaped volume with infinite height whose section side has α meters length. The generated image should cover the whole point cloud regardless of the number of points in each pixel. The goal of those images is to separately differentiate wire and pylon areas of the electric power line from the rest of elements in the point cloud. To do so, different measurement images should be created from the available data, in this case, intensity and height values, and then combined.

One of these measurement images is the minimum height, H. There, every pixel H_{ij} is assigned the minimum height from the points contained in the $[X_{min}+(i-1)*\alpha+1, X_{min}+i*\alpha; Y_{min}+(j-1)*\alpha+1, Y_{min}+j*\alpha]$ XY region of the point cloud, from now on, its associate volume section. There, X_{min} and Y_{min} stands for the minimum X and Y values for the point cloud.

By using H, it is possible to compute an image of amplitude, A. Every pixel A_{ij} of the image is assigned the result of the Expression 7.1:

$$A_{ij} = \sum_{k=1}^{n} (Z_{ijk} - H_{ij}) \tag{7.1}$$

Where Z_{ijk} stands for the height value of every point in the associated volume of the pixel (i, j). Therefore, A is an accumulator of height differences. In case no points are present in the volume, H_{ij} and A_{ij} are assigned the global minimum height and 0, respectively. Finally, A is normalized within the range $[0, 1]$.

Having A, a binary image, T, can be defined to signal the presence of power line pylons. The assignation of T_{ij} follows the rule:

- $T_{ij} = 1$, if $A_{ij} >= \gamma$
- $T_{ij} = 0$ otherwise.

There, γ is an adjustable threshold which should be assigned in a way that allows the detection of the maximum number of pylon points with the minimum possible number of false positives. Experimentation was conducted and introduced in Subsection 7.2.5 to find a proper value for the parameter, which was finally assigned 0.25. An example of a T image can be seen on Figure 7.1.

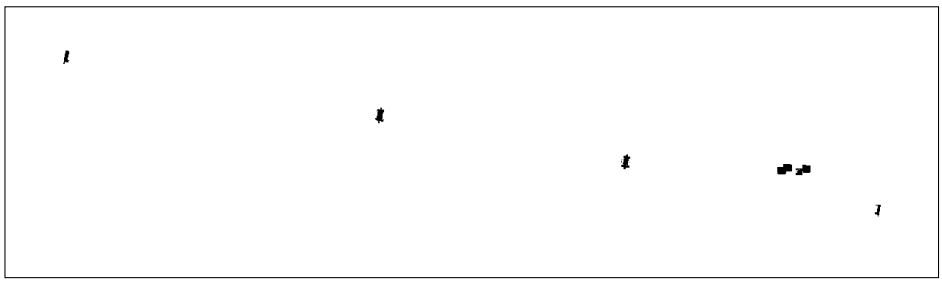


Figure 7.1: Inverse T image example. Four pylons are visible in the main diagonal, as well as two false positives located between the latest two pylons.

Other measure images are based on intensity values. Let ϵ be the minimum possible height value for a wire point in relation with the ground height below it. Considering this, a reflectivity accumulate image is defined, R, whose pixel values are assigned using the Expression 7.2:

$$R_{ij} = \sum_{k=1}^{n} (I_{ijk} \cdot Q_k) \tag{7.2}$$

Where I_{ijk} is the intensity value of the points in the associate volume of the pixel, and Q_k is a binary evaluation for each point whose value is given by the following rule:

- $Q_k = 1$, if $Z_{ijk} - Hij > \epsilon$
- $Q_k = 0$ otherwise.

Other image, N, is defined and their pixels assigned the number of points included in their associated volumes which fulfills $Q_k = 1$. This way, it is possible to define a mean intensity image, M, assigning their pixels as in Expression 7.3:

$$M_{ij} = \frac{R_{ij}}{N_{ij}} \tag{7.3}$$

M is then adjusted so its values range between 0 and 1. Having M, the binary image W which signals the presence of wires surges from the rule:

- $W_{ij} = 1$, if $M_{ij} < \omega$ and $Tij = 0$
- $W_{ij} = 0$ otherwise.

The ω parameter is used to filter regions with low mean intensities. A low intensity in LIDAR data implies a low reflectivity, only seen in objects as power lines, pylons or dense vegetation areas. Considering the intensity ratios commented in Subsection 7.2.2, it was decided to set this value to 0.3, which is an intermediate value between the wire average (0.23) and the pylon average (0.34), suppressing the latter. Figure 7.2 shows an example of wire images.

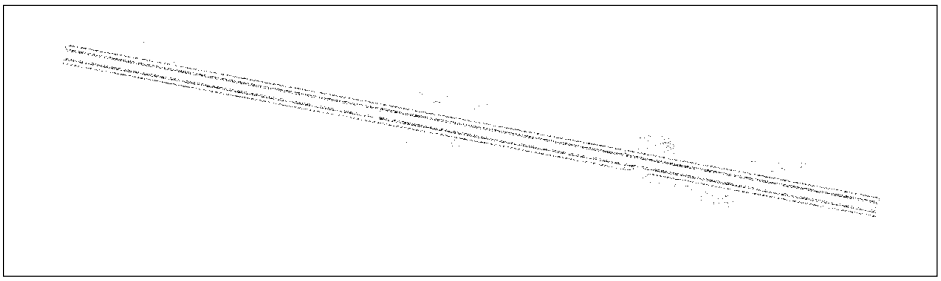
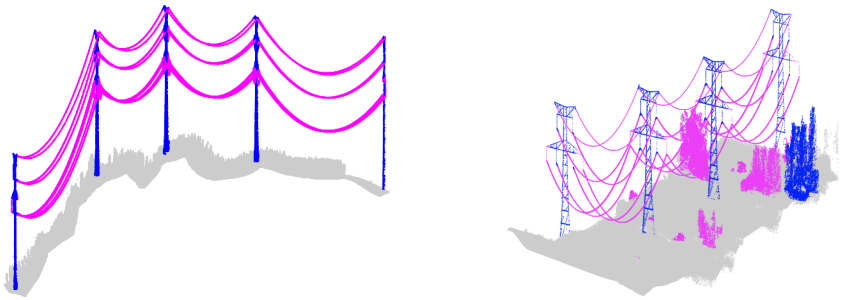


Figure 7.2: Inverse W image for a file of the validation set.

An initial classification of points is performed from T and W images, classifying as pylons all points whose associated pixel in T is selected, and as wires those points whose associated pixel in W is selected and have a height difference with the ground greater than ϵ.

7.2.4 Filtering selected areas

The initial classification provided by the first stage of the pipeline can include false positives, generally due to high vegetation with similar intensity values or placed immediately below the wire, as seen in Figure 7.3. This filtering stage aims to remove those false selections by using the return number of the points and their intensity.



(a) Irregular orography (b) Trees below line

Figure 7.3: Initial classification for two files of the validation set. Blue stands for pylon and magenta for wire areas.

The pylon case can be dispatched by checking the distribution of return values in the candidate area. As shown in the data analysis from Section 7.2.2, points grouped under the pylon category show a distribution of return values with low percentages in the first return and low deviations. It could happen that the candidate cluster contains some wire or terrain points near the pylon itself, so the conditions should be slightly less restrictive than the ones suggested by the analysis. We are considering a candidate area as a pylon if it fulfills all the following criteria:

- **First return points** $> 63\%$ of the points.
- **Second return points** between 19% and 30%
- **Third return points** between 2% and 6.5%
- **Mean intensity value** below 2200

The wire case has a special issue to take into account: there could be false positive points in correctly classified areas due to high vegetation or objects just below the wire itself. Each cluster extracted by using the algorithm is then evaluated, confirming them as a wire if their intensity mean values remain below 2000, and removing it otherwise. All intensity thresholds during this stage have been set according to the initial analysis of data.

In order to obtain the pylon and wire clusters for such analysis, two strategies have been used. The first one involves an agglomerative clustering algorithm [123]. The algorithm will join pairs of points or clusters step by step, choosing each time the closest pair based on a given distance, and stops when all possible distances between pairs are over a given cutoff. For this case, Euclidean distance between the positions of the points is used with a cutoff of 6. It is wide enough to differentiate wire clusters from possible vegetation areas near them, as it could be seen in the example of Figure 7.4.

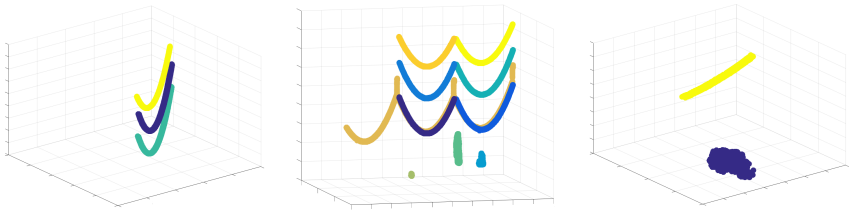


Figure 7.4: Agglomerative clustering applied to different wire candidate areas. Each cluster is represented with a different color. Subfigures in center and right include clusters with some small vegetation areas below the wires to be discarded.

The second one is the generation of *three-dimensional alpha shapes* from the point sets. This technique were introduced in 1994 by Edelsbrunner and

Mücke [204] as a 3D extension of the *alpha shape* [124], which on its part is a generalization of the convex hull. A point p_1 belongs to a given alpha shape when it is possible to have an sphere of radius alpha, from which p_1 is placed in its boundary, and another point p_2 of the shape is also in its boundary. This could be used to achieve similar groupings as in the agglomerative algorithm by using an alpha value of 6, without the needs of a massive distance matrix and ensuring an execution time of $O(n^2)$ for the worst possible case. However, this technique does not create clusters of size 1 so some points may not be considered during the process. This is critical since a wire is a line of individual points and we should consider all wire points. An experiment has been conducted to determine, based on the final results and the execution time, which one of the filtering techniques is most convenient for this stage.

7.2.5 Tuning and validating the corridor detection pipeline

In this section, several experiments are introduced to find the best configuration possible for the generation of images and validate the behavior of the method on a set with several different point clouds. A manual classification of the files is used as ground truth to compare.

All the experiments have been conducted in Matlab 2015b over a MacOS environment. From the four parameters given in the method, the same values have been provided in every experiment for the following three: $\alpha = 1$, $\epsilon = 8$, $\omega = 0.3$.

Four quality measurements have been explored throughout the experiments in order to assess the behaviour of the pipeline: *recall*, *precision*, *F1* score and IoU score. The formula to compute each one of the scores is introduced in Annex I.

Adjust of the γ parameter

As it has been already commented in the subsection *Image-based classification*, the generation of the binary image T for the selection of pylon candidate areas relies on a parameter, γ, which ranges between 0 and 1. It is necessary to find a value of γ that allows the classification of the maximum number of pylon points without generating a huge number of false positives to be corrected in the second stage.

γ	Total	Ok	FN	FP	Recall	Prec.	F1	IoU
0.15	266368	255223	11615	205067	0.956	0.554	0.702	0.541
0.20	266368	252065	14773	100289	0.945	0.715	0.814	0.687
0.25	266368	249377	17471	66929	0.935	0.788	0.855	0.747
0.30	266368	245111	21727	52528	0.919	0.824	0.868	0.767
0.35	266368	241838	25003	45471	0.906	0.842	0.873	0.774

Table 7.3: Results for pylon classification, using only the T image, over the validation set with different γ values.

An initial experiment was designed to tune this parameter in which the first stage of the algorithm is run with different values of the parameter γ over all the point clouds in the validation set. Comparisons with the ground truth are made to determine the evolution of the completeness and the correctness of the predictions. The results can be found on Table 7.3.

From those results can be extracted that the use of low values of γ generates great amounts of false positives, although it also classifies the majority of pylon points. By increasing the value, the precision rises drastically at expenses of a linear fall in the recall. The $F1$ and IoU indicators, that consider both recall and precision values, have an exponential fall for 0.15 and 0.20 values, and slightly increasing values for $\gamma = 0.25$ and higher, which suggests rising the γ value for a better interclass prediction. However, a good precision score is also necessary, as we should avoid not detecting the presence a pylon. As a compromise between both facts, $\gamma = 0.25$ fits well, fulfilling the goals of classify as much points as possible while controlling the false positives in classification and thus the execution time of the second stage.

Selecting the filtering method: accuracy results

Once the γ factor is set, the next step is to run the whole pipeline over the validation and test sets, considering the fact that there are two possible filtering algorithms to choose. This means that two runs per set have been done. The idea of this experiment is to analyse the behaviour of the methodology and choose a filtering option, comparing their completeness, correctness and execution time results.

Figures 7.5 and 7.6 show the results of execution times using the agglomerative and α-shape filtering methods on pylon and wire groups, showing that there are important differences between them. For all point clouds in the datasets, the α-shape filtering method is faster than the agglomerative clustering. Moreover, the agglomerative clustering execution times rise exponentially with the amount of points, whereas the execution times of α-shape filtering increase in a linear manner.

Class	Total	TP	FN	FP	Rec.	Prec.	F1	IoU
Pylon	266838	249367	17471	49735	0.935	0.834	0.881	0.788
Pylon	266838	249367	17471	49735	0.935	0.834	0.881	0.788
Wire	1695303	1651047	44256	11506	0.974	0.993	0.983	0.967
Wire	1695303	1650792	44511	11396	0.974	0.993	0.983	0.967
Power	1962141	1950026	12115	11623	0.994	0.994	0.994	0.988
Power	1962141	1949769	12372	11521	0.994	0.994	0.994	0.988

Table 7.4: Results of the execution of the method over the test set, using α-shape (light grey) and agglomerative (dark grey) filtering stages, over the validation set. Rec. and Prec. stands for recall and precision.

The quality results, which can be read in Tables 7.4 and 7.5, show that the

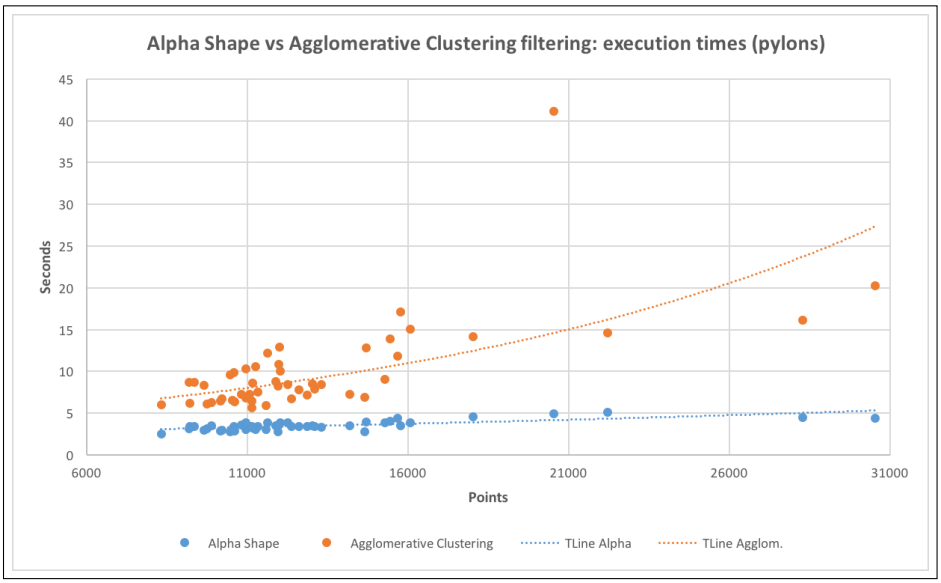


Figure 7.5: Execution time curves of the filtering algorithms for wire points, generated using all point clouds in the validation and test sets

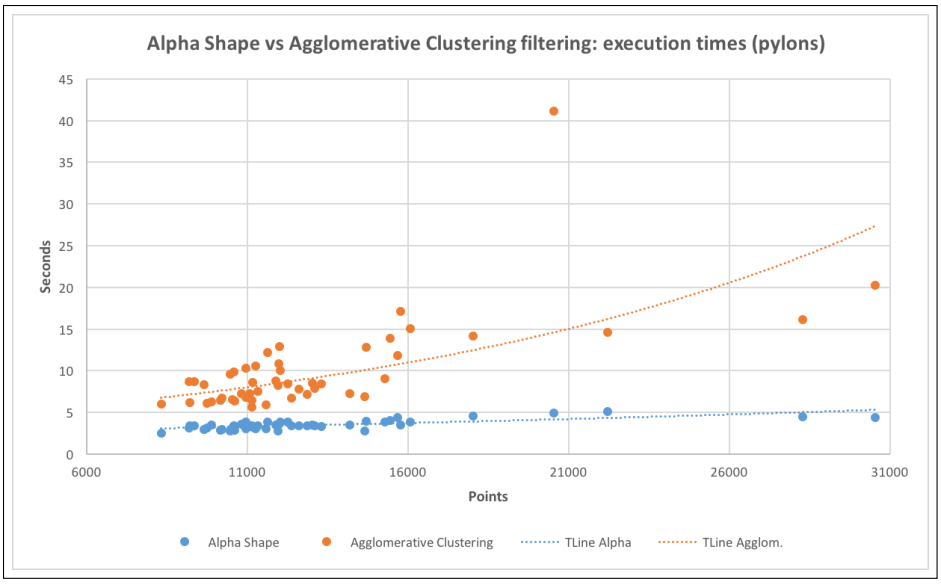


Figure 7.6: Execution time curves of the filtering algorithms for pylon points.

differences of using α-shape or agglomerative clustering to classify wire points in a point cloud are almost inappreciable: the variation on the true positive results

is of 300 wire points out of more than 1.6 million. Still, the best technique for wire filtering is α-shape. For pylon classification, the filtering algorithms perform equally for all point clouds in the set.

Class	Total	TP	FN	FP	Rec.	Prec.	F1	IoU
Pylon	262598	241963	20634	60135	0.921	0.801	0.857	0.750
Pylon	262598	241963	20634	60135	0.921	0.801	0.857	0.750
Wire	1652284	1610351	41924	16367	0.975	0.990	0.982	0.965
Wire	1652284	1609945	42339	16588	0.974	0.990	0.982	0.965
Power	1914882	1902540	12351	26295	0.994	0.986	0.990	0.980
Power	1914882	1902096	12785	26535	0.993	0.986	0.990	0.980

Table 7.5: Results of the execution of the method over the test set, using α-shape (light grey) and agglomerative (dark grey) filtering stages, over the test set.

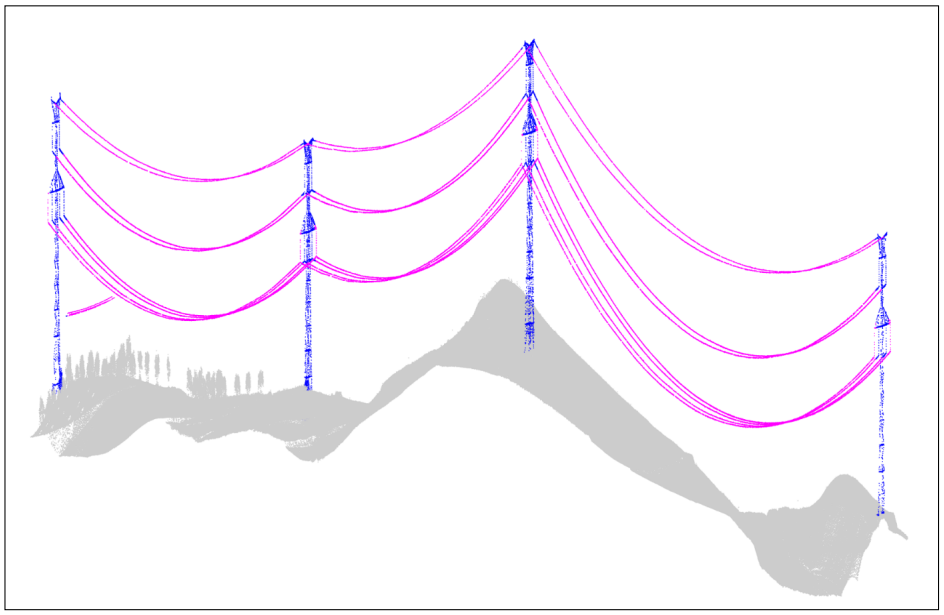


Figure 7.7: Final result of the classification process for a file of the test set.

Considering now the α-shape method as our filtering technique, it is possible to discuss the overall behavior of the methodology. It generates promising results, with completeness rates of 97.46% while classifying wires and 92.14% for pylon classification. According with these results, the method seems to be competitive compared with the reported results of other works: 97% and 92% respectively in the work of Kim and Sohn [190], 96.5% for wires from the work of Liang et al. [192] or 98,3% for wires from the work of Jwa et al. [189]. However,

it is impossible to ensure it without a direct comparison of all the methods over the same point clouds. A final result for a given point cloud of the test result can be shown in Figure 7.7.

On the other hand, there is still an appreciable number of false positives on pylon classification, of around 0.14% of the total size of the set, which affects the quality of the prediction. A smaller but also appreciable number of false negatives is also found in wire prediction. An unavoidable question comes: Do the wire false negatives and the pylon false positives match? Considering both power corridor categories as a single unit, called *Power* in the quality result tables, it is possible to see an answer: The number of *Power* false negatives diminished to be lower than those of the both subclasses all together, and the number of their false positives is also lower than those of the *Pylon* class alone. Only a number of FP of 0.06% of the set size remains, which confirms much of the wrongly classified points in both wire and pylon groups belong to the other group. Fortunately, this is an issue that can be corrected during the characterization of the corridor elements. This is explained in the next section.

7.3 Characterizing the corridor elements

This second section of the chapter defines some terminology related to the power corridor maintenance tasks and presents the methodology followed to refine the classification of the power corridor. This methodology uses six additional steps for reclassifying the initially detected corridor points into five, more complex categories: *insulator strain*, *pylon*, *common wire*, *shield wire* and *chain*. All conductor points are segmented into individual units so a three-dimensional catenary model is computed and saved for each of them. From pylons, a skeleton-like model is also generated from the classification. Finally, more experimentation are also introduced to assess the behaviour of the new pipeline, involving pylon/wire classification, insulator strain detection and 3D catenary model error.

7.3.1 About the different components of a power corridor

It is well known that a power corridor consists of series of two main elements: pylons and wires. However, there are several subcategories of each one of these elements. The ending points of a single wire are also different in relation with the type of pylon it is attached to.

Regarding the pylons, there are several types of them, from which two are widely used: *suspension pylons* and *anchor pylons*. Suspension pylons are towers in which the conductors are just hanging. Mechanical tension is expected to be the same in both sides of the suspension pylons, so only lateral and downward forces affect the structure. They are used whenever a corridor goes straight, without significant changes in its direction. Anchor pylons, also called dead-end or tension pylons, are self-supporting towers. Their structure can manage changes in the mechanical tension without collapsing, so they are particularly useful when the direction of the corridor should change. They are also present

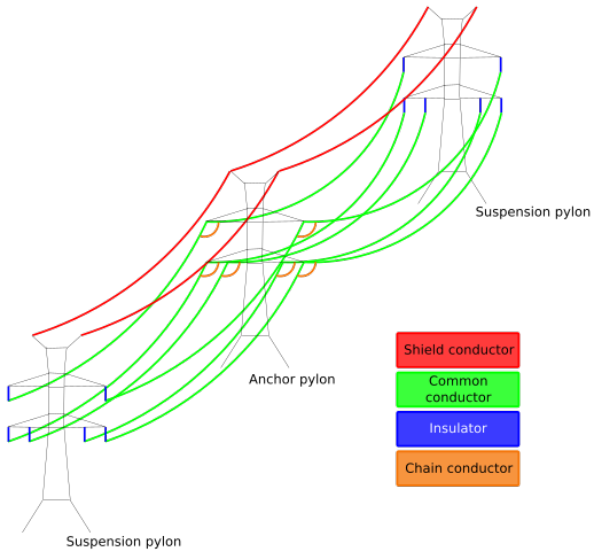


Figure 7.8: Components of a power corridor in which different pylon types are present.

from section to section, ensuring the infrastructure resists even if a conductor breaks. However, they are heavier, require more material and have a higher cost, so lines are generally not composed of only anchor pylons.

Figure 7.8 shows a representation of a power line corridor with suspension and anchor pylons. In suspension pylons, wires hang from *insulator strains* that are tied to the pylon. This way, losses of energy from the wire to the pylon are avoided. This should be taken into account at the moment of modeling the wire, since the insulator strains are not part of the wire and can affect the generated model. For the case of anchor pylons, conductors are not hung but tied to the pylon. The final part of the wire includes insulation. To connect two consecutive power lines, small conductors called *chains*, also named *bridges*, are used, as shown in Figure 7.8. As chains are individual conductors, it is interesting to differentiate them and generate their models.

Regarding the wires, there is also a special type that should be taken into account: the *shield wire*, also called ground wire or guide wire. Shield wires are earthed and placed at the top of the structure, and their goal is to reduce the likelihood of lightning strikes to the conductors in case of storms.

7.3.2 Architecture of the proposed solution

A sevenfold staged pipeline is proposed to perform the following tasks on a point cloud:

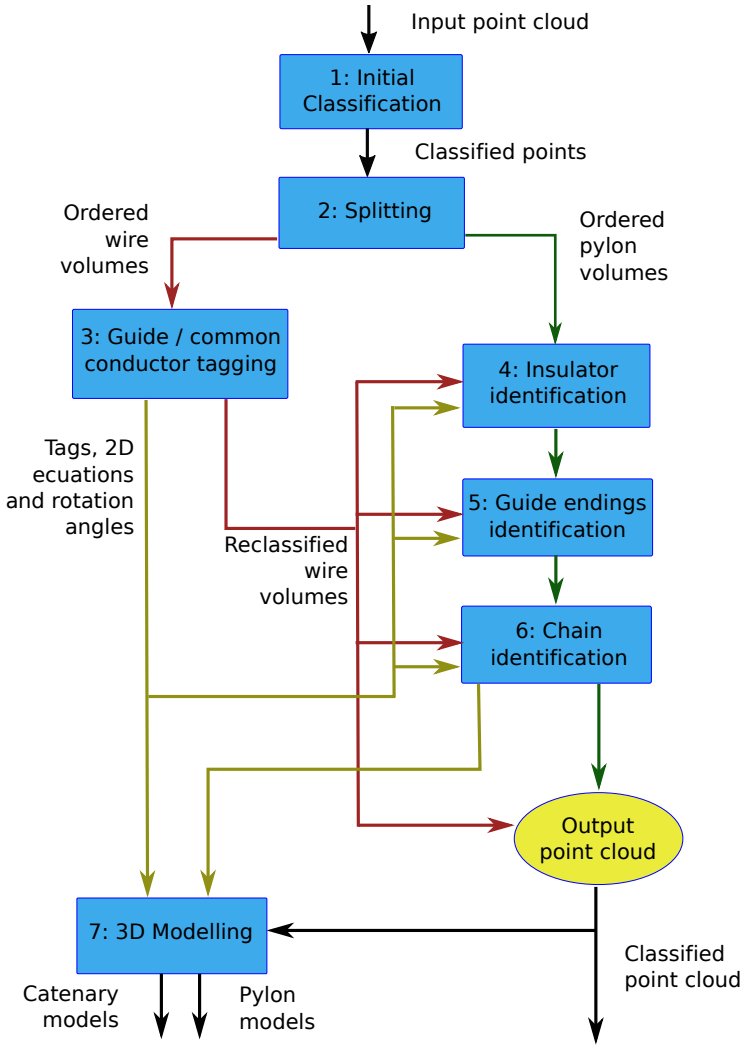


Figure 7.9: Stages of the proposed pipeline.

- Classification of power line points (pylon and wire).
- Proper identification of the insulators, the chains and the shield wires in the corridor.
- 3D-modeling of the individual conductors of all classes.

For this architecture, a diagram is shown in Figure 7.9. The first of the seven steps is an initial classification of the points. This can come from the previously proposed image-based procedure or from any other source. Stages from 2 to 7, which are explained in detail in the following subsections, include splitting, wire tagging and identification of shield wires, identification of insulator strains, identification of the endings of shield wires, identification of chains and model fitting.

This method will assume the following conditions to work:

- The point cloud is acquired by a helicopter flight, following the power corridor. This allows to acquire the majority of the power corridor in a single strip.
- The point density should be of at least 10 points per m^2. Moreover, it is recommended to have a point density of more than 25 points per m^2. This is due to the need of segmenting not only the main conductors, but also the insulator strains and the chains.
- The sharpest turning angle allowed between two pylons is of 90°. In case the corridor has sharper turns, the point cloud could be split in two before processing it, using one of the pylons as cutting limit to overcome the issue.
- Atmospheric and timeout related noise have been removed from the point cloud.

7.3.3 Stage 2: Splitting and ordering the corridor

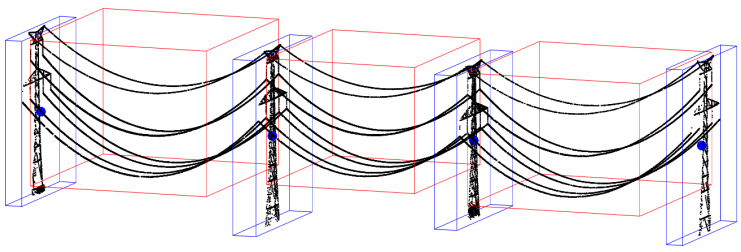


Figure 7.10: A representation of the pylon volumes (blue) and wire volumes (red) for a cloud with 2.1 million points. Pylon centers are depicted as blue dots inside each pylon volume.

This stage looks for providing ordered pylon and wire volumes. It starts by grouping all pylons points so each group correspond to a unique pylon. For this work, three-dimensional α-shapes with $\alpha = 4$ ensure the desired grouping, but any other grouping algorithm is also valid as well as it achieves the described effect. For every single pylon group, its center is calculated, and the bounding box containing the points of each pylon is generated with an X, Y margin of

value s. The volume should cover the whole pylon group as well as the wires near the pylon. The pylon groups are then ordered in the direction of the longest dimension of the point cloud.

Volumes enclosing all wire points between two consecutive pylon volumes are also generated: those will be the *wire volumes*. Figure 7.10 shows a representation of the splitting stage results.

7.3.4 Stage 3: Shield and common wire segmentation

This stage takes the wire volumes and the pylon centers from the previous stage. For every wire volume, its adjacent pylon centers, C_{p1} and C_{p2} are taken. The directional angle of the volume in relation with the Y axis, θ, is calculated by using the Expression 7.4:

$$\theta = -\arctan\left(\frac{c^y_{p2} - c^y_{p1}}{c^x_{p2} - c^x_{p1}}\right) \tag{7.4}$$

An agglomerative clustering algorithm is applied to split the points of each volume into different wire units, not only the ones going from the two pylons, but also any wire that has a different direction and crosses the power line below its wires. Clustering will consider XYZ values and will use standardized Euclidean distance. This is a variant of the Euclidean distance in which the coordinate differences between two points are scaled using the standard deviation of the coordinate. This coordinate standard deviation is computed considering all the observations. A cutoff ω_1 is set as a threshold to stop the clustering process. It is related to the distance and should be set so it can isolate the different wires without fragmenting them. For every cluster obtained, its own rotation angle θ_i is computed. We will select clusters which fulfill the Expression 7.5:

$$|\theta - \theta_i| < \epsilon_1 \tag{7.5}$$

in which ϵ_1 is set experimentally. From those selected clusters we will consider the equation of the line which best fits the X,Y position of their points and their minimum height, $\min(Z_i)$. Every non-wire-classified point in a wire volume whose X,Y position is well predicted by the straight line equation of one of the wires and whose height is higher than $\min(Z_i)$ is reclassified as wire. This step is introduced to improve possible issues in the initial classification.

After that, the clustering process is reapplied in the same conditions for the selected clusters and the reclassified points. The goal is to regenerate the groups by joining clusters that initially are split due to misclassification of some sections of the wire. The result of this operation can be seen on Figure 7.11. All the resulting clusters are assigned a unique, numerical tag. Finally, it is determined which of the clusters correspond to shield wires. As the different groups of wires can be found at similar heights and the shield wires are always at the top, a height-based grouping of the clusters is conducted, applying a cutoff ω_2 to properly separate the wires by levels. The highest group of clusters will be considered shield wires.

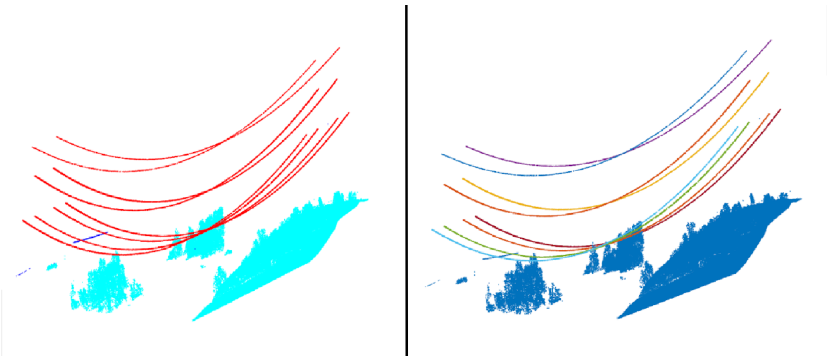


Figure 7.11: Results of the segmentation and tagging stage for a wire volume. Left image shows identified wire points in the main direction (red), crossing wires (dark blue) and non-wire points in the volume (light blue). At the right, main wires are segmented into individual conductors.

7.3.5 Stage 4: Insulator strain identification

This stage takes the pylon volumes from Stage 2 and the segmented conductors from Stage 3. For every point P_i in each pylon volume, its n nearest neighbours in the pylon volume, N_{ij}, are found, with n set experimentally. Index i will refer to the point and j to the neighbor. For each neighbor, the euclidean distance to the point, d_{ij}, is computed. The difference vector, $V_{ij} = P_i - N_{ij}$, is also computed. V and d are used to calculate a distance ratio, R, for every axis, as seen in Expression 7.6 :

$$\begin{aligned} R_{ij}^x &= V_{ij}^x / d_{ij} \\ R_{ij}^y &= V_{ij}^y / d_{ij} \\ R_{ij}^z &= V_{ij}^z / d_{ij} \end{aligned} \tag{7.6}$$

Once the ratios are calculated for all the neighbors of the point, their means are computed and a verticality test for each point is run as in Expression 7.7:

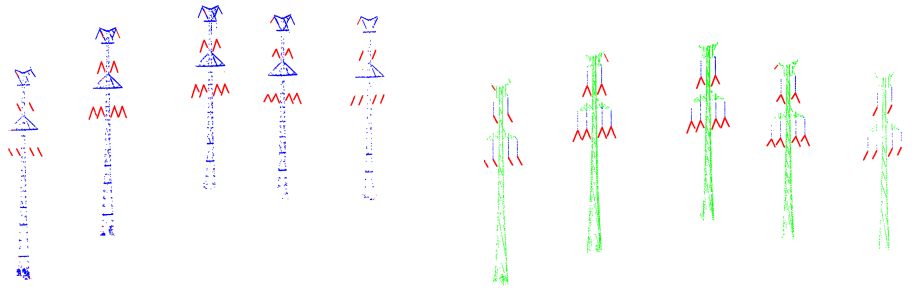
$$\frac{mean(R_{ij}^x)}{max(\alpha_1, mean(R_{ij}^z))} + \frac{mean(R_{ij}^y)}{max(\alpha_1, mean(R_{ij}^z))} < \epsilon_2 \tag{7.7}$$

The test is meant to find clusters in which all the neighbors are at the same planimetric position and, thus, correspond to a vertical cluster of points. For this particular case, $mean(R_{ij}^x)$ and $mean(R_{ij}^y)$ tend to zero. Values for α_1 and ϵ_2 are determined experimentally.

In suspension pylons, it is expected that some non-vertical points in the pylon volume will be found separated from the arms of the pylon and their X, Y positions should be predicted by the 2D equations of the conductors obtained in Stage 3: those will be the points at the end of conductors. To obtain such

points, agglomerative, standardized XYZ-distance-based clustering is applied to non-vertical points. A cutoff ω_3 is used so these points at the end of a wire are grouped apart from the main part of the pylon. Groups are considered to contain wires when more than 95% of their 2D positions can be predicted by a 2D conductor equation. For those groups, the point with maximum height is saved as ending point for catenary modeling. Points at both sides of the maximum are tagged with the id of the closest conductor.

If at least four wire groups are found near each pylon, the presence of insulators is considered. Those will be points that passed the previous verticality test and are placed in a square area of side length l, centered in the highest points of each group. They also must have a z value greater than the highest point of the group, P^z, but not higher than $P^z + h$. l and h are set experimentally. Otherwise, no points in the pylon volume will be classified as insulator. An example of the results of this stage is shown in Figure 7.12.



(a) Common wire endings (red). (b) Detected insulator strains (blue).

Figure 7.12: Results of the stage 4 for pylon volumes of a sample point cloud.

7.3.6 Stage 5: Identification of shield wire endings

This stage takes the pylon volumes and the shield conductor 2D equations from previous stages. Every pylon point whose X, Y position is predictable by a 2D equation with a tolerance of α_2 will be selected as shield wire candidate. Selected points will not only include the endings of shield wires, but also lower points belonging to the arms of the pylon. To differentiate the shield ending points, the closest wire point is found for each selected pylon point. If the closest wire point tag matches with a shield conductor tag, the pylon point is confirmed as belonging to the conductor, so it is reclassified and assigned the same tag. Otherwise, the point will continue being part of the pylon.

Finally, from all the points in the pylon volume selected as shield points, we take the maximum height per assigned tag and save its corresponding point as ending point for catenary modeling. A result of the stage for a sample point cloud is shown in Figure 7.13.

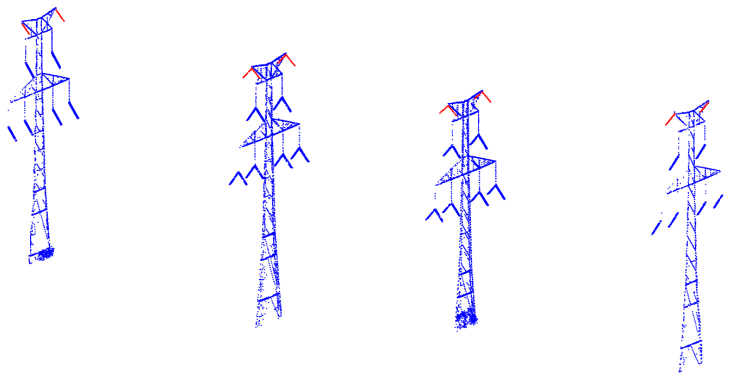


Figure 7.13: Results of the stage 5 (Identification of shield endings) for pylon volumes of a sample point cloud. Points identified as endings of a shield wire are shown in red.

7.3.7 Stage 6: Identification of chains

Chains are expected to be found only in anchor pylons. At this stage of the process, it is possible to consider as anchor pylon any pylon in which insulator strain points have not been identified.

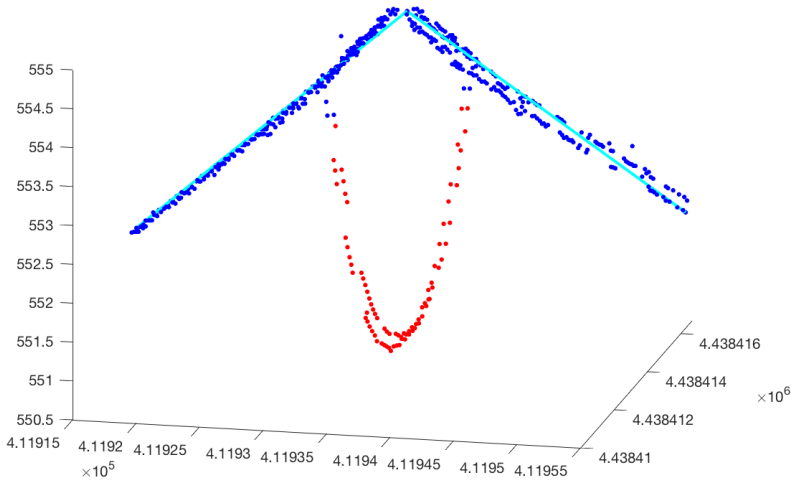


Figure 7.14: Classification of chain points (in red) as points whose height is lower than the one predicted by the 3D line equations (in light blue).

For those particular pylons, we select every point whose X,Y position can be predicted with a 2D equation of a common conductor with a tolerance of α_3. The points selected by this method can also include small points from pylon arms. To suppress false positives, agglomerative clustering is performed, using X,Y,Z positions and a cutoff ω_4, so different groups of points are obtained. Every cluster which includes less than s_2 points is discarded as candidate. A second grouping is then made for the non-discarded groups based only on the maximum height, with a cutoff ω_5 set experimentally so all clusters representing wires which are tied to the same arm are grouped together. The minimum of the maximum heights of every arm group is selected as maximum acceptable height for all the clusters in the group. Points over that height are discarded and remain classified as pylon points.

Finally, for each candidate cluster of points, the 3D euclidean distance between the points and the pylon are computed and the furthest points are taken, as well as the point with maximum height, which is the closest one. The behavior of the catenary in the proximities of the pylon can be approximated as a 3D-line, so 3D line equations are calculated between the furthest points of the cluster and the closest. All points whose position is predicted by such equations with a tolerance of α_4 are considered belonging to the closest conductor and tagged with its id. Points not predicted by the equations due to being below its theoretical height are classified as a single chain and tagged with its unique identifier. This is best visible in Figure 7.14.

7.3.8 Stage 7: 3D modeling of components

Once the classification stages are finished, it is possible to generate individual models of each corridor element. For the wire elements (shield, common and chain conductors), a 3D-wise catenary curve model is chosen. For the pylon, a vector data based model of the pylon skeleton is generated from the data. Both can be seen in detail in the following subsections.

Catenaries

The catenary is the ideal curve which represents the behavior of a wire which is suspended from its two endings. Its equation was first expressed by Huygens in 1691 and can be written as in Expression 7.8:

$$v - v_0 = a \cdot \cosh\left(\frac{u - u_0}{a}\right) \tag{7.8}$$

where a is the main catenary parameter and equals the horizontal tension in the wire endings, divided by the weight per length unit. u_0 is the u coordinate of the ideal lowest point in the curve, P_1, and v_0 can be defined as $v_1 - a$, being v_1 the coordinate v of P_1. This is best shown in Figure 7.15. Hence, the curve equation could also be expressed as in Expression 7.9. Neither a nor (u_1, v_1) are known, so a multivariable optimization should be used to find the values of those parameters which best fit the input data. To accelerate the computations,

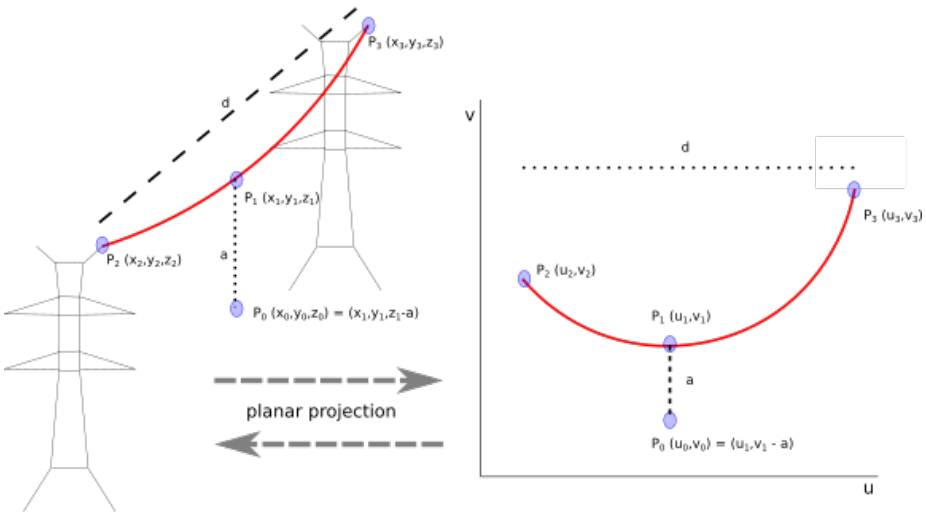


Figure 7.15: Elements on the catenary expression for 2 and 3 dimensions.

the lowest LiDAR point coordinates can be provided as starting (u_1, v_1) in the optimization process.

$$v - v_1 = a \cdot \left(\cosh \left(\frac{u - u_1}{a} - 1 \right) \right) \tag{7.9}$$

The input of this stage is composed of all the points of a conductor, including the endings, P_2 and P_3 and the rotation angle θ_i that was found for that conductor in the previous stages. The points are projected to a planar *(u,v)* space, reducing the dimensionality of the problem, in an analogous manner to the one seen in Chan et al. [205]. After that, we performed the above-mentioned optimization process to find the a, u_1 and v_1 values for the model which minimize the sum of squares of the residuals (SSR). For this minimization task, the Particle Swarm Optimization method [206] was applied. The SSR optimization function is defined considering all points in the conductor as in Expression 7.10:

$$SSR = \sum_{i=1}^{n} (P_i^z - v_i)^2 \tag{7.10}$$

with n the number of points in the conductor, P_i^z the actual height of the point and v_i the result of computing the Expression 7.11:

$$v_i = v_1 + a \cdot \left(\cosh \left(\frac{u_i - u_1}{a} - 1 \right) \right) \tag{7.11}$$

This results in a pair a - P_1 for the final model. At this point, the P_2 and P_3 input points for the endings of the catenary are checked. If the error between

those points and the adjusted curve is lower than 3 standard deviations from the general *Root of Mean Squared Error* (RMSE) of the curve, they remained the same. Otherwise, they will be considered outliers. This could happen when a pylon point is misclassified as being part of the wire. For that case, the closest non-outlier point will become the new ending of the curve, substituting the erroneous one. Then, a final three-dimensional expression of the catenary curve can be expressed as follows in Expression 7.12:

$$\begin{aligned} x &= x_2 + t \cdot (x_3 - x_2) \\ y &= y_2 + t \cdot (y_3 - y_2) \\ z &= z_2 + v_0 + a \cdot \cosh\left(\frac{t \cdot d - u_0}{a}\right) \end{aligned} \tag{7.12}$$

Where t ranges in $[0, 1]$, $[x_2,y_2,z_2]$ and $[x_3,y_3,x_3]$ are the 3D coordinates of the points P_2 and P_3; $[x_1, y_1, z_1]$ the 3D position of the P_1 point; and u_0, d and v_0 are respectively the result of Expressions 7.13, 7.14 and 7.15. As an output of the stage, a structure containing the basic elements of the model, a, P_1, P_2 and P_3, is generated for each conductor.

$$u_0 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{7.13}$$

$$d = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} \tag{7.14}$$

$$v_0 = v_1 - a = z_1 - z_2 - a; \tag{7.15}$$

Pylons

Generating a unique algorithm to extract a perfect 3D model out of any high voltage pylon is a hard task. There are no standards or rules on how a power pylon should be, but only on what specifications should match. These specifications are related to number of conductors, the tensions and forces the pylon should resist and possible height restrictions in the area. As well as they are satisfied, pylon makers can design freely their towers. However, all the different designs have common elements: a central mast and different arms to support the conductors. This fact allows to design a vector data based model which represents the pylon as a skeleton, which can be used not only for progressive streaming but also for inspection and maintenance purposes. A vector pylon model is defined as:

- A vertical line segment that represents the main pole.
- A variable number of parallel horizontal line segments that represent the pylon arms.
- A variable number of points that represents the union points between the conductors and the pylon.

As most of points in a pylon are distributed in the main pole, and most of who are not are at least symmetrically distributed around it, as it could be seen in Figure 7.16, the pole segment can be easily generated. Considering T to be a pylon point set, it is represented as the pair $[mean(T_x), mean(T_y), min(T_z)]$, $[mean(T_x), mean(T_y), max(T_z)]$.

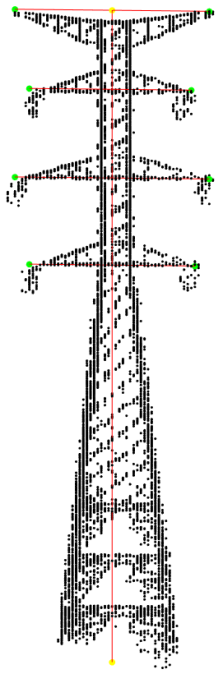


Figure 7.16: Pole and arm segments of the vector model (red) superposed over a pylon point set (black). Selected ending points of arm segments can also be seen marked (green).

Determining the number of pylon arms is a problem which can be solved the same manner as in the shield wire detection problem in Stage 3. The set of catenary endings belonging to the pylon are clustered according to their heights. This results in small groups, G, of few conductors which give information about the number of pylon arms and their arm heights H_i, that will be the maximum heights in each cluster i of catenaries.

The segments representing each arm should then be computed. For that, a subset A_i of T is extracted for each arm, considering all points in T whose height is in the range $[H_i - \delta, H_i + \delta]$. δ is set as the maximum between 1 and the amplitude of heights in each cluster. A_i is denoised using the algorithm described in Rusu et al. [207] and then, a *principal component analysis* (PCA) [208] is performed over the remaining points using only their X, Y coordinates. A directional axis is obtained as a result. Finally, all points in A_i and G_i are then projected to the axis and their distances with respect to the pole are calculated.

The two points whose distances are the largest will be considered the limits of the arm segment. This can also be seen in Figure 7.16.

The resulting segments are not perfectly aligned due to differences between the acquired points. To make them align, the directional angle of each arm is computed as in Expression 7.16, where s_1 and s_2 are the segment ends:

$$\theta = arctan2\left(\frac{s_2^y - s_1^y}{s_2^x - s_1^x}\right) \tag{7.16}$$

The mean of all the angles are also computed. For all arm angles differing less than 15 degrees from the mean, their segments are rotated the difference between the angle and the mean with respect to the vertical axis for alignment. Some pylons have special cases in which there are unaligned arms, as in line intersections. Those special arms are not taken into account for the mean and the alignment process. Instead, their segments are included as is in the model. Finally, all the catenary ending points are included as is in the final model. This final step is not really needed for streaming purposes (as they are included in the wire model), but they are necessary for inspection and management tasks in which the pylon model is also useful.

Incidence analysis

By using the points not classified in a power-related category and the generated catenary models, an analysis of possible risks and incidences in the corridor area can also be performed. For each modelled wire, the incidence analysis process looks for points that could collide if the wire rotates with respect of its resting position, e.g. because of a wind gust.

Four sequential tests are designed to check for incidences. Each new test receives as input only the points that passed the precedent tests. Let it be α the maximum expected rotation angle of the wire with respect to their resting position. Let also be d_{max} and h_{model} the minimum distance allowed between the wire and other objects and the minimum height of a given point on the wire, respectively. A given point I is an incidence if all the conditions are sequentially fulfilled:

1. **Height condition**: $h_I > (h_{model} - d_{max})$
2. **Segment condition**: Considering AI the projection of I on the catenary axis, it falls inside the axis segment formed by the catenary endings, A and B. A graphical explanation is provided in Figure 7.17.
3. **Angle condition**: $\beta <= \alpha$, where β is the angle between the vertical and the segment formed by I and AI.
4. **Distance condition**. Let it be I' the result of rotating I by an angle $-\beta$ over the catenary axis. I' will be in the vertical of AI, as well as a point p in the model. The height difference d_I between I' and p is computed. I becomes an incidence point if $d_I < d_{max}$.

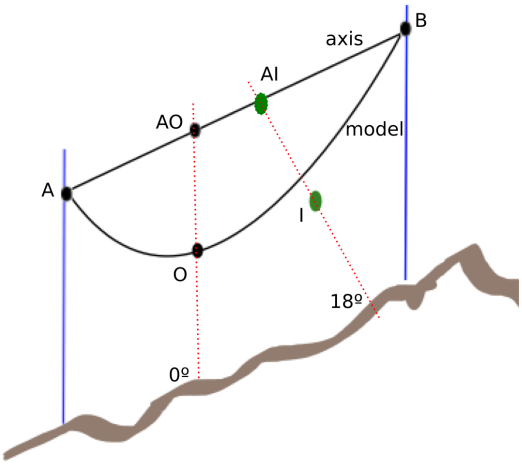


Figure 7.17: Tests of segment and angle for the incidence candidate point, I. Its projection AI on the catenary axis is calculated to determine whether it is in the wire area. The angle of I with respect to the vertical is also calculated to check if the wire movement upwards can reach the point position or not.

7.3.9 Validation of the pipeline

In this section, several tests to find the best possible configuration for the pipeline and validate its behaviour are introduced. A comparison between the pipeline and a reference method is also conducted, and a brief description of the point cloud datasets in which the tests have been done are included. All the experimentation have been conducted in Matlab 2015b over a Macbook Pro 2016 machine.

Tthe *validation* and *test* datasets described in the first part of the chapter (Section 7.2.1) have been used. A sample of those datasets is available for the public to download in the project website [1]. In order to validate the identification of insulator, shield and chain subclasses, a careful, hand-made identification of clusters representing each wire and insulator was done and given to us along with the datasets. This methodology is similar to the one exposed in the works of Guo et al. [199] and Cheng et al.[200]. The wire identification was done using industry standards, where a single conductor cluster can be composed of one, two or four wires, tied together with staple-like pieces.

Additionally, the *dataset 1* described in the work of Gu et al. [149] and already introduced in Chapter 5 has been used to provide a comparison between the here proposed method and their method. The dataset, as well as the implementation of their method, is available in [161].

[1] `http://www.ctim.es/demo113`

Adjustments of the pipeline parameters

The proposed method includes some cutoffs, tolerances, thresholds and other parameters that have been experimentally adjusted over the validation dataset to maximize the point classification recall and improve the modeling performance. This adjustment stage was performed by running mono-objective optimization over each one of the parameters.

Par.	***Range***	***Step***	***Value***	***Par.***	***Range***	***Step***	***Value***
$s(2)$	2-20	1	15 (m)	l (4)	1-3	0.5	2 (m)
$\epsilon_1(3)$	0.05-0.25	0.025	0.1 (°)	h (4)	1-5	0.5	4 (m)
$\omega_1(3)$	0.05-0.25	0.025	0.125	$\alpha_2(5)$	0.125-1.5	0.125	0.25 (m)
$\omega_2(3)$	2-10	1	5 (m)	$s_2(6)$	10-200	10	50
n (4)	2-16	1	6	$\omega_4(4)$	0.1-0.5	0.1	0.4
$\omega_3(4)$	0.01-0.1	0.005	0.025	$\omega_5(4)$	2-10	1	5 (m)
$\alpha_1(4)$	-	-	0.001 (m)	$\alpha_3(4)$	0.125-1.5	0.125	1 (m)
$\epsilon_2(4)$	1-2	0.1	1.7	$\alpha_4(4)$	0.125-1.5	0.125	0.75 (m)

Table 7.6: Resume of the parameter settings of the pipeline.

In Table 7.6, the range of variability of all the parameters during the experiment and the final value which gives the best results are presented. Those best results for the validation set are also given in the next subsections.

Pylon-Wire classification assessment

To obtain results related to point classification, the proposed pipeline has been run over the test set. Those results, which can be seen in Table 7.7, are promising. An average recall of 94.90% and precision of 90.77% have been obtained for pylon classification. For wire classification, the average recall is of 99.58% and the precision is of 99.44%.

	Test			**Validation**		
	Pylon	***Wire***	***Other***	***Pylon***	***Wire***	***Other***
Pylon	249204	2821	22553	256846	4183	10396
Wire	5461	1645395	3872	2792	1687270	1227
Other	7932	4058	42104576	7200	3850	43527832
Recall	0.9490	0.9958	0.9994	0.9626	0.9953	0.9997
Precision	0.9077	0.9944	0.9997	0.9463	0.9976	0.9997
F1 score	0.9279	0.9951	0.9995	0.9544	0.9964	0.9997
IoU score	0.8654	0.9902	0.9991	0.9127	0.9929	0.9995

Table 7.7: Confusion matrix and classification results. Pylon class includes insulator subclass. Wire class includes shield wire, common wire and chain subclasses. All other classes are grouped in Other.

The test was also run over the *dataset 1* from Gu et al. [149] using the

pipeline and their *MKSRC* implementation to compare the behaviour of the methods. It was chosen because the Matlab code of *MKSRC* was made available for the community. The dataset has a density of 13 points per square meter, which is lower than in the *test* set, and has no pylon points, but it is still good for validation of power line classification. Table 7.8 resumes the results for both methods in that dataset.

	Gu et al.		Proposed	
	Wire	**Other**	**Wire**	**Other**
Wire	1475	424	1461	22
Other	20	99829	34	100231
Exec. time (s)	2628.7		1.4	
Recall	0.9866	0.9958	0.9773	0.9998
Precision	0.7767	0.9998	0.9852	0.9997
F1 score	0.8692	0.9978	0.9812	0.9997
IoU score	0.7686	0.9956	0.9631	0.9994

Table 7.8: Confusion matrix and classification results per method, using the *dataset 1* of Gu et al.

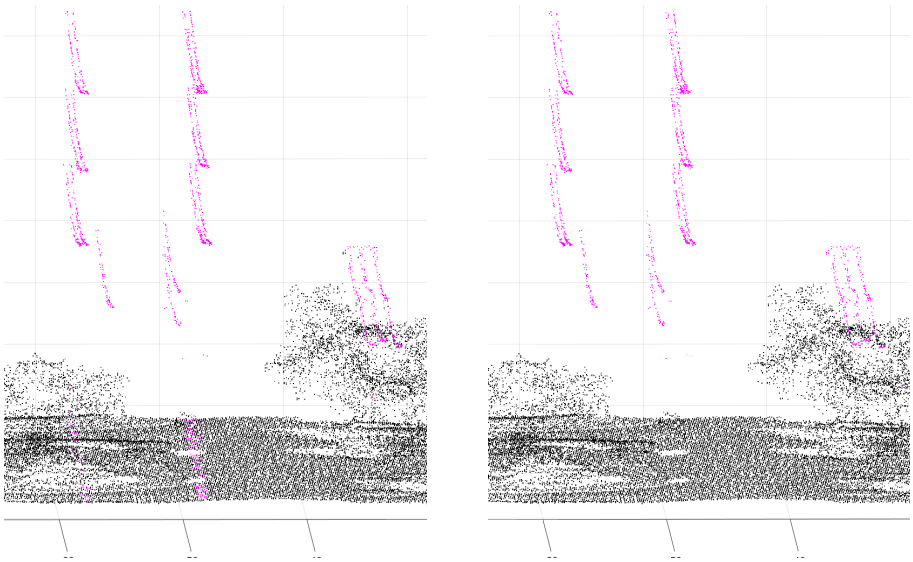


Figure 7.18: Wire classification (purple) for the *dataset 1*, using Gu et al. (left) and the proposed method(right).

In the *dataset 1* scenario, the results show that the presented method is much more precise in power line classification, at expenses of a slightly worse recall. The F1 and IoU scores are superior for the methodology here presented in both classes, and it is also significantly more efficient in execution time. Image 7.18

shows the differences between both methods. The kernel-based method from Gu et al. seems to misclassify the surface below the powerline as wire, where the proposed pipeline prevents that from happening.

Subclasses identification and catenary fitting assessment

The objective of the work described in this chapter is not only classification but also proper identification of the described subclasses of pylon and wire and modeling of the different power lines. After running the experiment on pylon/wire classification, a study covering the correctness of insulator, shield catenaries, chain catenaries and common catenaries identification is conducted. Furthermore, how the predicted catenary curve fits to its corresponding point cluster is also analysed.

The results for the insulator subclass can be seen in Table 7.9. There, an insulator is considered to be *correctly detected* when more than a half of its corresponding points are correctly classified, and *partially detected* when some points but less than a half of the total are classified.

Insulators	***Validation set***	***Test set***
Expected	501	493
Correctly detected	409 (97.80%)	474 (96.15%)
Partially detected	1 (0.20%)	2 (0.42%)
Undetected	10(2.00%)	17 (16.97%)

Table 7.9: Insulator subclass identification results.

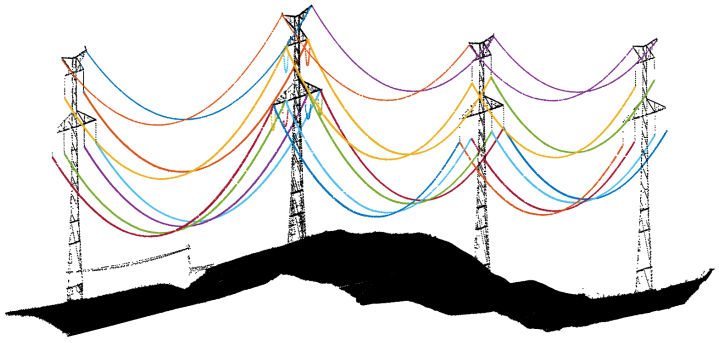


Figure 7.19: Point clusters representing different detected catenaries in a point cloud of the test set.

The results reflect that up to 96.15% of the insulator pieces present in the test set are identified properly by the proposed method. However, some of the pieces are not detected due to low local spatial resolution of the insulator points, or misclassification of the points belonging to the wire which hangs

from the piece. Upcoming research work can determine whether the insulator identification ratio uprises with the point density or not.

An analogous analysis can be done for shield, chain and common catenaries. To determine the correctness of the catenary its associated clusters are considered, as the ones shown in Figure 7.19. If a cluster covers a complete wire between two pylons and only that wire, the catenary is considered *correctly detected*, and *partially detected* if only covers a section. When the cluster includes points of more than one wire, then it is considered *over detected*. Finally, a catenary is *split* when its associated points are divided into several clusters. *Undetected* is reserved to cases where no points are found for the expected category or are entirely misclassified, e.g. a chain catenary identified as common.

	Shield		***Common***		***Chain***		***Totals***	
	V	**T**	**V**	**T**	**V**	**T**	**V**	**T**
Expected	152	154	456	462	36	36	644	652
Correct	142	143	439	434	18	35	599	612
(%)	93.4	92.9	96.3	93.9	50.0	97.2	93.0	93.9
Partial	9	7	12	19	18	0	39	26
(%)	5.9	4.5	2.6	4.1	50.0		6.1	4.0
Split	0	1	0	0	0	0	0	1
(%)		0.7						0.2
Over-detected	1	1	5	3	0	0	6	4
(%)	0.7	0.7	1.1	0.7			0.9	0.6
Undetermined	0	2	0	6	0	1	0	9
(%)		1.3		1.3		2.8		1.4
RMSE	8.0	7.3	22.8	23.1	25.5	24.1	19.3	19.4
RMSE XY	6.9	5.9	21.9	21.9	19.8	20.4	18.2	18.0
RMSE Z	2.7	3.0	3.6	3.9	10.0	7.6	3.5	3.9

Table 7.10: Catenary subclasses detection results. RMSE values are expressed in centimeters. V and T stands for validation and test datasets.

The results per category are given in Table 7.10. RMSE measures between the point and the adjusted curve (internal precision) are used for validating the fitting. They are based on the euclidean distance which is orthogonal to the curve. Those error measures are computed only for well detected catenaries.

The results reflect that a 93.87% of the total of catenaries are properly detected in the test set. Although a direct comparison cannot be established, the method seems again to be competitive compared with the results reported in the work of Guo et al. [199], offering a higher recall and less cases of undetection and splitting.

Going to a subcategory detail, the recall is slightly worse for shield catenaries (92.86%) and slightly better for common catenaries (93.94%). For chain catenaries, the recall for the test set is the highest (97.22%), but it is also noteworthy an appreciable behaviour difference when the validation set is used, where a half of the chains are only partially detected.

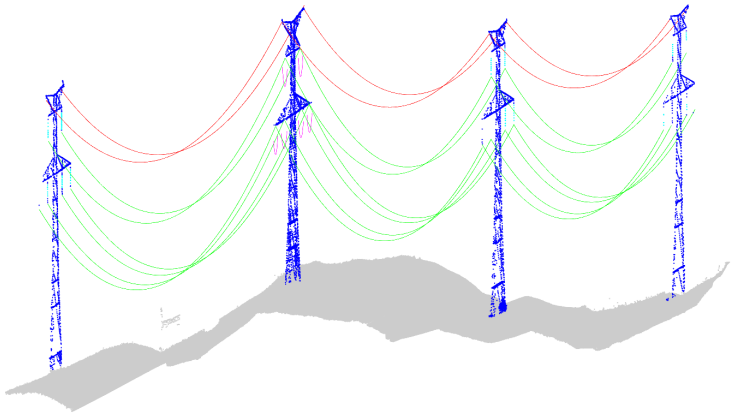


Figure 7.20: Curves representing the estimated shield catenaries (red), common catenaries (green) and chain catenaries (purple).

In relation to the fitting of the computed 3D curve equation to the points, the results show an average RMSE of 19.4 cm. This is worse than the 5.5 centimeters reported as a result in the previous work of Guo et al., but still competitive considering the future application of this work, which is the detection of objects which could collide with the power line. An image representing the generated tridimensional curves for a set is shown in Figure 7.20.

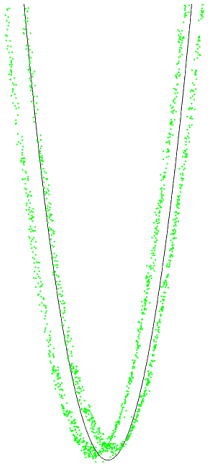


Figure 7.21: Adjust of a section of a catenary model (black) to input data (green).

Focusing into a subcategory detail, the average RMSE for fitting of shield catenaries is of 7.3 cm., while common and chain catenary have worse errors, 23.1 and 24.1 cm. There are also significant differences between the error with

respect to the XY coordinates and the error with respect to the Z component. This second one is lower for all cases, ranging only between 2.7 cm. and 7.6 cm. with an average of 3.9 cm. for the test set. By analyzing carefully the distribution of the points in respect with the generated model, it is observed that common and chain lines are often composed of two small sublines, with the generated model placed between them, as it can be seen in Figure 7.21. Extensions of this work will include ways to detect these situations and generate submodels in order to improve the error fitting.

7.4 Conclusions

In this chapter, a methodology for classifying and modelling the elements of a power distribution corridor from airborne LiDAR point clouds has been introduced. An initial stage looks for detection of the corridor in the points using binary images which combines different features extracted from the LiDAR data and determined the presence of pylon and wire elements. It differs from similar algorithms in the fact that it does not require an initial ground detection stage, saving on computational cost. Any false positives in the detection are solved by grouping the candidate points and analyzing the distribution of return values and mean intensities of each cluster.

A second stage adds detail to the power corridor classification and generates vector models for all the different elements in the corridor. It identifies the lines and categorizes them into three subclasses: shield conductors, common conductors and chains. It is also able to separate pylons into two types: suspension pylons, from which points belonging to the subclass insulator are identified; and anchor pylons, in which chain conductors can be found. Considering the current state of the art, this seems to be the first work so far in which such a differentiation is applied regarding classification of power corridor scenes. The pipeline also segments the wire points into individual conductors and generate models of each of them, calculating the parameters of its related 3D catenary equation. For pylons, the system determines the number of arms and generates a vector model including a line for the pole and lines for each arms. The points in which conductors hang from the tower are also added to the pylon model.

The results given by the method on a test set with 25 different point clouds are promising, with a recall of 94.90% for pylon classification and 99.58% for wire classification. Furthermore, the pipeline is able to correctly segment and model up to 96.15% of insulator strains, 92.86% of the individual shield conductors, 93.93% of common conductors and 97.22% of chain conductors, with an average error of 0.19 m. between the models and the data, which is accurate enough for the industry standards in terms of maintenance of the power corridor. Moreover, the proposed pipeline was directly compared with a reference method over a set shared by their authors, demonstrating the technique is competitive against the state of the art.

However, there are still drawbacks to be solved in this proposal. The main one is the error generated in the wire modelling, which is higher than in other

state-of-the-art techniques. Much of this the reported error occur in the XY coordinates and come from the fact that a conductor is not always a single wire, but several wires slightly separated and tied together with staple-like pieces. Although the industry considers this a single conductor, an extension of the technique that allows the detection of all the small wires and the staple pieces can be convenient and will improve heavily the accuracy of the model.

Additionally, other new lines of work and collaboration in the field of power line management could surge from the work described in this chapter, contributing to the prior detection and solution of potential risks and avoiding energy service incidences and forest fires. Some of them are the detection of objects that could affect the power line from the point cloud data, the inclusion of machine learning techniques in the pipeline to ease the identification of elements in the scene, or the development of an efficient viewer to show the extracted power line and its surroundings with the goal of indicating possible risks and their location.

Chapter 8

Identification and removal of noise artifacts in airborne point clouds

In the previous chapters, the need of point segmentation in order to generate understandable models from point clouds has been exposed and different solutions for extracting ground, vegetation, trees, urban objects and power objects have been proposed. These point segmentation processes, as well as the others in the state-of-the-art, require clouds in which all the points are semantically and spatially correct and truly represent the acquired scene. Although the current LiDAR sensors are capable in most cases of producing accurate point clouds, sometimes points that are not representative of the scene or are incorrectly placed are introduced in the point cloud. Such points are known as noise, and they are generated due to multiple natural and artificial causes.

To properly perform point segmentation, all the noise should be identified and filtered out from the point cloud in the first place. There are many studies dedicated to remove particular noise artifacts from the echo waveform [209], from the points [210], or even programming libraries of point cloud tools with statistical noise filters (LasTools[1], PDAL[2]). Despite of this, in practice most of these tasks must be manually revised, generating cost overheads. In the most severe cases, with high densities of noise points, the problem can lead to reacquiring the whole point cloud, which is not always economically or practically feasible, e.g. the high costs of programming acquisition flights.

In this chapter, a methodology for noise removal is proposed after the identification of different types of noise in one-strip airborne LiDAR point clouds. The filtering pipeline takes the original point cloud and optionally the acquisition data, generates multiple slices to stabilize time execution and solves in parallel each observed noise artifact by exploiting their characteristic reflectivity

[1] https://rapidlasso.com/category/noise/

[2] https://pdal.io/workshop/exercises/analysis/denoising/denoising.html

and spatial features. A junction classifier returns the final noise classification. This methodology is validated against a data set of 20 point clouds that contain many different types of noise, demonstrating its potential for noise filtering.

8.1 Related work

Several authors have proposed methodologies on how to remove noise and artifacts out of the LiDAR output point cloud. Ferdinandov, Tsanev, and Todorov [211] introduced the *Signal to turbulence*, a quality metric for behaviour validation, also commonly called *Signal to noise*. This metric is used afterwards in several works. After this initial step, the works can be divided into two categories: (i) full echo waveform and (ii) noise classification over the raw point cloud.

In relation to the echo-related works, several lines of research have been published. One of them is the use of a semi-automatic pipeline which applies low-pass and high-pass filters over the signal before the manual removal is performed [212]. Lerkvarnyu, Deijhan, and Cheevasuvit [213] proposed the *Moving Average* (MA) technique, in which the output signal is generated using the average of several points in the input signal. Fang and Huang [214] applied *Discrete Wavelet Transforms* (DWT) to extract weak signals which could be missed due to noise. In the work of Reddy et al. [215], *Hard Wavelets* (HW) for filtering noisy echo signals, and Li, Gong, and Ma [209] introduced the *Empirical Model Decomposition* (EMD) methodology for the same objective. A later research conducted by Sarvani, Raghunath, and Rao [216] showed a superior performance of the EMD technique when compared with MA, HW and Fourier transforms for noise filtering in airborne and atmospheric LiDAR point clouds. More recently, Chang et al. [170] proposed an improvement of the EMD technique based on a roughness penalty, and Qin and Mao [217] extended the DWT concept to machine learning with the creation of a wavelet self-adaptive neural network for noise reduction.

About classification of noise points, most of the current works deal with differences in height or intensity or make spatial analyses of the data. An example of this can be seen in the work of Nardinocchi, Forlani, and Zingaretti [218], in which step regions are detected using a 2D height grid generated from the cloud and spatial relations between those regions are used to distinguish between vegetation and noise regions. Nobrega, Quintanilha, and O'Hara [219] applied a diffuse anisotropic filter over a 2D image generated using point intensities, with the assumption that their objects of interest share the same intensities throughout their points, which suggests that acquisition of data was made via vertical flight and distances object-sensor are approximately the same. Liu et al. [188] applied a filter stage for points with abrupt changes in height or intensity prior to power line detection. Zuowei, Yuanjiang, and Jie [220] voxelized the cloud and used *Finite Element Analysis* to determine, based on a 26-neighbourhood criteria, whether a point group should be tagged as noise. Rashidi and Rastiveis [144] removed points with an intensity lower than a certain difference from the

local mean of intensities. Ullrich and Pfennigbauer [210] exposed the concept of range noise and proposed an averaging technique for their elimination. And finally, Cheng et al. [221] presented the unsupervised network *LidarStereoNet*, which aims to correct noise and misaligning issues in point clouds, and at the same time, find the depth of segmented objects in stereo camera images by using the points correctly acquired.

Most of the works in the classification category only focused on filtering the most common issues, normally related to outliers and aerial, very scattered punctual noise. However, other types of more complex noise artifacts can be found due to errors in the installation of the sensor in the aircraft, the sensor itself, delay on the echo acquisition, special atmospheric conditions, etc. These artifacts could cause many hours of manual post-processing or even repeating the acquisition flight, and thus an increase in the economical cost of the LiDAR acquisition and classification tasks. Our contribution is a more detailed study on these noise artifacts, and possible techniques to get rid of them in an automatic manner.

8.2 Case of study

The case of study in this work is a new open data set with 20 noise point clouds that can be found in the work demo webpage[3]. The point clouds represent different power line corridors and have diverse sizes of between 360000 and 2900000 points, and point densities between 16 and 39 points per square meter. Their individual sizes and point densities can be found in Table 8.1.

Cloud	Size	Density	Cloud	Size	Density
1	28086858	23.82	**11**	24341943	20.3
2	28086858	25.57	**12**	24341943	20.13
3	11754595	24.67	**13**	24341943	37.7
4	360605	21.21	**14**	24341943	38.93
5	24341943	19.93	**15**	24341943	37.33
6	24341943	20.17	**16**	24341943	33.93
7	24341943	19.93	**17**	8880053	31.61
8	24341943	19.46	**18**	24341943	31.11
9	24341943	20.14	**19**	24341943	29.47
10	24341943	21.16	**20**	18227923	16.52

Table 8.1: Sizes and point densities per square meter for each of the case study point clouds

The point clouds have been provided by a company specialized in LiDAR flights, came from 5 different helicopter flights and have been acquired with the Riegl VQ-580i and VUX-1LR sensors, mounted over a IMU-FSAS-NG inertial unit. Flights have been performed in three countries: Angola, Panama and

[3] `http://193.145.147.50/noiseDemo/`

Spain, during 2017 and 2018. They have been done with a variable height, between 100 and 300 meters over the ground, and an average speed of 40 knots, in order to minimize the number of strips and avoid potential point overlaps. This is also the main reason to use a helicopter for flight acquisition instead of an airplane. Additionally, a noise ground truth made by experts from the company has been provided for each point cloud.

8.3 Noise case studies in airborne LiDAR point clouds

The first step to be followed in a research about automated detection of legitimate LiDAR echoes and noise points is to identify the different possible scenarios and how undesired points are distributed into them. In this section, the different noise typologies found in the case study are enumerated. According to their visual appearance, they have been classified into seven categories: *scattered*, *arched*, *tubular*, striped, *echo*, *floating blob* and *dense*. Details for each of them are provided in the following subsections.

8.3.1 Scattered noise

It is the most common noise type in LiDAR point clouds and it consists of points that are floating in the point cloud in an isolated manner and have no clear relationship between them or with the rest of the points in the cloud. A good example of scattered noise can be seen in Figure 8.1.

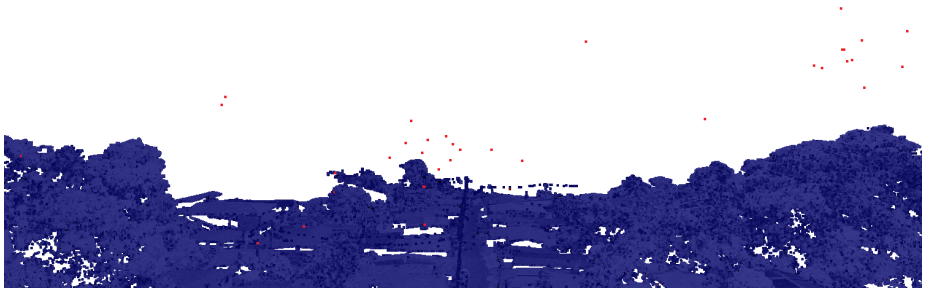


Figure 8.1: Scattered noise (in reddish tones) over a section of a point cloud.

This kind of noise can appear in any place on the point cloud and occurs due to diverse causes. Those causes are both natural, like dust, aerosol gases [127] or flying birds; and artificial, as in the case of spurious sensor misreads.

8.3.2 Arched noise

This noise type is the structured version of the scattered type. Also called *range noise* [210], the points belonging to this category are distributed in a way that resembles those of an arc aligned in the same direction of a singular strip. This is shown in Figure 8.2.

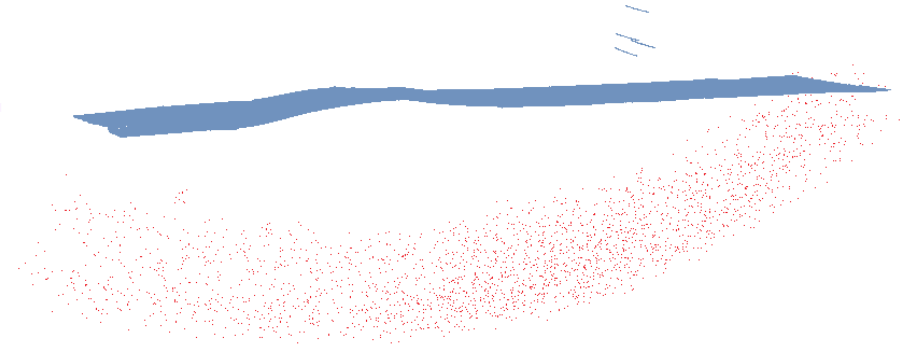


Figure 8.2: An arched noise artifact found in a section of Point Cloud 4 (red points). The artifact is aligned with the acquisition strips direction.

Due to its aligning and geometric nature, this error type is supposed to be related only with sensor misreadings and can be found below, over or even crossing the terrain-representative part of the point cloud.

8.3.3 Tubular noise

The term *tubular noise* refers to dense noise artifacts whose points are distributed with a tubular shape. These noise artifacts traverse the point cloud following a meandering path that is also relative to the position and yaw-pitch-roll angles of the flight vehicle (in this case of study, a helicopter). An example of these noise artifacts is shown in Figure 8.3.

Identically to the arched type, the tubular noise can go over the cloud, below the point cloud or through the terrain. Moreover, in all the point clouds of the case study where this type is present, two distinctive characteristics have been observed:

- **Repetition**: wherever a tubular artifact is present, it is very usual to find one or two similar tubular artifacts in its vertical.

- **High reflectivity**: In the studied cases, tubular artifact points show, disregarding the point cloud, a very high intensity average value. This average value is always superior to 1.5 positive standard deviations over the reflectivity average of the rest of the point cloud. These patterns can be seen in Figure 8.3.

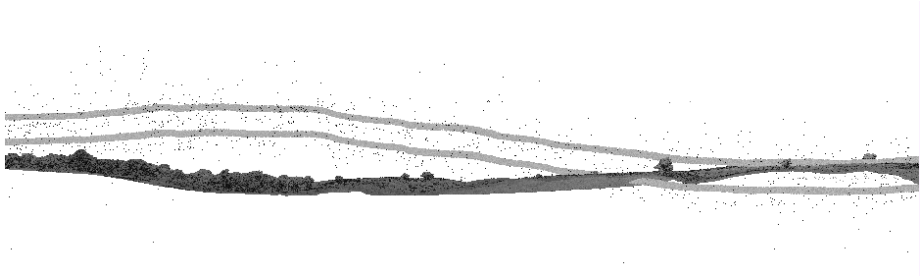


Figure 8.3: Tubular-like artifacts traversing a point cloud, along with some scattered noise. Point reflectivity is reflected in grayscale. High reflectivity and repetitive patterns can be appreciated.

8.3.4 Striped noise

This name has been given to noise clusters disposed in the shape of lines, rays or spirals which could affect the whole or just small areas of a point cloud.

Striped noise artifacts can be subdivided into two main groups. The first one has certain similarities with the tubular type: it traverses the point cloud using a path dependent on the position and angles of the helicopter flight. From now on, it will be called *strand*. The second type of striped noise artifacts tends to be local and has no relation with the helicopter trajectory: it will be called *ray*.

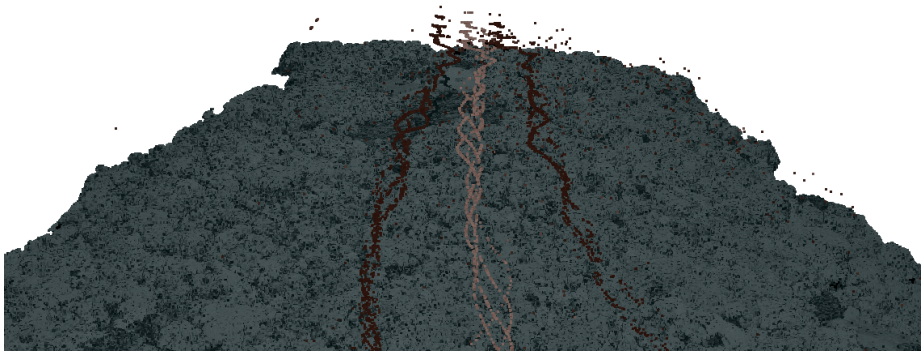


Figure 8.4: Multiple strand noise artifacts affecting a point cloud. They can be seen in reddish colors, darker or lighter according to their reflectivity.

The strands, which are visible in Figure 8.4, differ from the tubular type in their shape (which is a polyline instead of a tubular-shape group) and in the fact that, although they have distinctive intensity and repetition patterns, they admit more variability than the tubular case. Differences can be resumed in the following manner:

- **Repetitions**: more than two repetitions of the pattern can show up ver-

tically stacked . Additionally, the pattern can occasionally be found laterally duplicated with the same repetitions.

- **Reflectivity**: When the strand artifact only repeats vertically (or does not repeat), all the strands present an average reflectivity of at least a positive standard deviation with respect to the average intensity of the rest of the point cloud. On the contrary, when lateral repetitions are found, they present a very low average reflectivity: lower in all the studied cases than two negative standard deviations from the average of the rest of the point cloud.

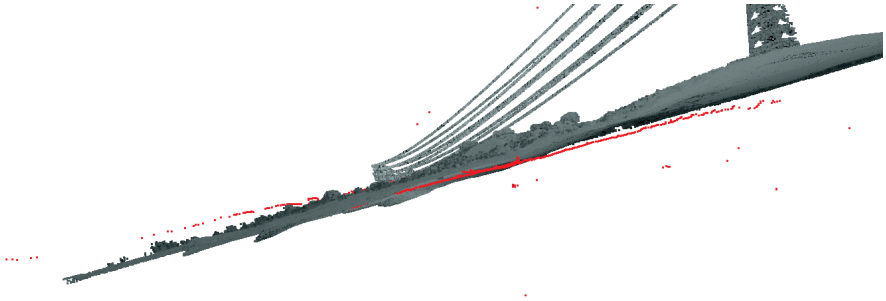


Figure 8.5: Ray noise artifacts (marked with red tone) going through the terrain part of the point cloud.

Rays are arbitrary alignments of noise points (see Figure 8.5) which occur locally in the point cloud. Rays occur in any location: over, below or crossing the terrain-representative part of the point cloud. Unlike strands, rays can also have any reflectivity and do not repeat in the same exact shape in other parts of the point cloud.

8.3.5 Echo-like noise

Noise artifacts are said to be echoes if they are produced by a delay of the sensor on reading a response echo and they generate a representation of the real object in an incorrect position, generally further from the sensor than the expected one. When echo artifacts occur, a lesser density or even no points are also appreciated in the real representation of the object in the point cloud.

Echo noise artifacts are particularly easy to identify and severe for the acquisition process when they affect power line scenes, as in Figure 8.6. Apart from their shape, they are identifiable for their reflectivity, which are still the expected for the corridor and normally higher than the ones for their surrounding points, as the reflectivity in LiDAR diminishes in relation with the square of the distance to the sensor [170].

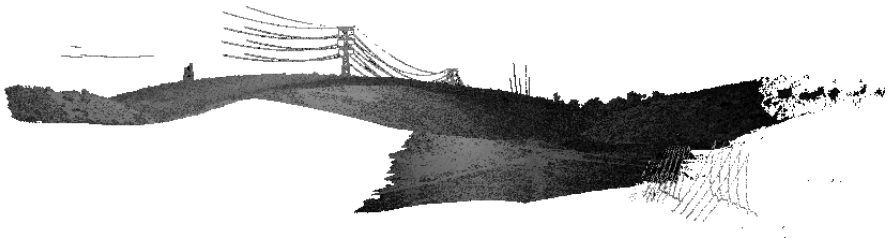


Figure 8.6: Echo noise artifacts (right).The input respects the shape of the line and the reflectivity of the points on the line being inspected, but the position is incorrect. A void is also observed in the affected corridor area.

8.3.6 Floating blob noise

This noise type, previously defined in the work of Santana et al. [222], has the appearance of a floating cloud of particles, like the one in Figure 8.7. Unlike tubular noise, a floating blob is always distributed really close to the GPS position of the sensor in the given moment, does not repeat in other parts of the cloud and does not have any other distinctive feature, nor in reflectivity neither in density. It usually happens due to sensor misalignments in the helicopter that generates false responses.

8.3.7 Dense noise

A cloud is said to have dense noise in cases where one or multiple types of noise are found on it and the amount of noise points per square area is high enough to make the differentiation difficult, even manually, as happened in the case of Figure 8.8. Dense noise could require filters that exploit non-spatial LiDAR features. Although dense noise scenarios rarely occur, it is usually observed in environments with an atmosphere full of reflective particles (water, dust, ashes) or when the acquiring sensor is severely damaged.

8.4 Strategies for the filtering of noise

In this section, different solutions for filtering each one of the abovementioned noise typologies are exposed, and also how to combine them to generate a final noise classification for any input point cloud. Each problem described in the previous section is managed separately, leading to a working pipeline with different stages whose architecture is depicted in Figure 8.9.

In this preprocessing architecture, the inputs are the original point cloud and, when optionally available, the flight data of the helicopter. The first stage aims to solve floating blobs of noise, which requires the flight data input. After it, a

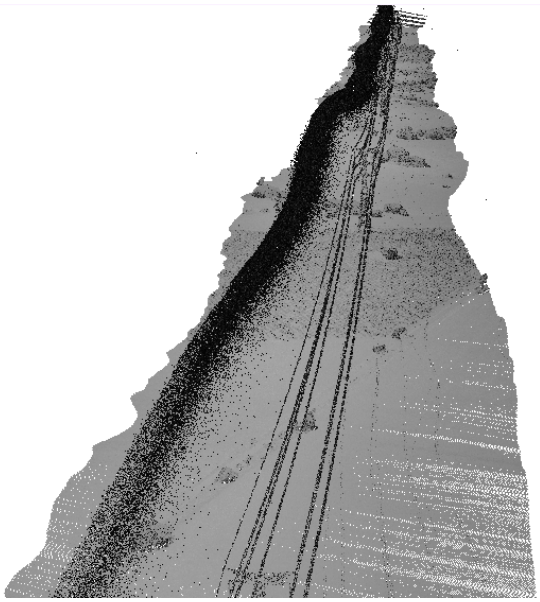


Figure 8.7: Floating blob of noise (in dark, left part of the cloud) parallel to a power corridor inspected using a LiDAR flight.

second stage divides the point cloud into slices and, for each slice, all possible point groups are determined in a coarse manner. Slice information is used thereafter as input for the parallel detectors for the remaining noise typologies, along with the point cloud. Up to five noise detectors can be enabled, and they are focused on subterranean, arched, scattered, echo, tubular and striped noise artifacts, respectively. Finally, a junction classifier combines the predictions and returns as output the point cloud with its final noise classification.

Each one of these stages are detailed in the following subsections:

8.4.1 Floating blob detector

This noise detector looks for proposing a classification of floating blob noise related points by applying a modified version of the algorithm of Santana et al. [222]. This algorithm starts by pairing each point in the cloud with the sensor position at reading time, using timestamps and flight data, and for all the pairs, a 3D Euclidean distance is computed.

A histogram of all sensor-point distances, like those in Figure 8.10, is generated. When a floating blob of noise is found in the point cloud, the histogram will show an initial pick of high density of points close to the sensor, followed by a valley and the main pick of points. On the contrary, a progressive rise to the main peak of the histogram is observed when no floating blobs are found.

The algorithm considers part of a floating blob and therefore filters any point belonging to the initial peak of the histogram by computing the local minimum

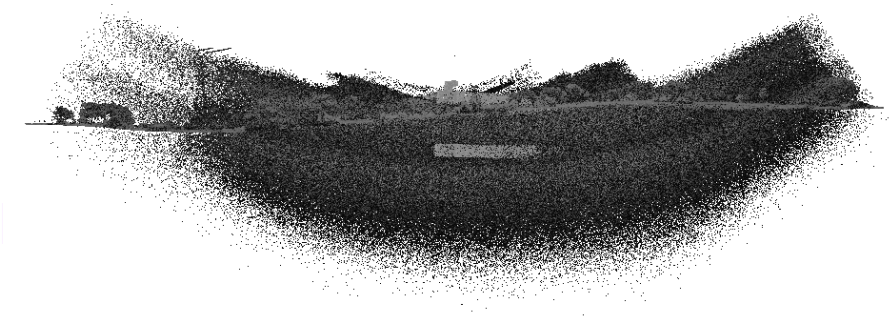


Figure 8.8: Point cloud heavily affected with arched and tubular noise artifacts. The density of noise points is high enough to make pointing out which points are representing terrain or trees hard, even with the human eye.

d_{min} in the valley between both histogram peaks. In our proposal, this remains untouched.

However, Santana et al. proposes the removal of any remaining scattered noise between the blob and the real scene by using the algorithm of Rusu et al. [207]. It suppresses outliers using deviations with respect to the mean of point-to-point distances as basis. This technique is useful for such a task, but it was designed to be applied in small clouds representing interior areas and it makes point-to-point calculations for all the points in the cloud. This fact becomes a great weakness in terms of computational cost when used against heavy sized, high density airborne point clouds, as in this case of study. To avoid this situation, this part of the original algorithm is ignored, and any possible scattered noise remaining in the point cloud will be later processed by the scattered noise detector of the pipeline.

8.4.2 Point cloud partitioning: the slice generator

This is a control stage which aims to extract valuable spatial information about the point cloud which can be used in detectors and at the same time maintains the computational time stable for high volumes of data.

In natural and urban scene point clouds, the most valuable information is to find the approximate position of the terrain level or, at least, of elements which are obviously related with the real scene. If spatial clustering techniques are applied to the point cloud, the expected result is to have most of the points in the real scene grouped in a unique cluster, whose shape is probably elongated and oriented in the same directions followed in the flight. However, most of the clustering techniques required a previous knowledge of the total number of groups (e.g: k-means), which is normally not possible for point clouds; or paying a high computational cost that make the algorithm impractical for great volumes of work. This is the case of hierarchical clustering methods [123], although they do not require previous group knowledge.

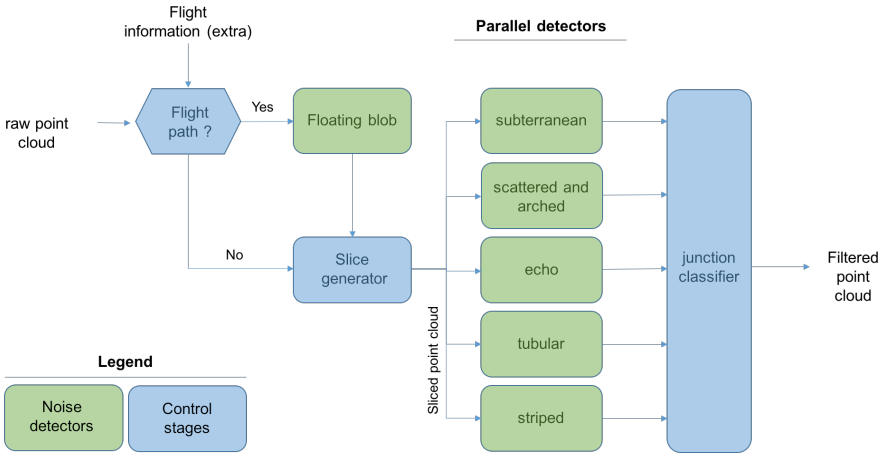


Figure 8.9: Noise filtering pipeline architecture. Noise detectors for the different noise topologies are marked in green. Control stages are marked in blue.

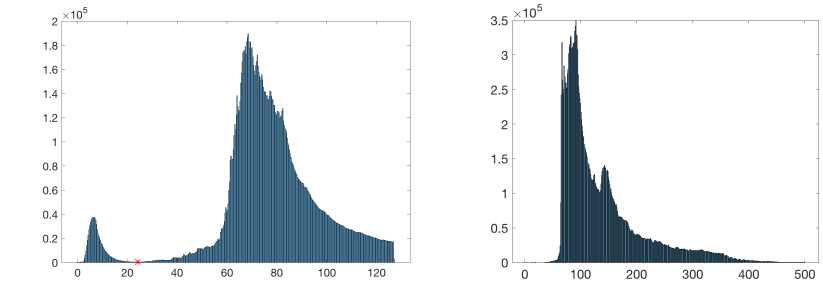
To circumvent this obstacle, the point cloud is sliced perpendicularly to its longest axis, from now on X. The cuts are performed equidistantly along the Y axis a distance ΔX. Hence, the agglomerative clustering algorithm [123] can be applied to each slice instead of the whole point cloud, resulting in groups like those of Figure 8.11.

ΔX must be chosen so it is large enough to keep the noise on it identifiable, and short enough to keep the clustering time bounded. Let an example be a point cloud with a spatial resolution of 50 points per square meter and an average width of 100 meters. $\Delta X = 10$ m. generates slices in which terrain and other objects are ensured to be grouped, and at the same time stabilizes in a maximum of half a million points the amount to be processed, which can be done at a maximum time of 2 seconds per slice in a low-end computer.

Another decision to take is the stopping offset, C, for the clustering algorithm. It should be wide enough so points related to real objects are grouped in a singular cluster, but tight enough to separate suspicious points in independent groups. For the previously described example, an offset between 2 and 5 m. works fine for almost all the cases.

The grouping result should be analyzed to determine which of the G_{ij} groups in a slice F_i represent real objects. This is achieved by testing the following length, size and adjacency conditions:

- **Length**: $\left(max\left(G_{ij}^{Y}\right) - min\left(G_{ij}^{Y}\right)\right) > \alpha_1 \left(max\left(F_{i}^{Y}\right) - min\left(F_{i}^{Y}\right)\right)$
- **Size condition**: $sizeof\left(G_{ij}^{Y}\right) > \alpha_2 \cdot sizeof\left(F_i\right)$
- **Left adjacency**: $hausdorff\left(G_{ij}, F_{i-1}\right) < C$ and closest point in F_{i-1} passes at least one condition.



(a) Floating blob - cloud 19 (b) Normal - cloud 5

Figure 8.10: Two distance histograms generated for cloud with and without a floating blob. A secondary peak and a valley are visible in the histogram when floating blobs affect the cloud

- **Right adjacency**: $hausdorff\,(G_{ij}, F_{i+1}) < C$ and closest point in F_{i+1} passes at least one condition.

where the *sizeof* operation returns the amount of points in the set and the *hausdorff* operation returns the Hausdorff distance [223] found between the two sets of points. The length condition looks for groups in the slice which extends longitudinally over the slice: they are supposed to contain terrain and objects. The size conditions look for the groups with the majority of the points in the slice for similar reasons: clusters of terrain and real objects are expected to be denser than those with noise. Length and size conditions depend on factors, α_1 and α_2, relative to the longitude and the amount of points of the slice. Experimentally, both are assigned 0.33 of the total. Finally, both adjacency conditions look for identifying smaller groups representing objects that continue in adjacent slices. That could happen for tree and rooftop clusters, for instance.

A group represents the real scene when at least one of the four conditions is fulfilled. Otherwise it remains unmarked. Whether their content is noise or not is to be determined by the next stage: the noise detectors.

8.4.3 Subterranean point detector

This detector is dedicated to finding noise artifacts of any kind happening below the terrain level. It follows these stages:

1. Create a grid, H, in which each cell represents a square meter of horizontal area in the point cloud. It should be initially empty. We assign each cell the minimum point height found in it for the real points.

2. For the rest of points in the point cloud, the ones which lay in cells with an assigned value are taken. If their height is lower than the cell value, the point is considered *subterranean noise*.

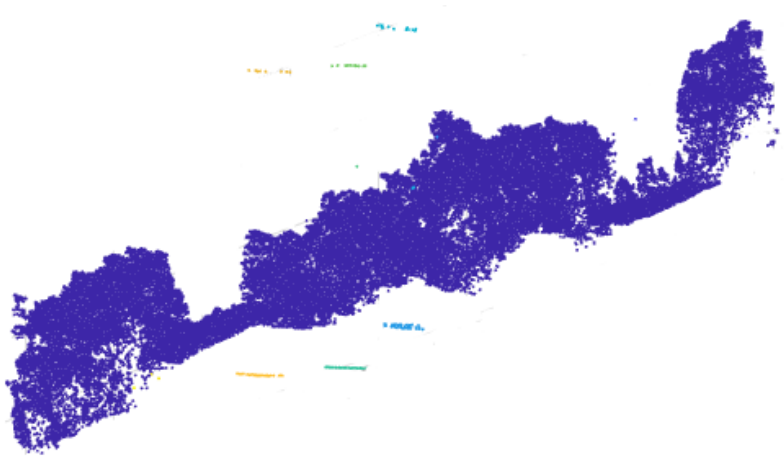


Figure 8.11: Clusters found in one slice. The larger bluish one is as long as the width of the point cloud, so it is not expected to be a noise cluster. Shorter ones (in different colors) could be noise or other objects, like sections of wires, rooftops or trees.

3. For any group G_{ij} of points containing at least a subterranean noise point, the whole group is considered subterranean noise and their points classified in that manner.

The output is a classification of subterranean noise points for the entire point cloud.

8.4.4 Scattered and arched noise detector

Given the slice information and the point cloud, this detector proposes as scattered noise any cluster G_{ij} of points whose total size is 1. For filtering purposes, the arched noise is quite similar to the scattered noise and can be dealt with using the same technique, taking into account the density of the arched artifact when selecting the proper clustering C factor in the slicing stage.

8.4.5 Echo detector

This stage finds echo noise artifacts by exploiting the fact that reflectivity is dependent on the distance to the reflectant point [170]. An echo point usually has an intensity similar to the expected for the object it is supposed to represent, as commented in Section 8.3, and higher than the expected for the position it finally had.

Finally, for most cases the echo phenomena occurs below the main point mass and it could also be removed using the subterranean detector, so the echo detector looks for filtering echo artifacts places at the borders of the point cloud,

where the terrain level might be unclear and the surrounding points have the lowest possible intensity values. The proposed steps to do so are:

1. Generate a grid, H, analogous to the one in the subterranean detector.
2. Select as candidates all the points p in the point cloud which lay in unassigned cells of H.
3. For all the candidates p, their average reflectivity R_p and the standard deviation S_p are computed. The average reflectivity of all the non p points, R_{scene}, is also computed. A point i in p could be considered echo noise when it fulfills the conditions:

$$R_i > (R_p + S_p)$$

$$R_i >= R_{scene}$$

Those point clusters containing a majority of points that fulfill the conditions will be considered echo noise as a whole and all their points will be classified into this category.

8.4.6 Tubular detector

In order to detect tubular noise artifacts, the tubular detector exploits their high reflectivity feature in the following manner:

1. For all points marked in the slicing stage as belonging to the real scene, their average reflectivity and the standard deviation of their reflectivity values are computed.
2. The same is computed for each one of the non-marked groups, per group. A group is proposed as a candidate to be part of a tubular artifact if their reflectivity average is above 1.5 positive standard deviations with respect to the real scene average.
3. Another particular condition of tubular noise artifacts is that they may traverse the terrain, meaning that there could be points in the real scene groups that could belong to the noise artifact, if existing. A preventive test is applied to detect and remove such points. Again, the average and the standard deviation of reflectivity is computed, in this case for all the already existing candidates. A very high average R_{tube} and a small deviation S_{tube} are expected. In the points marked as real, those whose intensities move in the range $[R_{tube} - S_{tube}, R_{tube} + S_{tube}]$ are also selected as candidates. All the candidates are then grouped by the same clustering algorithm used in the slicing stage, with identical settings. All groups generated containing original candidates can be considered as tubular noise, discarding the rest.

4. All the noise candidate points N in Step 3 are the output of the stage if and only if their number of points satisfy the expression:

$$sizeof(N) >= \Delta N^X \cdot 2 \cdot widthS \cdot repsS$$

where *widthS* is a width factor for the tube and *repsS* is the expected number of tube repetitions in the point cloud. As a default, they are assigned 5 m. and 2, as they prove to work well for the studied dataset, but they can be modified to fit the scenario. This last step tests the repetition and continuity patterns identified for the tubular noise.

8.4.7 Stripe detector

This detector also exploits the extreme reflectivity values of some stripe noise artifacts. The detection algorithm is analogous to the one exposed for the *tubular detector*, except for two main differences:

- In Step 2, the points are selected if they have a reflectivity value higher than a positive standard deviation with respect to the average, or lower to two negative standard deviations.
- Step 4 is suppressed, and the results of Step 3 are suggested directly.

8.4.8 Junction classifier

This final control stage generates the final classification as the union of all the parallel-generated predictions. Optionally, it can separate noise predictions into safe and ambiguous: they are considered ambiguous when the points were also marked as belonging to the scene by the slicing stage. This report eases the intervention of a human expert in case of false positives. However, for validation purposes, all of them are considered noise.

8.5 Validation

In order to validate the behaviour of the exposed noise removal methodology, tests have been run over all the point clouds included in the open data set described in Section 8.2. Noise predictions have been compared with the provided expert-made ground truth and quality measures have been extracted: *recall*, *precision*, and *F1* scores. Details on how to compute this scores are provided in Annex I. The results obtained for the test are introduced in Table 8.2.

Promising conclusions can be extracted from the test. The overall noise classification has a completeness rate of 96.61%, a precision of 97.88% and a quality score F1 shows 97.24%. For the non-noise class, the three scores are in numbers over 99.98%. This seems to confirm the applicability of the methodology for noise filtering in airborne LiDAR point clouds.

However, it can be individually seen that the results of the clouds 18 and 19 differ from the overall results, with worse results of 73.8% and 63.9% in F1.

	TP	FP	FN	Recall	Precision	F1
1	14704	150	24	0.9895	0.9983	0.9939
2	49761	56	758	0.9850	0.9989	0.9919
3	74637	206	751	0.9741	0.9972	0.9856
4	2462	0	86	0.9662	1	0.9828
5	243172	2093	14871	0.9424	0.9915	0.9663
6	164380	8474	8985	0.9482	0.9510	0.9496
7	212752	9217	10864	0.9514	0.9585	0.9549
8	209769	5393	18782	0.9177	0.9749	0.9454
9	173657	2027	6163	0.9706	0.9902	0.9803
10	148869	9888	10467	0.9432	0.9461	0.9446
11	148669	11281	9254	0.9414	0.9295	0.9354
12	190045	411	2372	0.9877	0.9978	0.9927
13	45586	644	948	0.9796	0.986	0.9828
14	31563	533	487	0.9848	0.9834	0.9841
15	29206	1552	1893	0.9391	0.9495	0.9443
16	26772	2651	949	0.9658	0.9099	0.9370
17	15295	64	266	0.9829	0.9958	0.9893
18	1559	528	578	0.7295	0.7470	0.7382
19	358	317	86	0.8063	0.5304	0.6399
20	750593	621	975	0.9987	0.9992	0.9989
Totals	2587639	56106	90789	0.9661	0.9788	0.9724

Table 8.2: Final results obtained for each one of the point clouds of the open data set for noise classification.

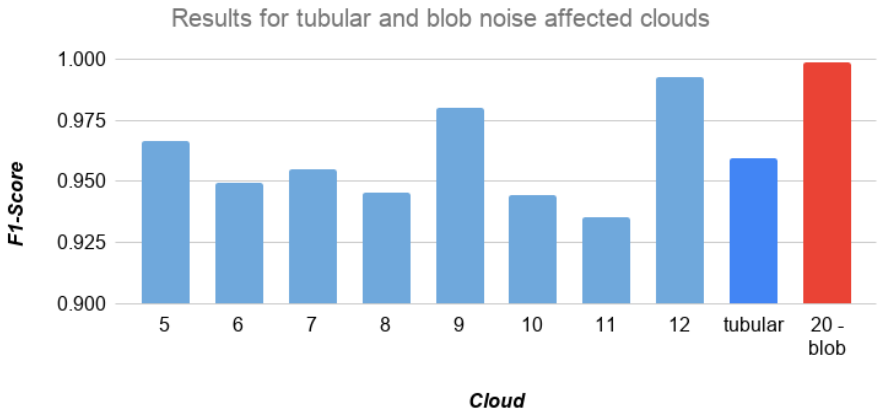


Figure 8.12: F1 scores for all the tubular and blob noise affected point clouds. The overalls for each type are marked in highlighted colors.

	1	2	3	4	5	6	7	8	9	10
subterranean	×	×	×	×		×	×	×		×
scattered	×	×	×	×	×	×	×	×	×	×
tubular					×	×	×	×	×	×
striped										
echo										
floating blob										
dense		×	×							
	11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
subterranean	×		×	×	×	×		×	×	
scattered	×	×	×	×		×	×	×	×	×
tubular	×	×								
striped			×	×	×	×	×			
echo								×	×	
floating blob										×
dense										

Table 8.3: Noise typology per cloud. All the point clouds present at least two of the studied noise scenarios.

The referred classes also differ in amount of noise with respect to similar sized point clouds, which makes necessary an analysis per subcategory. This will take into account the noise typology present in every point cloud, which is offered in Table 8.3.

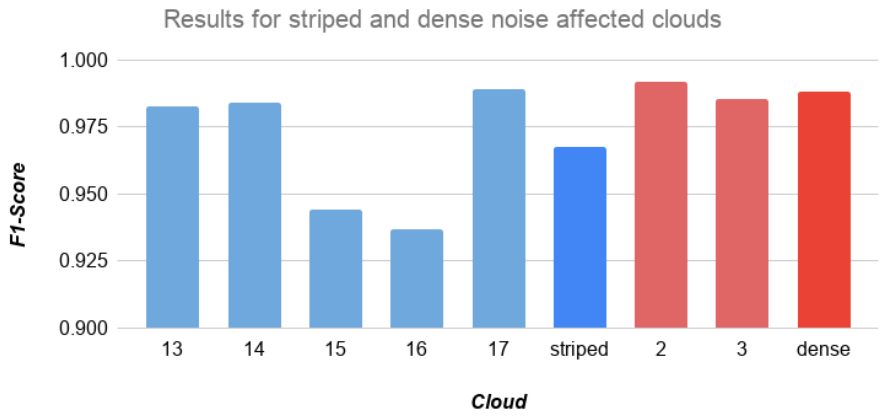


Figure 8.13: F1 scores for all the stripe and dense noise affected clouds. The overalls for each type are marked in darker colors.

Each point cloud presents two or more of the studied noise types. Scattered and subterranean noise are present in 19 and 15 point clouds, respectively, being

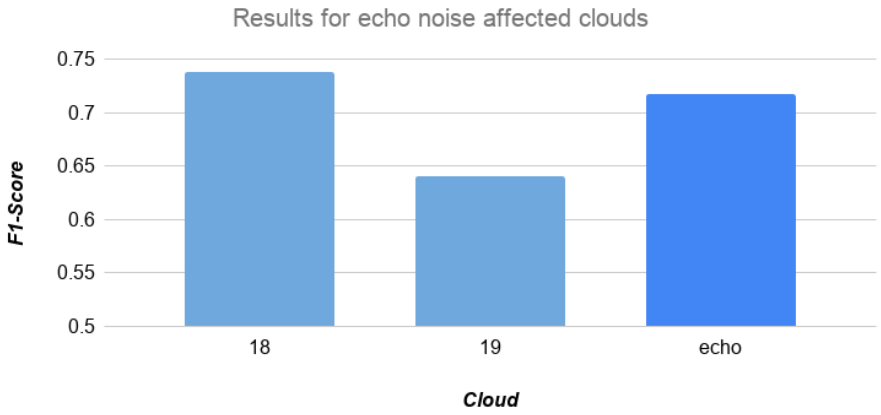


Figure 8.14: F1 scores for the echo noise affected point clouds. The overall is marked in darker color.

the most common problem to deal with. Tubular, striped, echo, floating blob and dense scenarios do not occur at the same time, but tend to be associated with one of the different flights from which the point clouds were acquired, and they are always accompanied by common noise. Particularly, the above-mentioned clouds 18 and 19 are the only ones affected by echo noise.

In the subcategory detail, the overall F1-score achieved for the clouds that include tubular noise is 95.95% , with all the point clouds involved ranging between 93.5% and 99.2% in this quality measure, as it is seen in Figure 8.12. Similar results have been found for the point clouds that include line noise, whose overall F1 score is 96.7%, with a minimum of 93.7% for cloud 16 and a maximum of 98.93 for cloud 17, as shown in Figure 8.13. From both referred figures can also be seen an even better behaviour for clouds with dense subterranean and scattered noise, with all the point clouds scoring at least 98.5% in F1, and for floating blob noise, with an F1 score of 99.89%. For all the types of noise scenario, a sample figure of the final prediction is given in Figure 8.15.

This is found not to be the case for the echo noise point clouds (see Figure 8.14). As commented, both clouds performed worse than the rest of cases, with an overall F1-score of 71.76%. Those are clouds only slightly affected by noise, but in which echo, subterranean and scattered cases can be found. The subterranean and scattered noise are common in all the point clouds and uniquely represented in the two dense scenarios of the open set. As they are filtered with good accuracy for the rest of the point cloud, the poorer results achieved in Clouds 18 and 19 could point at the echo detector.

From the visual inspection of the point cloud 18 (Figure 8.15 - e) it can be seen that parts of the echo are well classified and others are false negatives, which means that, for some slices, not all the echo points are reflected in a single group, and failures are scattered instead of being concentrated. A possibility is

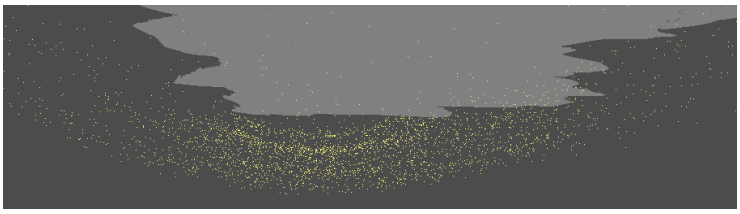
that a methodology only based on intensity averaging is more prone to fail due to one point in the group modifying the average of its group.

8.6 Conclusions and future work

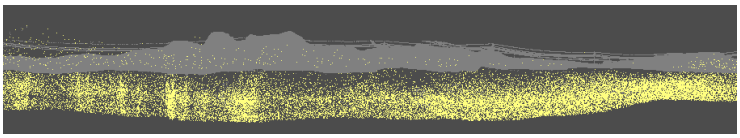
In this chapter, a research on noise artifacts has been performed on a case of study composed of several LiDAR point clouds. Noise artifacts have been identified and categorized according to their visual appearance, resulting in the subcategories scattered, arched, tubular, striped, floating blob, echo, and dense noise artifacts. Moreover, a parallel multistage pipeline has been proposed to filter out the noise. Clustered slices are extracted from the original point cloud to stabilize the execution time and reflectivity, point-helicopter distances and spatial features are extracted and combined from each slice to filter each type of noise in a specialized detector. A junction classifier then generates the final result from each parallel detector.

The achieved results are promising, achieving an F1-score of 97.24% in noise removal for the complete set of 20 point clouds. The same score, when calculated only for point clouds containing the particular tubular, striped, echo, dense and floating blob artifacts, is 95.95%, 96.74%, 71.76%, 98.81% and 99.89% respectively. This demonstrates the utility of the proposed pipeline for the filtering of most of the noise artifacts in a LiDAR point cloud.

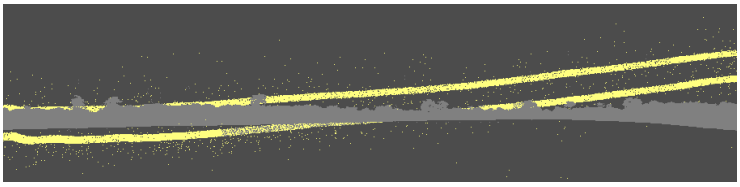
Future steps in this line of work can include improvements in the echo and stripe noise detectors, considering not only reflectivity but also spatial and echo features, and to make an extended validation over a larger amount of noise-affected point clouds.



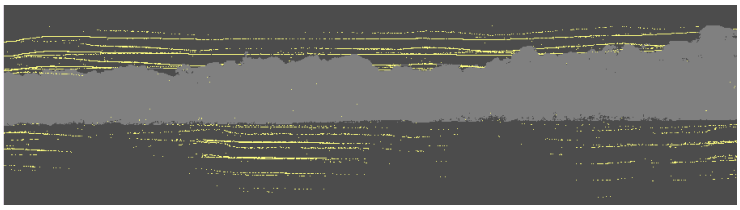
(a) Point cloud 1 (arc-like noise artifacts, in yellow).



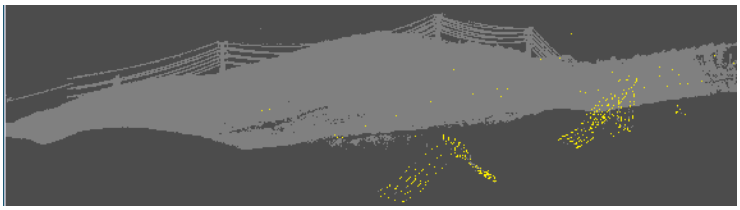
(b) Point cloud 2 (dense subterranean noise, in yellow).



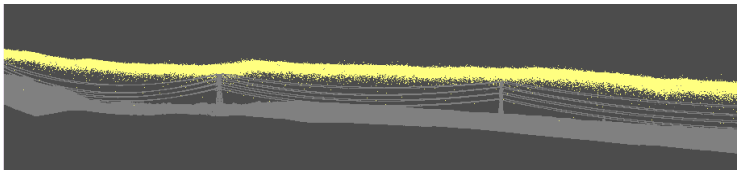
(c) Point cloud 6 (tubular and scattered artifacts, noise in yellow).



(d) Point cloud 15 (stripe artifacts, noise in yellow).



(e) Point cloud 18 (echo artifacts, noise in yellow).



(f) Point cloud 20 (floating blobs, noise in yellow).

Figure 8.15: Predictions for different noise situations in the case of study.

Chapter 9

A lightweight viewer and editor of LiDAR point clouds

In previous chapters, methods for segmenting, characterization and modelling of different classes of objects represented in LiDAR point clouds have been introduced. During the research in those topics, the need for a viewer in which to observe the different obtained results in a quick and friendly manner surged. As the computer used for viewing the results is the same used for generating the results, a light client with the minimum possible resource consumption becomes desirable. This way, reviewing the results in some point clouds can be done as other clouds are processed in parallel.

Moreover, the viewer should have features for the edition of the point classification. Editing allows to create ground truths for the evaluation of the algorithms proposed in this document or, if it becomes necessary, to train models for new machine learning based algorithms. This implies having several tools for selecting all the points inside an area of interest, or only the points from a determined class. Points should also be displayed coloured in base to the already present classification values and also other point attributes, such as intensity. By showing that information, the task of creating a whole new classification from scratch is eased.

Two alternatives had been explored: using already existent viewers and creating a custom point cloud viewer for the daily tasks required in the research. The second one was the chosen option. In this chapter, an analysis of the different tested point cloud viewers is exposed. After that, the proposal of the new viewer is introduced. It includes the decisions that were taken in relation to the camera controls, the LoD strategy and the implemented point selection modes.

9.1 A review on point cloud viewers and tools

This section enumerates several applications and tools which are widely used at the moment of writing this chapter for the visualization and edition of point clouds. Their main features are also described and resumed, in order to have a general view of the state-of-the-art on this topic. However, it should be noted that there could be more point cloud viewers that are not included in the list. Additionally, the viewers in this list can introduce new features and modifications in the future, or can be outdated and removed from the state-of-the-art, so the reader is encouraged to visit the included sources for updated information.

The viewers have been divided into three categories according to the targeted platform: desktop applications, web applications and toolboxes.

9.1.1 Toolboxes

LasView

It is a basic editor of point clouds included as a standalone command-line tool in the LasTools library [224]. It has been implemented using OpenGL and it was designed mainly for manual classification and removal of noise points in .LAS, .LAZ and ASCII formatted point clouds. It allows navigation of the cloud via selection of the preferred action on the keyboard: translation, zoom, pan or tilt. Selection of points can be done one-by-one or by specifying a polygon. The classification value of each selected point can be manually changed to any possible value or cleared. Alternatively, classification processes of the LasTools library can be called.

For a fluid visualization, the viewer performs subsampling of the point cloud in order to fit a point budget. By default, the budget is set to 5 million points, although the user can change this setting freely. Additionally, it is possible to avoid undesired classes to be shown in the scene. Point colors are automatically assigned according to the attribute of the point cloud of interest.

The LASTools library in which this viewer is included is a suite of command lines for classification, tiling, compression, conversion, filtering, rastering, triangulation, clipping and polygon generation from LiDAR data. It was developed by Martin Isenburg and it is distributed in two parts, one of them as open source (LGPL license) and the other is closed and oriented for commercial use. Although it can be used freely, it is more known for their ease integration into GIS platforms such as ArcGIS and QGis than for their standalone use.

Matlab PointCloud Toolbox Viewer

It is mainly a plotting tool included into the Point Cloud toolbox of Matlab [225]. It allows translation, zoom, panning and tilting via selection of the desired action in a camera toolbar, as can be noted in Figure 9.1. Color is automatically generated based on height component of the points, although it allows alternative options based on RGB or user-criteria. Selection can be performed point by point in order to query the data, but edition is not allowed in the viewer. No *level of detail* (LoD) strategy is applied to the figure: it plots the user input as

is. The viewer is not standalone: it requires Matlab to be executed.

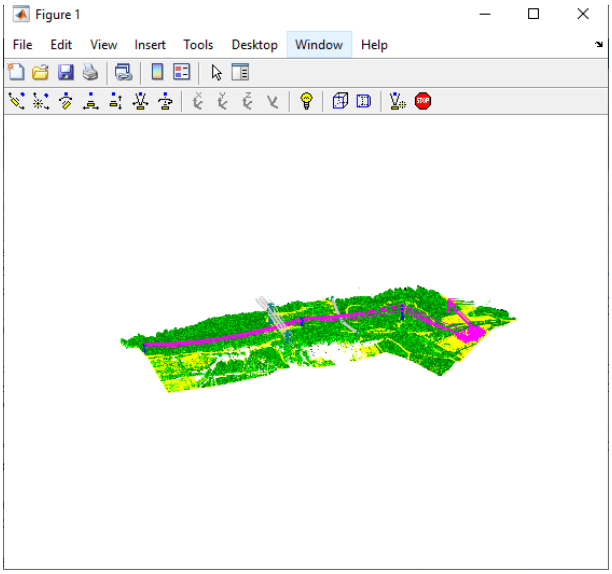


Figure 9.1: Matlab point cloud viewer showing a power line corridor cloud. Navigation controls can be found in the camera toolbar (the lowest).

The Computer Vision and Point Cloud Processing Toolbox, in which this viewer is included, provides reading, writing, downsampling, transformation, segmentation, denoising and shape recognition functionalities. As the toolbox works in the Matlab environment, edition of all the data in the point cloud, triangulation and use of several machine-learning techniques for classification are also available. The toolbox is oriented to PCD formatted clouds, although plugins for other point cloud formats have been designed for the community.

PCL Visualizer

Analogous to the Matlab viewer, it is a plotting tool included into the *Point Cloud Library* (PCL) [226]. The viewer allows mouse navigation in a virtual-globe style and multiple user-customizables ways to colorize points and other PCL algorithm outputs. Selection can be done point-wise, although edition is not available in the visualizer. The visualizer does not perform any LoD strategy either and limits itself to plot what the user requests. The viewer is not standalone but can be called in any custom software that makes use of the PCL Library.

The PCL Library, in which this viewer is included, is the most used toolbox for C++ and Python to process point clouds. It was developed by Rusu and Cousins [227] and includes several algorithms for feature estimation, segmentation, filtering, surface generation and creation of LoD structures such as KD trees and Octrees. It is distributed in the form of libraries for each main operating system and it is free to use (BSD license) for both commercial and

research purposes.

Unreal Point Cloud Plugin

It is an add-on for the Unreal engine [228] which allows to import point clouds and exploit the visualization, illumination and navigation capabilities of that graphic engine. Additionally, it includes capabilities for selection of points, using a polygon as reference input. The selected points can then be edited and reclassified. Point colors can be assigned according to any given point attribute. It also includes functionalities for collision analysis. To improve the performance of the visualization, the plugin relies on a point budget, similar to that in LasView, which can be adapted for the needs of the user. As a default, it is set as 1 million points.

9.1.2 Web applications

Potree

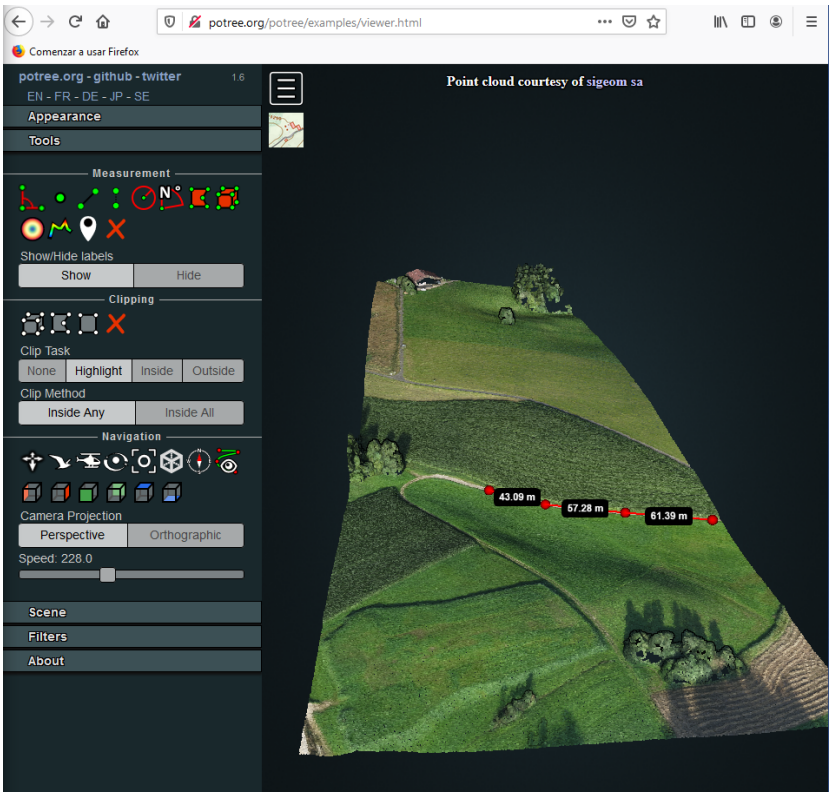


Figure 9.2: Figure 2: Distance measurements between points in Potree viewer. GIS and navigation tools can be seen in the left bar. Source: http://potree.org/potree/examples .

It is a free point cloud renderer based in WebGL whose main goal is to show large, general purpose point clouds. It was developed by Schütz [229] as part of his master thesis in the TU Wien University. It requires a specific Potree format to visualize data. A converter tool is included for the most common point cloud formats.

Navigating throughout the cloud is possible by using several types of controls, including virtual globe, fly and helicopter modes. Point colors can be assigned according to any cloud attribute. Selection is allowed by box (3D) and polygon (aerial view only) in order to do scene clipping. However, editing points is not allowed in this viewer. Smoothness load of large point clouds is achieved by a combination of a level of detail structure based on a local octree [230] and an adjustable point budget. Additionally, several GIS tools are included in Potree, including angle, point distance (see Figure 9.2), area and volume measurements, as well as the possibility to define and export spatial annotations.

Plas.io

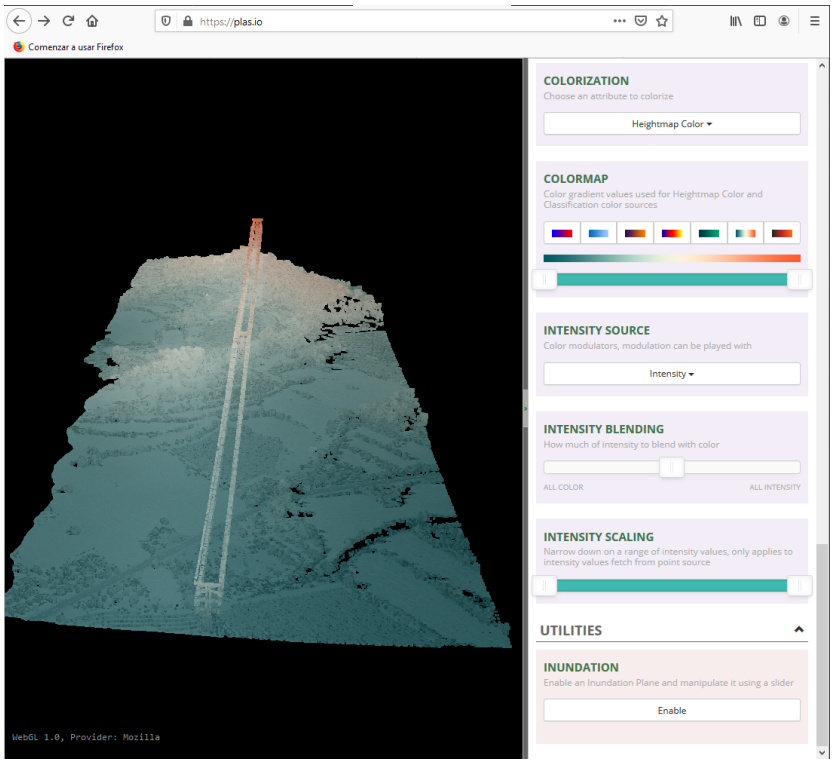


Figure 9.3: Height and intensity colormaps combined to set point colors in Plas.io. Source: https://plas.io .

It is a WebGL-based point cloud viewer focused on loading only .las, and

.laz formatted files, or online sources from the Greyhound lidar repository. It is open source (MIT license) and it was developed by Butler et al. [231] . It allows navigation in a virtual globe style, although the reference point for tilt and pan is the center of the cloud. The main strategy to guarantee fluidness in the visualization is a user-adjustable decimation of the full cloud. Point colors are automatically generated via gradient from RGB, intensity, height or classification features. Two of these colormap attributes can also be blended, as it can be seen in Figure 9.3.

Additionally, point distance measurements, placing of 3D objects over the cloud and inundation planes are other tools included in this application.

9.1.3 Desktop applications

Fugro viewer

It is a free desktop application for viewing LiDAR .las files and combining them with ortho imagery and vector spatial datasets, as well as generating new vector datasets. It is developed and distributed by the Fugro group [232]. The software can represent the cloud in a single 3D view or divided into two 2D views. Navigation is allowed by using common virtual globe controls. Smoothness rendering is achieved with two different techniques: decimation of the cloud and a grid-based LoD structure. Non-desired classes of points can also be made invisible at user request.

The selection of points can be performed individually or using a *transect*: marking a 2D area from which generate a profile in a second 2D area. There, the selection can be performed polygonally. Direct edition of the point cloud is not allowed, although derivative products such as DEMs, contours, points of interest and areas can be generated and exported. Point colors can be assigned according to any attribute of the cloud.

CloudCompare

It is a 3D processing software for the edition of point clouds and triangular meshes. It was developed using C++, QT and OpenGL by the R&D Division of the EDF company [233] over an initial core of Girardeau-Montaut et al. [234]. That initial stage looked for detection of differences between two point clouds or between a point cloud and a mesh. Nowadays, the software also includes several features to perform projections, registration from multiple sources, difference computation, statistics and geometric feature estimations. They also include capabilities for manual and automatic classification of point clouds, based on the method of Brodu and Lague [235].

CloudCompare generates automatically an octree structure from the input cloud, not only for the visualization but also for comparison and feature estimation purposes. Selection is available by specifying a reference polygon or point by point. Point colors can be assigned from their attributes or by features and differences previously computed by the software.

Fusion-LDV

It is a research-oriented, free software for LiDAR and georeferenced data

developed by the Department of Agriculture of the United States of America [236]. It is composed of two separate parts: Fusion, which provides a graphic interface for the presentation and analysis of 2D spatial data, and LDV, which provides a 3D environment for the visualization of LiDAR data. LDV is not standalone: it requires the use of Fusion and an associated raster source in order to load a point cloud.

User interactions with the point cloud and point coloring have been designed and implemented in a similar manner to the Plas.io viewer. Selection of points is available via reference polygon for manual classification purposes. Algorithms for direct classification of terrain, water and trees are also available in the software. Finally, GIS features have been introduced in order to ease the identification and the measurement of height and diameter of the trees.

Terrascan

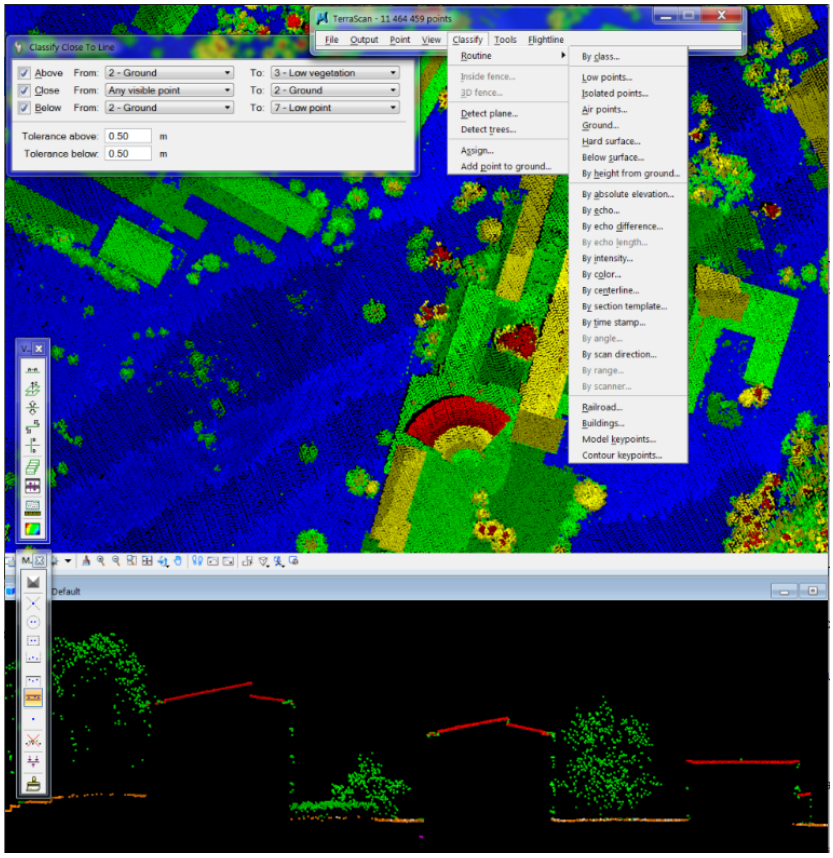


Figure 9.4: Classification flow in Terrascan. A transect view composed of a general aerial view and a selected profile area is shown. Point edition is then performed in the profile view manually or via routines. Source: http://www.terrasolid.com

It is a licensed application for managing and processing LiDAR point clouds developed by the TerraSolid company [237]. It is oriented to commercial scanning campaigns and includes several features for automatic classification of the main point classes, generation of DEM models [132], creation of LoD2 building models or vectorization of power lines, between others. Although it contains a 3D viewer, the program is intended to be used in transect mode (see Figure 9.4), with multiple 2D views which contain aerial and profile representations.

Navigation, point coloring and point selection in Terrascan work in a similar manner to the Fugro viewer. It also uses the same LoD strategies to improve the rendering performance. The main difference with Fugro is the possibility to edit any point attribute, manually or using one of their integrated features.

Unity Viewer

It is a point cloud visualizer developed by Santana, Trujillo, and Ortega [238] using the Unity graphic engine. It is designed for opening multiple point clouds at once. It only accepts a preprocessed cloud format designed by the authors, in an analogous manner to Potree, which includes a binary-tree based level of detail structure for a seamless load of the scene. A conversion tool for .LAS and .LAZ files is also provided.

Navigation through the cloud in Unity Viewer is possible through game-based camera controls: mouse movements affect your center of view and WASD key presses allow you to traverse the cloud. Selection of points is possible individually to query their attributes. However, the edition of such attributes is not available. Point colors are automatically assigned according to the point classification.

3D Reshaper Viewer

3DReshaper [239] is a licensed, industrial-oriented software for processing 3D data distributed by Technodigit-Hexagon which has support for LiDAR point clouds, images, meshes and polylines. User interactions with the cloud are similar to those implemented in Potree, and fluidness rendering is achieved via a user-adjustable point budget. Point colors are set according to the point attribute of interest.

The selection of points in 3DReshaper is based on bounding volumes instead of polygons: boxes and spheres. Edition of points is only available in the full and paid version of the software. Additionally, tools for 3D measurement, comparison between clouds and meshes and creation of labels and vector data are available.

Limon Editor

It is a point cloud editor developed by the Dephos group [240] with support for OpenGL, DirectX and CUDA. It allows up to four visualizers which support 2D and 3D views. Although navigation in the 3D viewer is available via virtual globe controls, the software is oriented to work with multiple 2D viewers, in a similar manner to Fugro and Terrascan. The paid version of the software uses dynamic level of detail with an adjustable point budget similar to the proposed in the Unreal plugin. In its free version, this feature is disabled and a maximum

input size of 25 million points is accepted instead.

Selection of points is performed using the transect scheme of Fugro and Terrascan. Polygon, shape and active intensity and height ranges can be used in the profile view to choose the desired points. Edition and reclassification of points and creation of 3D shapes and vector data are available. Point colors are assigned according to any point attribute. Additionally, features for measurement of volumes, areas, point distances and coordinate conversions are integrated in the software.

A comparative summary of each introduced point cloud viewer in all the analyzed features can be seen in Table 9.1. For navigation, two items are introduced: navigation style (up) and type of pan/tilt interaction (down). Select is an enumerate of all the different techniques implemented for such a task.

Name	Platform	Navigation (Pan/tilt)	LoD	Select	Edit	Colors	GIS tools
LasViewer	Toolbox	1 action Cloud center	Point budget	Point Polygon	Yes	Attributes	No
Matlab PC Viewer	Toolbox	1 action Cloud center	No	Point	No	Attributes, user defined	No
PCL Viewer	Toolbox	Virtual globe Cloud center	No	Point	No	Attributes, user defined	No
Unreal point cloud plugin	Toolbox	Virtual globe Cloud center	Point budget	Polygon	Yes	Attributes	No
Potree	Web	Virtual globe Orbit	Octree	Polygon Box	No	Attributes	Yes
Plas.io	Web	Virtual globe Cloud center	Decimate	Point	No	Attributes	Yes
Cloud compare	Desktop	Virtual globe Orbit	Octree	Point Polygon	Yes	Attributes	Yes
Fugro	Desktop	Virtual globe Cloud center	Grid	Point Transect	Yes	Attributes	Yes
Fusion-LDV	Desktop	Virtual globe Cloud center	No	Polygon	Yes	Attributes	Yes
Terrasolid	Desktop	Mouse 2D actions	Grid Decimate	Transect	Yes	Attributes	Yes
Unity viewer	Desktop	First-person game action	Binary tree	Point	No	Attributes	No
3DReshaper	Desktop	Virtual globe Orbit	Point budget	Point, sphere, box	Yes	Attributes	Yes
Limon Editor	Desktop	Mouse 2D actions	Point limit	Transect Intensity	Yes	Attributes	Yes

Table 9.1: Comparative summary of reviewed point cloud viewers.

Most of the studied point cloud viewers and tools have several of the needed features and some of them, like Cloud Compare or Fugro, have been considered as possible alternatives. However, no one of them fits completely on the objectives mentioned in the introduction, specifically for the needs of a lightweight 3D editor with no input preprocessing requisites and a precise and 3D-wise point selection method for ground truth generation purposes.

9.2 Megavisor: a custom point cloud viewer and editor

In order to have a tool which fulfills all the prerequisites, a custom point cloud viewer and editor is designed. The viewer must have, as a minimum:

- Ability to read and write LiDAR point clouds in .las format

- Seamless visualization of tens of millions of points
- Camera controls in a virtual globe style
- Precise 3D point selection and reclasssification tools
- Point colors assigned via classification or intensity. Classification colors should not be automatic but user-configurable.
- Possibility to hide undesirable point classes in the representation.

Two types of application have been considered for the viewer: a web application based in Javascript and WebGL technologies, and a desktop platform using C++, WxWidgets and OpenGL. The WebGL-based application has the advantage of portability: it can be used in computers regardless of their operating system as well as in mobile devices. However, the main disadvantage comes with the need of point cloud edition: dedicated servers are needed to upload the input cloud, edit it and enable the download of the output results. As such an infrastructure was not available, the desktop application alternative was chosen instead.

The drawing primitive chosen to represent the cloud is the most simple one: the point. When a view change event is triggered, the subset of points to be shown is chosen according to a level of detail strategy and a single OpenGL paint call is executed using the point primitive. Details about user interactions with the point cloud, which trigger the above-mentioned events are provided in the following subsections. The strategies for a level of detail representation and the point selection and edition are also introduced.

9.2.1 Camera and navigation gestures

Three types of navigation gestures associated with mouse events have been integrated in the viewer:

- **zoom**, which is triggered by a mouse wheel move and allows getting further or closer to the point cloud.
- **drag**, which is associated with a continuous click of the mouse left button and allows displacements through the scene while keeping fixed the view direction.
- **orbit**, which is associated with a continuous click of the mouse right button and enables the change of the view direction both vertically and horizontally.

In order to implement these three navigation controls, the first issue to solve is how to initiate them in a discrete point cloud. A common approach in 3D viewers and virtual globes is to find a reference point in the object, the *pivot point*. The pivot point is computed in real coordinates, using as input the click initial position in projected coordinates [33]. To do so, the click is used to

extract the direction vector of a ray that starts in the current camera position, and the object-ray intersection is computed to obtain the pivot point. That point is after used as a reference to apply the desired transformation on the 3D object or on the camera. However, the intersection is not guaranteed to be found due to the discrete nature of the cloud. Unexpected ray intersections with the point cloud can also occur, e.g. when there are overlapping of points in the rendered scene, leading to uncomfortable user interaction.

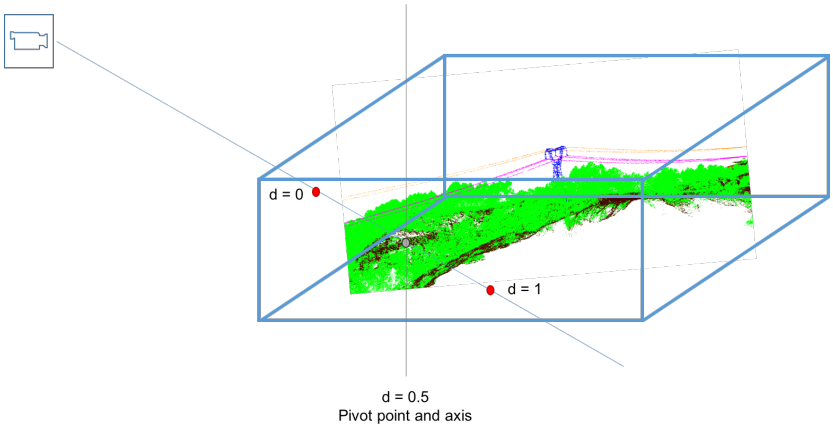


Figure 9.5: Pivot point estimation for the point cloud case. Two intersections (in red) with respect to a bounding volume which encloses the entire cloud are computed. The middle point on the segment between the intersection is taken as the pivot point.

To solve that issue, it is possible to estimate a pivot point by using a bounding volume of the cloud. With a bounding volume, the problem is reduced to finding two intersection points, $d = 0$ and $d = 1$, between the ray and the volume, as it is shown in Figure 9.5. From the ray segment defined by those points, computing the middle point is straightforward. This middle point will be the pivot point for the transformation. Finally, a particular case should be taken into account in which the camera position is placed inside the bounding volume. Only a ray intersection $d = 1$ can be found for such a case, so $d = 0$ is assumed to be the camera position itself.

Once the pivot point is found, the transformation can be initiated. In the following paragraphs, details about to the camera transformations needed for each gesture are provided:

Zoom: For zooming, the ray direction to be applied is $\overrightarrow{OC}$, being O the camera position and origin of the ray and C the position, converted to real coordinates, of the cursor on the screen. This ray is used on the calculation of the pivot point, P, commented previously. The ray enables the computation of the new camera position, O', as the result of the following expression: $O' = O \pm f \cdot \overline{OP}$. There, f is a displacement factor, experimentally set to 0.1. Its

sign, + o − , is to be determined by the movement of the mouse wheel onwards or backwards.

Drag: The ray direction for the purposes of finding the pivot point P should be the camera view direction, $\overrightarrow{V}$. During all the drag interaction, the camera position is updated according to the following expression:

$$O' = O + dx \cdot \overrightarrow{u} + dy \cdot \overrightarrow{v}$$

where $\overrightarrow{u}$ and $\overrightarrow{v}$ are respectively the horizontal and vertical view vectors in the projection plane, and the displacements dx and dy are computed in real coordinates as:

$$dx = \frac{\overline{OP}}{Znear} \cdot \frac{-\Delta X_{mouse}}{0.5 \cdot Y_{screenPixels}} \cdot 0.5 \cdot Y_{screenMeters}$$

$$dy = \frac{\overline{OP}}{Znear} \cdot \frac{\Delta Y_{mouse}}{0.5 \cdot Y_{screenPixels}} \cdot 0.5 \cdot Y_{screenMeters}$$

in which $Znear$ refers to the minimum distance between the camera and the projection plane, in meters, and $Y_{screenMeters}$ is obtained from $Znear$ and the *field of view* (FoV) angle in the following manner:

$$Y_{screenMeters} = 2 \cdot \tan(\frac{FoV}{2}) \cdot Znear$$

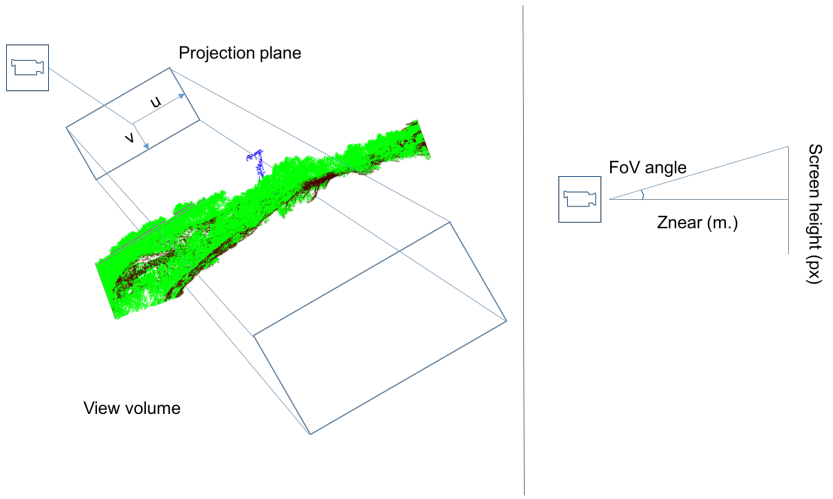


Figure 9.6: Graphical definition of the FoV angle and the horizontal and vertical view vectors.

Having the camera position, the center of the screen (projection plane) and half the total height of the screen, the FoV angle is the relation between the

three of them. A graphic definition of the FoV angle and the view vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ can be seen in Figure 9.6.

Orbit: This gesture is a combination of the *pan* (horizontal orbit) and the *tilt* (vertical orbit) around the pivot point, P. The ray direction for the purposes of finding P is $\overrightarrow{OC}$, analogous to the zoom case. During all the orbit interaction, O and $\overrightarrow{V}$ will be updated according to the following expressions:

$$O' = M \cdot O$$

$$\overrightarrow{V}' = M \cdot \overrightarrow{V}$$

being M a transformation matrix composed of a translation to P, a rotation of ry radians over the horizontal axis, $\overrightarrow{u}$, a subsequent rotation of rx radians over the vertical axis and a final translation opposed to the initial one, in that order. The amounts ry and rx for rotation are defined based on the cursor displacement during the orbit interaction, in the following manner:

$$rx = -\Delta X_{mouse} \cdot \frac{2\pi}{X_{screenPixels}}$$

$$ry_0 = -\Delta Y_{mouse} \cdot \frac{2\pi}{X_{screenPixels}}$$

$ry = ry_0$, when $ry_0 + \arccos \frac{\overrightarrow{V} \cdot \overrightarrow{up}}{|\overrightarrow{V}| \cdot |\overrightarrow{up}|}$ is in range $[0, \frac{2\pi}{3}]$, $ry = 0$ otherwise.

9.2.2 Level of detail scheme

Another problem to be solved is how to manage the visualization of point cloud files with a potentially huge number of points: tens or hundreds of million points. This problem is comparable to those presented in other point viewers [229, 238], virtual globe applications and other 3D-oriented software [30]. The common approach to solve this issue is to create progressive *level of detail* (LoD) representations of the object to be displayed.

Most of the LoD representations require out-of-core processing and are based on binary trees, kd-trees, quadtrees or octrees [22], which can be related to geographical or local coordinate systems. These structures enable an unlimited on-demand load of point clouds with any size and extension from different sources, being this fact their main strength. However, the generated structures could be unbalanced if a point cloud presents density changes or a number of different objects with multiple returns in a small area (e.g. a forestal area): This fact can lead to obvious changes of the level of detail in the displayed scene: e.g. a point area with an open area represented in full detail and a forestal zone represented with a coarser resolution, although both are close to the view. Moreover, the tree generation stages are not practical for lightweight clients, as the one proposed in the objectives of this chapter, where only a point cloud is visited and edited at a given time. An in-core LoD structure based on grid volumes is implemented instead.

When a point cloud file is loaded, a 2D grid is generated so it covers the full extent of the cloud. Their cells are volumes with a squared base of area SxS. All the points in the point cloud are uniquely associated with a cell in the grid according to their XY coordinates. The volume height is equal to the maximum height difference between all the points enclosed in the volume. The choice of the parameter S depends on the point cloud size, dimensions and density, so an experiment is conducted and described in Section 9.3 to determine their ideal size.

A check on the number of volumes to be drawn is conducted during a camera update. In order to do so, the projected area of a minimum bounding sphere that contains each volume is computed, as well as the distance d that separates it from the camera. When that projected area is null due to the sphere being out of the view volume, the grid volume is ignored. Otherwise, d is compared against a minimum distance variable d_{min}, which is updated in case d is lower than d_{min}. Initially, d_{min} is set to the distance $Zfar$ between the camera and the further view clipping plane.

For all the visible grid volumes, the amount of points to be displayed is decided according to the following decimation factor:

$$f_{decimate} = 1 + (\frac{f_{area}}{f_{pitch}} \cdot d_{min})$$

This factor depends on the minimum distance d_{min} between the camera and the visible volumes, as well as on a relation between two other factors:

- **A painting area factor**, $f_{area} = \frac{R}{A}$, resulting from a division between a reference area of painting, R, and the current canvas area A, both in pixels. The factor looks for admitting a larger amount of points to be painted on bigger screen sizes. As this factor depends on a reference area parameter, R , another experiment is conducted in order to find the most appropriate value for it whose results are exposed in Section 9.3.

- **A pitch factor** $f_{pitch} = N - (N-1) \cdot \cos(pitch)$ which depends on the pitch angle and an adjustable importance parameter, N. This factor is introduced since the pitch affects the comprehension of the scene and thus the amount of needed points. In an aerial view, with a pitch angle close to 0, the user only sees the upper part of the represented scene. Therefore, only an amount of points enough to fill each grid cell in a regular manner will be necessary. Adding more points in that situation only results in cluttering and does not add any value. On the contrary, for a profile view case, in which pitch angle is close to $\frac{\pi}{2}$, it is desirable that the user can identify the shape of the different objects in the scene, which normally requires more points, but at the same time cluttering is expected in points that have similar height (e.g: terrain). To find the best value of N for all the cases, an experiment analogous to those made for S and R is also conducted and introduced in Section 9.3.

When a camera update event is triggered, one out of each $f_{decimate}$ points for each visible volume is rendered. Optionally, paint priority can be given to user-defined classes of interest. In this case, decimation of points only affects non-interesting classes and all the points in each visible volume that belong to the classes of interest are painted.

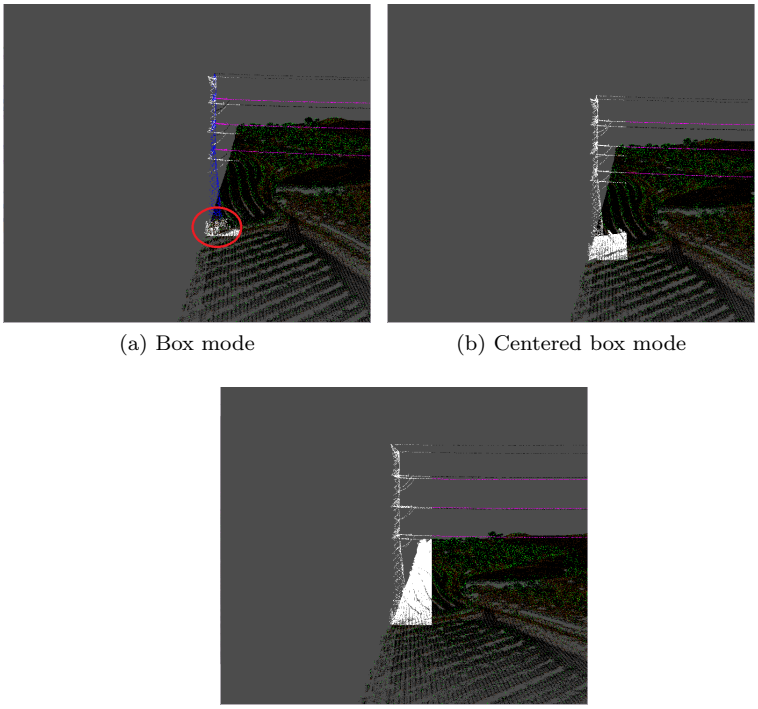
9.2.3 Point selection scheme

A fourth type of user interaction is also implemented to allow the selection of points. Its mechanics consist in choosing one of the three supported selection modes and. After that, a drag interaction with the Shift key pressed generates a rectangular frame. Points inside the frame are the target of a possible selection based on the method of choice. The three implemented modes are *rectangle*, *box* and *centered box*.

- **Rectangle mode**: it is analogous to the *polygon* mode used by viewers such as CloudCompare or Unreal Point Cloud plugin. For each point in the cloud, it is checked whether the point is ahead or behind the view plane. If it is ahead, the point is discarded due to its invisibility. Otherwise, the pixel coordinates corresponding with the point are computed and compared with the chosen frame. If the point lies inside that polygon, the point is selected.

- **Box mode**: The box mode is similar to the one implemented in the 3D-Reshaper viewer and tries to limit the selection only to a short depth near the frame in which the targeted object is placed. It differs from the box method implemented in Potree in the fact that the depth is computed automatically. The mode starts by finding the candidate points with the rectangle mode algorithm, but additionally it stores the distances between each candidate point and the plane of projection. The minimum of those distances is taken as the reference to compute a box front plane. The front plane is then cropped by calculating the intersections between the plane and four rays with origin in O and direction $\overrightarrow{OF}_{1,..,4}$, being $F_{1,..,4}$ the centers in local coordinates of each side of the selection frame. From the cropped plane, the length of the shortest side is computed and used as the box depth. Final selected points are those whose distance is inferior to the sum of the minimum distance point-projection plane and the box depth.

- **Centered box mode**: The original box mode has some unsolved issues when points closer to the viewer than those to be selected are accidentally introduced in the selection window, as it can be seen in Figure 9.7. To solve them, the centered mode introduces an inner polygon of half the size of the selection window and centered in the same place. By using the same procedure, a box center plane is computed instead of a front plane. Front and rear planes are calculated afterwards by adding and subtracting the depth calculated using the shortest side of the frontal plane. The



(a) Box mode (b) Centered box mode

(c) Polygon mode

Figure 9.7: Selection (white) of the points of a power pylon in a point cloud for a selection rectangle which fits the pylon. The box mode worked in an unexpected manner due to vegetation (red circle) which is closer to the viewer. Polygon mode selects everything in the frame and the centered box mode adjusts better the selection on to the power pylon.

final selected points are those inside the selection window whose distance point-projection plane ranges between the distances with respect to the projection plane of the front and rear planes.

The point selection criteria can be modified by the user in the panel of visible and editable classes. As a default, all the classes of points are visible and editable. When a point is marked as non-editable, it will not be selected any longer. Non-visible classes cannot be neither rendered nor selected for edition. An example of this panel and their effects is shown in Figure 9.8.

9.3 Tests and practical results

In this section, the experimentation regarding how to adjust the parameters S for grid cell size, N for importance of the pitch angle and R for reference

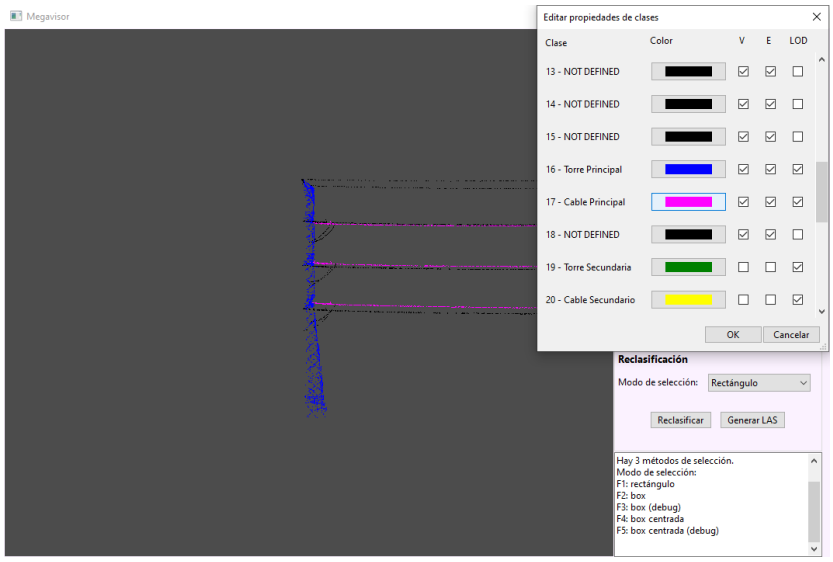


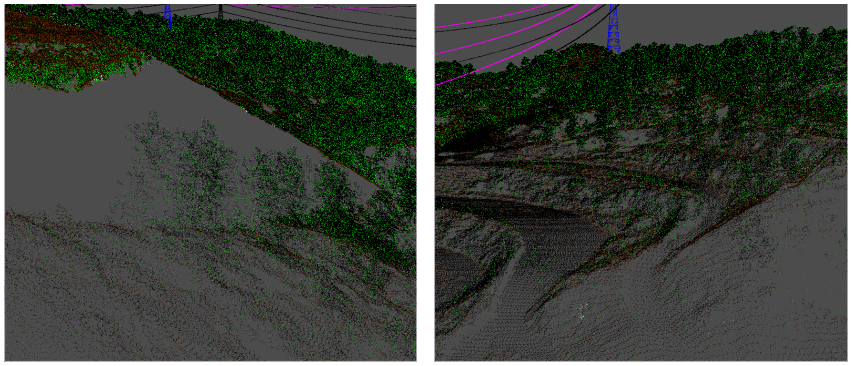
Figure 9.8: Edition and visibility panel of Megavisor over the scene of Figure 7. Classes 13 to 18, referred to the inspected power line corridor, are visible (V) y editable (E). The rest of classes (terrain, vegetation, buildings, etc) have been marked as non-visible and are not rendered in the scene.

drawing area are introduced. The tests have been conducted over two different LiDAR point clouds which represent sections of a power line corridor in a rural environment. Their main characteristics are exposed in Table 9.2.

Cloud	Points	Length(X)	Width (Y)	Average density
Cloud 1	57.1 million	1780.86 m.	2916.59 m.	45.3 p/m2
Cloud 2	10.8 million	906.22 m.	497.77 m.	41.1 p/m2

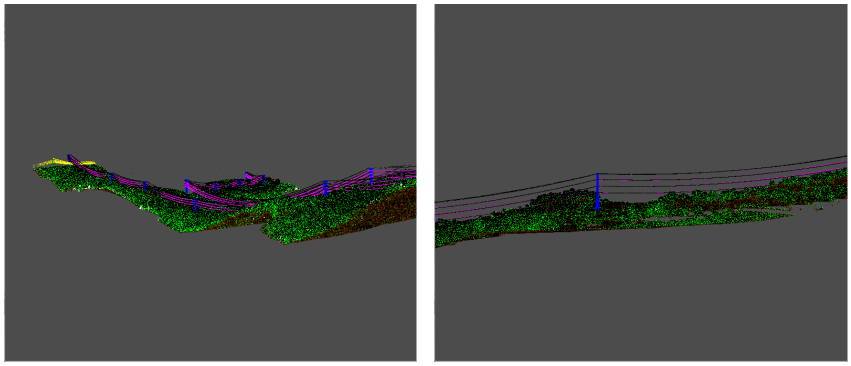
Table 9.2: Main characteristics of the experimental point cloud dataset.

Per cloud, eleven different positions have been selected and the execution time, the amount of points drawn and the screen difference between the full cloud and the LoD strategy are studied in each one of the positions. Ten of the positions are similar in both clouds, with eight 3D positions and two 2D positions. Five positions are close to the rendered cloud and five positions are placed afar. Finally, the eleventh position is chosen arbitrarily in each cloud, showing a close object in detail with other objects in the background of the scene. In Table 9.3, the defined positions and view directions are described. It is considered for the positions that the largest dimension of the cloud equals to the range [-1:1]. For some of such positions, including the two arbitrary ones, their views can also be seen in Figures 9.9, 9.10 and 9.11. The results obtained for each position have been averaged in order to have a general result for each



(a) Cloud 1 (b) Cloud 2

Figure 9.9: Chosen view scenes for the position 11



(a) Cloud 1 (b) Cloud 2

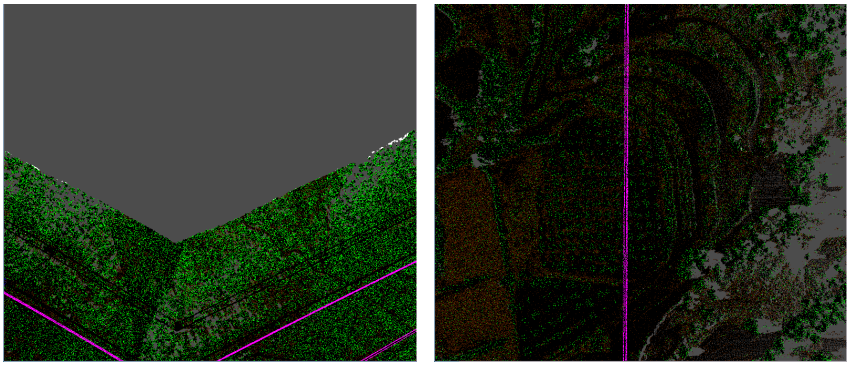
Figure 9.10: Chosen view scenes for the position 1

cloud.

In order to conduct each test, only a parameter is changed while all the others remain fixed (mono-objective optimization). Moreover, the canvas size of the viewer is the same in all the experiments: 784x679 px. The machine in which the tests have been done was a Mac Mini (late 2015 edition) with a Windows 10 running over Bootcamp as the operating system.

9.3.1 Grid cell size, *S*

From the achieved results in both clouds, which are presented in Tables 9.4 and 9.5, it can be seen that the volume size has little impact on the execution time. The average execution time of a frame was, for all cases in both clouds, lower than the 16.66 ms needed to guarantee a fluid visualization of 60 FPS.



(a) Cloud 1 (b) Cloud 2

Figure 9.11: Chosen view scenes for the position 9

Name	Position	View
Common 1	[1.5,0,0]	[-1,0,0]
Common 2	[-1.5,0,0]	[1,0,0]
Common 3	[0,1.5,0]	[-1,0,0]
Common 4	[0,-1.5,0]	[1,0,0]
Common 5	[0.5,0,0]	[-1,0,0]
Common 6	[-1.5,0,0]	[1,0,0]
Common 7	[0,0.5,0]	[-1, 0, 0]
Common 8	[0,-0.5,0]	[1, 0, 0]
Common 9	[0,1.632,0]	[0.239,-0.97,-0.028]
Common 10	[0,0.632,0]	[0.239,-0.97,-0.028]
11 - Cloud1	[-0.239,-0.018,-0.324]	[0.879,-0.298,0.370]
11 - Cloud2	[0.115,0.008,0.138]	[0.952,-0.189,-0.236]

Table 9.3: Normalized positions and view directions for each studied scene.

As expected, the number of points to be displayed grows as S grows, but the increase only started to be really significant for cell sizes larger than 100 meters.

In terms of visual similarity, the closest similarity with respect to a full visualization of the cloud without LoD techniques is achieved in Cloud 1 for 40 divisions of approximately 72 meters large. The shortest tested sizes of 64 and 58 meters have only slightly worse results, so it could also be considered as an option. In Cloud 2, however, the closest similarity is found for the largest tested size (20 divisions of approximately 45 meters large). This fact suggests that the appropriate volume size is more dependent on the cloud density than on the cloud extent and it should be the same for the two tested clouds regardless of their dimensions: approximately 70 meters large for a cloud density of around 45 points per square meter.

	Longitude (m)	Average time (ms)	Average points (thousands)	Average screen diff. (kilo px.)
20	145.83	13.3	3031.7	15.78
25	116.66	12.8	2649.1	15.76
30	87.22	**12.3**	2585.6	15.21
35	83.33	**12.3**	2614.2	14.72
40	72.91	12.6	2548.7	**14.20**
45	64.81	**12.3**	2518.0	14.27
50	58.33	12.4	2481.3	14.23

Table 9.4: Results in terms of average time, number of points and screen differences for the different voxel sizes tested in Cloud 1.

	Longitude (m)	Average time (ms)	Average points (thousands)	Average screen diff. (kilo px.)
20	45.31	7.4	2557.0	**6.50**
25	36.25	7.2	2414.1	7.07
30	30.21	**7.1**	2401.3	6.99
35	25.89	7.2	2355.7	6.79
40	22.66	7.2	2350.5	6.84
45	20.14	**7.1**	2345.1	6.64
50	18.12	**7.1**	2326.7	6.63

Table 9.5: Results in terms of average time, number of points and screen differences for the different voxel sizes tested in Cloud 2. In both clouds, $N = 20$ and $R = 3$ Mpx is assumed.

9.3.2 Pitch importance factor, *N*

N	Average points (thousands)	Average screen diff. (kilo px.)
5	3438	**7779**
10	3068	8533
15	3039	8573
20	3029	8985
25	2976	8539
30	2938	9220
35	2929	8878

Table 9.6: Influence of the variation of the pitch importance factor, N, in the amount of displayed points and the screen differences for Cloud 1.

The obtained results, which can be observed in Table 9.6 and 9.7, suggest that the influence of N in the screen error is linear and it is convenient to maintain this factor low (less or equal to $N = 20$) for minimizing this error. The

N	Average points (thousands)	Average screen diff. (kilo px.)
5	2196	**2018**
10	2104	2157
15	2087	**2018**
20	2054	2069
25	2038	2311
30	2035	2505
35	2032	2453

Table 9.7: Influence of the variation of the pitch importance factor, N, in the amount of displayed points and the screen differences for Cloud 2. For both clouds, S=35 and R=3 Mpx have been assumed during this test.

variation in the amount of displayed points is opposed but tends to stabilize as N increases. Hence, a compromise can be found between them by setting N in values between 10 and 20.

9.3.3 Drawing reference area, *R*

R (Mega px.)	Average points (thousands)	Average screen diff. (kilo px.)
1	2921	2634
1.5	2928	2440
2	2947	2442
2.5	3008	2412
3	3017	2069
3.5	3023	2206
4	3029	2131

Table 9.8: Influence of the parameter R over the amount of points to be rendered and the screen differences in Cloud 1.

The results of the experiment, which can be seen in Tables 9.8 and 9.9, do not show a great impact of R in the visualization. In Cloud 1, the minor screen differences are observed for the lowest and the highest tested values (1 and 4 Mpx), with a peak between them. In the second cloud, the results point to an inverse relation between the screen differences and the R parameter. For the amount of points to be drawn, a linear increase is denoted as the R parameter increases. According to this data, the best setting of R for the tested clouds is $R = 4$ Mpx. This value provides a better visualization that compensates a slightly worse amount of points to draw and execution time.

R (Mega px.)	Average points (thousands)	Average screen diff. (kilo px.)
1	2025	2634
1.5	2032	2440
2	2035	2442
2.5	2038	2412
3	2054	2069
3.5	2061	2206
4	2087	2131

Table 9.9: Influence of the parameter R over the amount of points to be rendered and the screen differences in Cloud 2. For both clouds, $S = 35$ and $N = 20$ have been assumed.

9.4 Conclusions

In this chapter, a new lightweight point cloud viewer, Megavisor, has been proposed. Virtual globe-like camera controls have been implemented in the viewer with an adaptation to take into account the discrete nature of the point cloud. To avoid preprocessing of the cloud prior to its visualization, and at the same time ensuring a stable scene rendering time, a dynamic LoD strategy is introduced. This strategy is based on the dynamic decimate of points inside a grid of squared volumes. The decimation depends on three parameters, S, N and R, whose settings are analyzed via experimentation.

Three modes for automatic selection of points inside a window chosen by the user have been implemented: rectangle, box and centered box. The centered box strategy is novel and it is an adaptation of the box strategy to better adjust the selection to the object of choice, regardless of any undesired point closer to the viewer which is projected inside the selection window. The rectangle and box strategies are well known and introduced in other viewers of the state-of-the-art but they present some issues in this particular aspect, respectively selecting more and less points than expected.

From the experiments conducted over two different point clouds to determine the best parameter settings for the LoD strategies, some initial conclusions have been extracted. The first one is that the size of each volume should not be adjusted based on the cloud extent but on their average density. The ideal size of S resulted to be quite similar for two clouds with very different extent but a similar point density. The parameter N has a linear influence on the amount of points to be painted and in the visual similarity with respect to painting the full cloud. For the tested clouds, the best option is to keep that parameter in a range between $N = 10$ and $N = 20$. Lastly, it does not seem to be a clear influence of R on the similarities between decimated and non-decimated visualization. However, the best results have been achieved for a high value of R, $R = 4$ Mpx.

Chapter 10

Conclusions and use cases

At the beginning of this dissertation, the main objective of the work was defined as the proposition of new methodologies for the processing of large geo-referenced vector data sets. These methodologies should be focused on the creation of structures that ease the selective and progressive transmission of such data and improve the comprehension of the data by the user. The proposed algorithms should explore the use of structures that allow different abstraction levels of the data for their progressive visualization in any device. Moreover, at least a streaming architecture adaptable for any desktop and mobile device should be proposed to transmit the above-mentioned structures. Finally, the preprocessing methods should be compared with other existing methodologies whenever possible.

The way in which all these objectives have been addressed is described across the different chapters of this document. In all of them, a review of the state of the art in each covered topic is introduced, including the different alternatives against our methodology can be compared with. The three main types of vector data, points, lines and polygons, have been used in at least one methodology during this research work. Several preprocessing techniques that generate LoD structures or simplified 3D vector models from punctual data sets have been elaborated. An architecture for the transmission of data based in one of such LoD structures has been introduced as well. And finally, studies on finding the best manner of present geo-referenced information have been performed. In the following lines, the key contributions of this doctoral thesis are enumerated:

10.1 Contributions

Firstly, a pipeline has been introduced specifically for the generation of level of detail structures from point datasets that contain potentially Earth-scale information. Due to this particular requirement, the structure is designed based on a quadtree associated to a regular grid of the planet (DGGS), instead of the more common octrees, kd-trees and binary trees used for generation of LoD

on point sets. To create the different levels of detail of the set, two strategies have been used. One of them, the *sorting*, relies on a property of each point used as sorting criteria to distribute the existing points on the parent nodes of the quadtree. This strategy is oriented to progressive streaming, in which more detail is added during navigation without creating any metadata. The second strategy, *clustering*, generates a metadata which saves the existing amount of points in a certain area of the map for coarse representations of the set. This strategy slightly augments the number of data to stream but favours a clean visualization of the data set.

Secondly, a server-client architecture capable of transmitting the previously generated point data structures developed to mobile devices and displaying it as *marker* symbols has been developed. The architecture requires: (i) a servlet for reading the spatial database, (ii) API endpoints to request additional data, and (iii) a mobile client capable of requesting data on demand during the navigation and generating the symbology needed to represent the data. In order to develop the client, an existing virtual globe engine, Glob3 Mobile, has been adapted to include efficient marker rendering features. Moreover, an experimental study has been conducted to determine the right amount of markers to be displayed on screen necessary for the best understanding of the visualization. The results, obtained for an open, global and large point data set and displayed on a mid-range smartphone, suggest that the best choice is to cover approximately a 30% of the screen with content. The quadtree node content should also be limited to a maximum of 8 markers with text information.

The third contribution of this work also looks for achieving a better comprehension of the displayed data. It is a study which tries to determine the best technique to show polyline sets representing underground pipe networks in a virtual globe environment. Taking the state of the art and the Glob3 Mobile engine as references, a distance-based α-blending was implemented and applied to each pipe model. The distance of each pipe vertex to the camera determines how transparent the pipe looks like. In this regard, eight different mathematical functions have been tested for the diffumination effect. Additionally, a virtual globe implementation of the excavation reference method was implemented, and a novel visualization technology for underground data, the *ditch*, was introduced. Ditches are semi cylindrical meshes designed to enclose a pipe. By combining these meshes with an adequate texture, a trench effect which gives the desired depth cues can be achieved. After a user experience survey, it was seen that the method which gives the best depth hints in the test application was the excavation feature, firstly, being closely followed by the α-blending technique with a softsign transition function. The novel ditch visualization technique generates variable sensations: it is quite easy to understand for technicians, but really hard to comprehend for non-technical users. Nonetheless, the three tested techniques outperform the most common reference method: the α-blending with stable transparency value.

A fourth contribution was a methodology for the automatic generation of 3D city building models from LiDAR point clouds and footprint polygons gathered from the OSM open data set. The models generated for each building fol-

lowed the CityGML standard, which considers progressive levels of detail. The level 2 of this standard, which establishes individual polygons for ground, walls and rooftop surfaces, is used for the representation. As any footprint on the OSM dataset can contain one or multiple buildings, a first step in the proposed pipeline aims to identify buildings inside each footprint and separate them when necessary. For each separated building, a new footprint is generated by using a novel corner-based line simplification algorithm. After that, the pipeline determines the category of the building rooftop from five possible options: *flat*, *shed*, *hipped*; *pyramidal* or *complex*. This is needed to generate the best simplification for the rooftop model, and it is done via plane extraction and a rule system. The rule system takes into account the number of planes found, their mutual intersections and statistical variables obtained from the input points. An initial method validation applied to a point cloud that represents a middle-sized city showed promising results for the identification of the different building categories. Finally, the pipeline demonstrates its ability to reduce the volume of data to represent the same content: the final CityGML city model sizes only 8.5% of the original cloud size.

The fifth contribution was a methodology for ground filtering in LiDAR point clouds. This contribution has utilities on the generation of digital elevation models and in works which require point segmentation. It introduces a new concept, the *patch*, which is a cluster formed by points with local minimum height in the cloud. Patch clusters are generated using hierarchical clustering based on distance and a anisotropic filter to give extra relevance to the height dimension. From each obtained patch, statistical descriptors are defined from their points and raster maps of multi-scale features. Using these descriptors, a decision tree is designed for determining whether the patch represents terrain. Using the ground points classified with this decision tree, an algorithm is also proposed to generate triangular regular meshes with progressively coarser resolutions for its use as digital elevation models. The algorithm solves an equation system to find the heights of each mesh vertex that best fit the input points. To avoid extremely high maximum errors, a subsequent stage adjusts the vertices that contribute more to the global error of the mesh. The proposed methodology for ground filtering in point clouds was tested against multiple algorithms from the state of the art in a well-known urban benchmark, obtaining the best results from all the algorithms that use only the point cloud as input.

The sixth exposed contribution aims to find a semantic meaning for all the non-ground points in a given point cloud. Using vehicle-borne LiDAR point clouds as input, a non supervised pipeline was introduced to classify such points into four categories: *building*, *vegetation*, *poles* and *cars*. To do so, an architecture of progressive one-class detectors has been designed in which the input for one detector is the output of the previous one. The detectors make use of raster feature maps that are extracted from the points that fall in each map cell. Such maps enable the determination of the areas of the cloud with potential for the presence of poles and cars. Using clustering algorithms and extracting descriptors from each cluster it is possible to confirm such presence of poles or cars. The remaining points are grouped as well, and their categorization as building

or vegetation is performed using a recursive algorithm for vertical plane extraction. The proposed pipeline was compared with neural network based methods from the state of the art using a vehicle-borne LiDAR benchmark that represents urban scenes. The results showed that the progressive detection of classes with a non-supervised approach is capable of outperforming all the tested neural networks but one, demonstrating its competitiveness for point segmentation of urban clouds.

Continuing this line of work, a seventh contribution is a proposal for the classification and 3D modelling of power line corridor elements in LiDAR point clouds. The preprocessing algorithm starts by searching the corridor in the clouds, which is done via combinations of feature raster maps. Those feature maps include local height, intensity and accumulation information. This search returns as a result an initial classification of pylon and wire points, similar to the one offered by most of the already existing algorithms for this particular task. The pipeline continues by refining the classification in the areas in which wires are tied to each pylon. Additionally, detectors are introduced for the classification of subcategories of power elements: *insulator strains*, which are subcategorized from pylons, and *chains*, *shield wires* and *conductor wires*, which are subcategories of wires. These parts of the pipeline are implemented with clustering algorithms and fitting of points to line equations. Taking into account the previous state of the art, this contribution is the first one done for the differentiation of such elements in LiDAR points, at least to the best of our knowledge. Due to this, the comparison against other algorithms of the state of the art has been done only in the general level. However, promising initial results have also been exposed for the extraction of insulator strains and chains.

In this contribution it is also included an algorithm for the individual segmentation of each wire and pylon present in the corridor and the generation of 3D vector models for each one of them. The generated model for a single conduct contains the four parameters needed to define a catenary curve in three dimensions: an origin point o, two points representing the endings of the curve and a torsion parameter a. o and a are computed using the PSO algorithm for multivariable optimization, using the points of each conductor in the cloud as input. The RMSE error margin of each model obtained this way achieves the industrial requirements established for its use in diverse applications. Moreover, a vector model is created for each pylon after an automatic identification of the number and direction of all its arms. Obtaining these models allow an important reduction of the volume of data necessary to represent the corridor in any viewer. As an example, the three points and the torsion value of a catenary can be sent instead of the hundreds or thousands of points in which the same conductor is represented in the original cloud. Using those four parameters, the catenary can be represented in any level of detail as it only requires to divide the length of the curve in the desired amount of points. An identical solution can be followed for the pole and the arms of a pylon in coarse representations, as they are reduced to a vertical line and n horizontal lines. Finally, the models can be exploited as-is for subsequent anomaly calculations and periodical corridor maintenance tasks.

In order to use noise-affected point clouds for all the previously commented processing tasks, an eighth contribution is done. It identifies up to seven different noise schemes that appear in point clouds according to their visual appearances. After this identification, a parallel processing methodology is introduced for the separate filtering of each noise scheme. In order to stabilize the execution time, the cloud is split into slices. The filtering process is performed in each detector by using clustering processes and descriptors based on the intensity, the spatial position and the distance between each point and the LiDAR sensor. The partial results of each detector are unified into a single noise classification. Finally, a new benchmark that contains 20 noise-affected point clouds has been made open for the public, and the validation has been done using it. The obtained results were promising and demonstrated the usefulness of this proposal for noise filtering in point clouds.

And the ninth and last contribution introduced in this doctoral thesis was a lightweight viewer for point clouds. It surged as a response for the needs of the daily work on the rest of the exposed contributions. Navigation controls adapted for point clouds have been introduced, as well as three methods for selection of points in 3D scenes. One of them, the *centered box*, is novel. Additionally, a LoD scheme was integrated for a seamless and on-demand rendering of the cloud. The scheme is based on a regular grid instead of an octree, as well as it was decided in the marker case. However, as no network functionalities have been required, the coarser representations are being generated on the client via decimation applied in the grid cells. The decimation function takes into account the grid cell size, the pitch angle and a reference screen area to add or remove detail on the scene. Experimentation conducted shows more similarities with respect to a non-decimated visualization when the grid cell size is chosen based on the cloud density, the reference screen area is greater and the parameter which regulates the importance of the pitch angle is kept in a stable range.

To summarize, as a result of this research work the following items have been accomplished:

- Two novel strategies for the generation of level of detail structures on point datasets and digital elevation models.
- Two novel strategies for the generation of reduced 3D vector models from point clouds, with applications on smart cities and power line inspection.
- Four novel and different methodologies for the segmentation in categories of large volumes of data in the form of point clouds. Such categories relate to ground, urban elements, power line elements and noise, respectively.
- An architecture for the progressive streaming of quadtree-based structured point data into virtual globe mobile applications.
- Three studies which analyze the best ways to display geo-referenced information to a user in the form of markers, pipeline networks and point clouds.

Finally, the majority of the here exposed contributions were successfully transferred into diverse applications designed for tourism, educational, research and industrial purposes. Those transferences are detailed in the use cases section that the reader will find immediately after this paragraph. With these transferences, the objectives defined at the beginning of this research work have been, on our point of view, accomplished.

10.2 Use cases

All the research work exposed in this dissertation has been done so the generated knowledge could be easily transferable to companies and entities and be useful for the society. In this section, the different use cases in which such a transference has been made are commented. Transferences were possible thanks to different collaborations between the university and a company or a public entity in which I have the chance of participate in.

10.2.1 Explora Gran Canaria

Explora Gran Canaria [241] is a project which comes from a collaboration between the *Universidad de Las Palmas de Gran Canaria*, the local authorities of the island of Gran Canaria (*Cabildo de Gran Canaria*) and the software company *Singular Factory*. Its objective is to publicize the nature of Gran Canaria by proposing hiking routes and a series of associated sensorial experiencies to the visitors of the island. Such experiencies go from seeing the impressive landscapes of the islands from a viewpoint to test a typical canarian dish in restaurant. They also include knowing the geologic and etnographic history of the island or experience the smells of the endemic flora and the sounds from the surrounding local fauna.

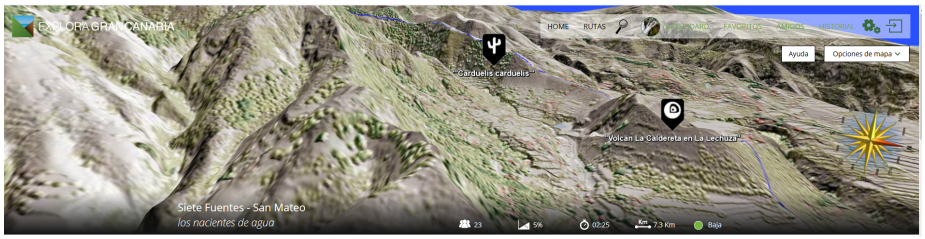


Figure 10.1: Hiking route (blue line) and points of interest displayed on the virtual globe interface of Explora Gran Canaria

The result of this project is a social application for web and mobile (Android/iOS) platforms. The hiking and experience recommendations are generated by a recommender system [242] based on preferences extracted from the user profile. The user can let be carried away by the recommendation or choose his own route. The application shows the full hiking path in a 3D scenario, offers

technical, geologic, and historic information and display different points of interest in the route, at it can be seen in Figure 10.1. For the mobile versions, the app also gathers your location and displays it in the scene. This way, following the path becomes easy and the hiker can relax and enjoy the experience.

The app Explora Gran Canaria relies on the Glob3Mobile framework for the visualization of the hiking path and the points of interest. In this sense, the transferred contribution from this research work to the project is the module for an efficient transmission and visualization of point markers avoiding cluttering, shown in Chapter 2. The raw point of interest data set containing positions, names and icons was preprocessed so a LoD wise spatial database is generated. The data is then visualized using the client proposed in the chapter. As the marker viewer was developed as a part the Glob3Mobile engine, the integration with the rest of the application features is straightforward.

10.2.2 SmartPort

The *SmartPort* project [243] surges from a collaboration from the *Universidad de Las Palmas de Gran Canaria* and the city port authorities *Autoridad Portuaria de Las Palmas* within the context of the FIWARE program of the European Commission. In the SmartPort project, a distributed architecture is designed for the continuous collection of spatial data from various sensors placed in the seaport of Las Palmas de Gran Canaria. A web application has also been developed in order to ease the management of the daily tasks of the seaport based on the collected information.

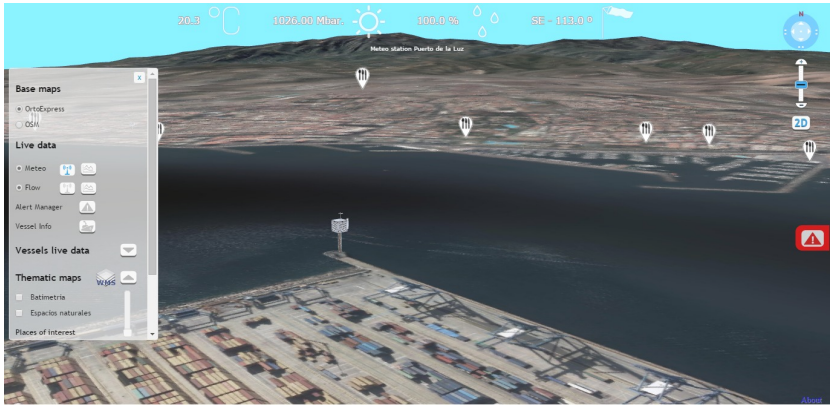


Figure 10.2: A 3D model representing a sensor and some restaurant related point markers shown in the SmartPort web application.

The web interface of SmartPort offers all the information in the context of a 3D scenario of the seaport area. The different sensors, the active vessels in the port and all the monitored containers and cranes are shown in the scene represented as 3D models. The positions and additional data of the vessels are

updated every few minutes by using the AIS standard [244]. For the rest of the sensors, queries to the SmartPort platform are performed on demand.

SmartPort allows the port management staff to use such information to generate and show graphs and statistics about the port. A fuzzy logic based alarm system is also implemented. Additionally, cartographic and point of interest information about the island of Gran Canaria are offered in the web application, as it can be seen in Figure 10.2.

The web interface relies on the Glob3 Mobile framework for the visualization of all the required spatial data. In this sense, the contribution of this thesis to this project is analogous to the one on the Explora Gran Canaria application. The system for an efficient transmission and visualization of point dataset presented in 2 has been integrated in SmartPort to manage the point of interest database.

10.2.3 Epigraphia3D

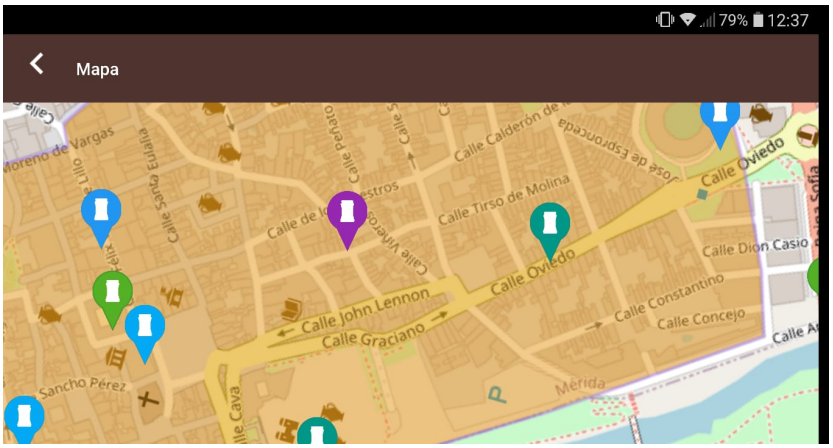
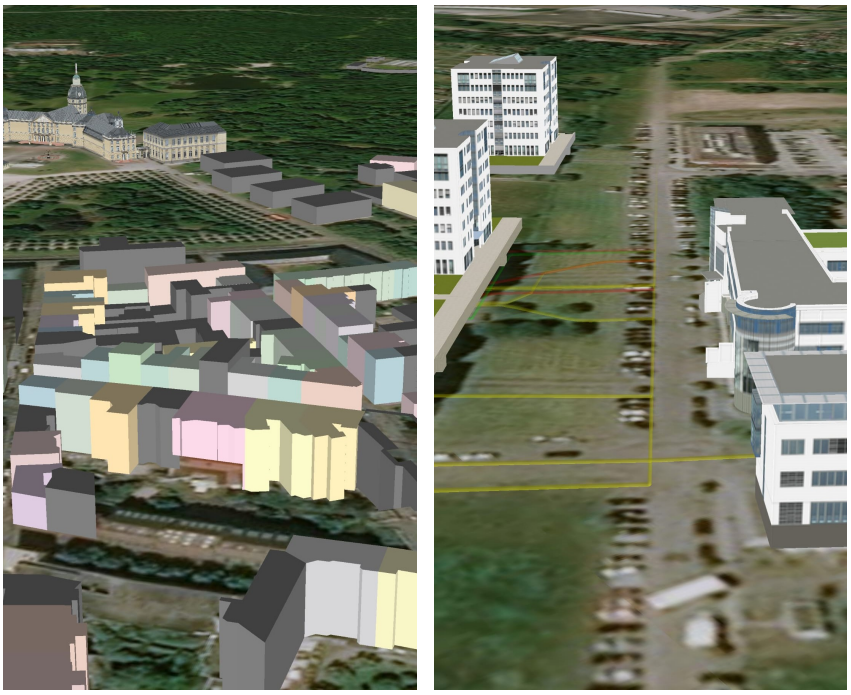


Figure 10.3: Point markers representing archaeological sites from the ancient Emerita Augusta where epigraphies have been found, over a map of the current city of Mérida (Extremadura, Spain) generated with Glob3Mobile.

Epigraphia3D is a project from the Spanish Government, the Spanish National Archeological Museum (*Museo Arqueológico Nacional*) and the Spanish National Museum for Roman Art (*Museo Nacional de Arte Romano*). It looks for making the ancient inscriptions of the Roman civilization discovered in Spain, called *epigraphies*, better known for the general public. In this project, the *Universidad de Las Palmas de Gran Canaria* collaborates in the 3D model generation of each epigraphy and in the design and development of an Android mobile application which work as a digital museum. The application aims to display the models, explain their meaning, history and details and show the places in which they are discovered.

The mobile application was developed using the Glob3Mobile framework [245]. It becomes the third practical use example for the system for efficient and cluttering-free transmission and visualization of point markers exposed in Chapter 2. For this case, the data set displayed with the system is the location of all the different places in which epigraphies have been found, which is shown in Figure 10.3.

10.2.4 Eifer MultiVis



(a) Heat demand by building.

(b) Underground gas, water and energy networks.

Figure 10.4: Spatial data regarding the city of Karlsruhe presented in the MultiVis mobile application.

MultiVis [62, 75] is a multi-modal mobile application for Android and iOS developed in the context of the *Smart City Lab* project of the *European Institute of Energy Research* (Eifer). It aims to showcase the results of different models and simulations regarding energy and smart cities. The application allows a seamless and smooth transition between a classic map environment, virtual reality and augmented reality. This multi-modal design allows technicians to take the key decisions about the planning of a city on the field, with a mobile phone and without any hardware requirements.

The application uses as an example case the German city of Karlsruhe. Over a map of the city, different spatial data sets are loaded containing 3D building models in CityGML format, solar radiation data, heat and energy consumption (see Figure 10.4 - a) or solar energy generation potential, among others. The implementation of the three scene models relies on Glob3Mobile and the native camera and positioning frameworks of Android and iOS.

Most of the MultiVis features have been implemented prior to the start of this research work [34]. However, the module for the representation of underground network data exposed in Chapter 3 was transferred to this project as a new feature for the application. The application gathers the locations, diameters and types of each pipe in CityGML from a 3DCityDB database, the same that contains the building models. The module generates the appropriate mesh for each pipe and displays it in the preferred mode from those exposed in the chapter: with dynamic distance-based alpha blending (Figure 10.4 - b), along with a surrounding ditch model or inside an excavation.

10.2.5 Aerolaser LiDAR classifier

Aerolaser System is a company based in Las Palmas de Gran Canaria specialized in acquisition and processing of geo-referenced data. One of its lines of work is the inspection of low, mid and high voltage power lines for maintenance tasks. At the present time, up to 60000 Km. of lines are monitored by the company with the help of LiDAR sensors, RGB cameras and thermal infrared cameras.

This results on very large volumes of point cloud data which should be processed in order to find the lines and all the near objects that could affect the power service. In this regard, a collaboration with the *Universidad de Las Palmas de Gran Canaria* has been opened and it is still active nowadays. The main goal of the collaboration is the automation of all the post-processing task of the LiDAR data acquired during an inspection flight. As a part of this collaboration, several different projects have been opened and knowledge generated in this research work has been successfully transferred to all of them.

The first of those projects involves the design and development of an airborne LiDAR point cloud semantic classifier. The classifier should be customizable for different needs of the company and should identify, at least, points belonging to the categories *ground*, *vegetation*, *building*, *project pylon*, *insulator strain*, *project wire*, *shield wire*, *bridge chain*, *cross line pylon*, and *cross line wire*. In order to difference between project and cross line corridors, a corridor poly line is available. The classifier modifies the input point cloud, attaching the result of the point segmentation class to each point record.

The executable developed for the project includes a full implementation of the Patch Decision Tree exposed in Chapter 5 of this dissertation for ground detection. It also includes a full implementation of the method for detection and refinement of power line corridor classes in airborne LiDAR point clouds exposed in Chapter 7 to perform classification of pylon, wire, insulator strain and bridge chain classes.

In Chapter 6, a method for detection of building and vegetation classes,

```
<config>
    <n_ficheros> 1 </n_ficheros>
    <polilinea_entrada_dxf in="1" fileName="C:\Users\Lab2\Documents\MATLAB\aerolaserproject\MatlabCode\nubes\dxf\d3c5.dxf">
        Nombre del fichero de entrada DXF con la polilinea del corredor para ayudar al metodo a detectar torres.
    </polilinea_entrada_dxf>
    <etiqueta_las_clasificado ext="_clasificado">
        Ejemplo: Entrada -> 000041.las; Salida -> 000041_clasificado.las
    </etiqueta_las_clasificado>
    <directorios>
        <las_entrada dir="C:\Users\Lab2\Documents\MATLAB\aerolaserproject\MatlabCode\nubes\entrada\">
            Directorio de ficheros de entrada las
        </las_entrada>
        <mat debug="0" dir="C:\Users\Lab2\Documents\MATLAB\aerolaserproject\MatlabCode\nubes\mats\">
            Provisional
        </mat>
        <las_salida dir="C:\Users\Lab2\Documents\MATLAB\aerolaserproject\MatlabCode\nubes\salida\">
            Directorio de ficheros de salida las
        </las_salida>
    </directorios>
    <nombre_fichero_log_errores fileName="error.log">
        Nombre del fichero del log de errores
    </nombre_fichero_log_errores>
    <valores_clasificacion>
        <unclassified value="0"> No clasificado </unclassified>
        <ground value="2"> Suelo </ground>
        <tree value="3"> Vegetacion </tree>
        <building value="6"> Edificio </building>
        <pylon value="16"> Torreta </pylon>
        <secondary_pylon value="19"> Torreta secundaria </secondary_pylon>
        <normal_wire value="17"> Cable </normal_wire>
        <secondary_wire value="20"> Cable secundario </secondary_wire>
        <insulator value="30"> Aislador </insulator>
        <insulator_over_bridge value="27"> Aislador en puente </insulator_over_bridge>
        <guide_wire value="31"> Cable guia </guide_wire>
        <bridge_wire value="29"> Cable puente </bridge_wire>
    </valores_clasificacion>
</config>
```

Figure 10.5: Example of the LiDAR classifier configuration file.

among others, in vehicle-borne LiDAR clouds has been introduced. Those clouds have conditions different from the airborne clouds [170], However, the feature maps explored in that chapter regarding relative height, point accumulation, omnivariance, eigensum and planarity are still valid for prediction and have been re-utilized in this project. Additional linearity [158], average return and presence of non-ground patch maps have been included as new feature maps. Using a ground truth generated by experts of the company, a random forest based grid classifier has been trained to distinguish the remaining points between the two affected classes. This allows to fulfill the minimum requirements of the application.

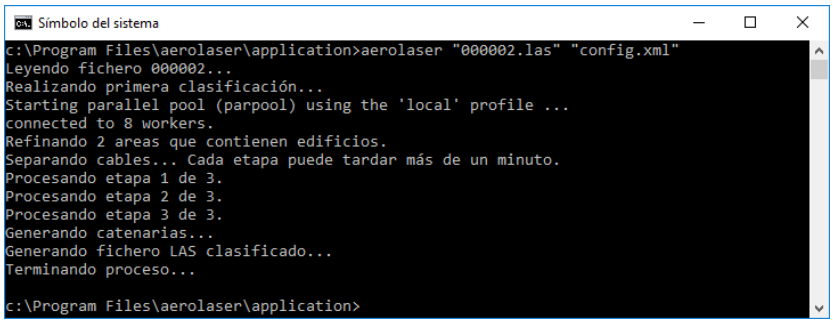


Figure 10.6: Execution progress of the LiDAR classificator over an input point cloud.

The classifier allows the user to set the class values and the batch execution of multiple point clouds. A configuration file example can be seen in Figure

10.5, an example of execution flow can be seen in Figure 10.6 and an example of the final result can be seen in Figure 10.7. Future steps of this ongoing project include research on roads, poles and car classes, between others.

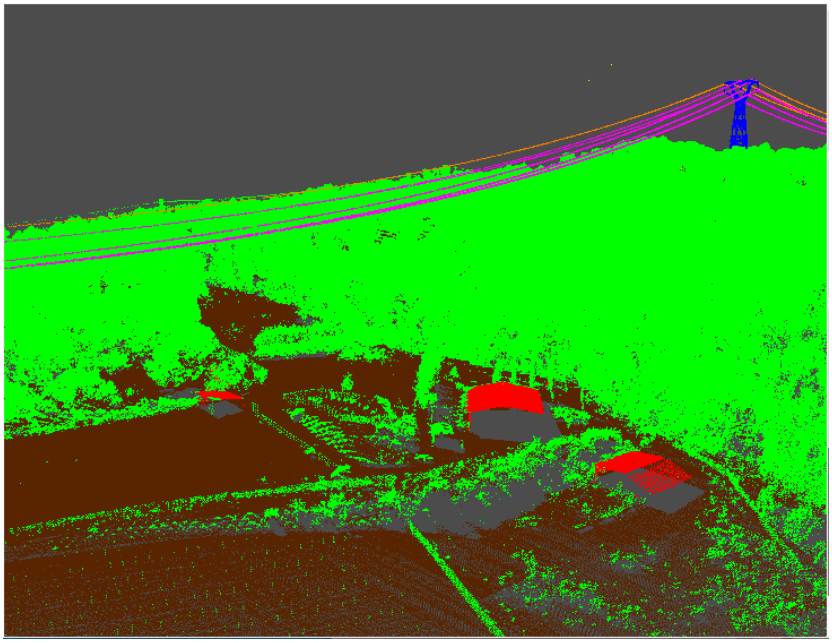


Figure 10.7: LiDAR Classificator output. Ground (brown), vegetation (green), building (red), primary wires (magenta), shield wires (orange) and pylon (blue) can be appreciated.

10.2.6 Aerolaser LiDAR digitizer and incidence detector

The second of the projects opened as a result of the collaboration between the *Universidad de Las Palmas de Gran Canaria* and *Aerolaser System* involves the design and development of a digitizer software. The digitizer should be able to extract vector catenary models for all the primary and shield wires and for all the bridge chains, given a power line corridor. Additionally, it should generate vector models of each pylon of the studied corridor. The generation should be start from previously classified point clouds.

The software created for such a task benefits from the modelling procedures for such elements that are introduced in Chapter 7. The program is capable of sequentially reading corridor data composed of multiple point clouds, extracting the models and saving them into a unique vector data set. This data set is finally save in the privative .dxf format to ease the integration of this tool with the company workflow. An example can be seen in Figure 10.8.

Combining the catenary models and the classified point clouds, a prototype

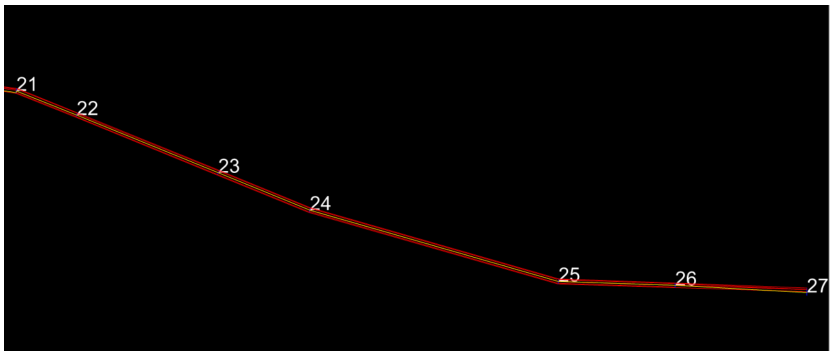


Figure 10.8: Example of digitization of a power line corredor in .dxf format.

is also implemented for the detection of possible incidences near the power line. To perform this task, the incidence tests introduced in Chapter 7 have been followed.

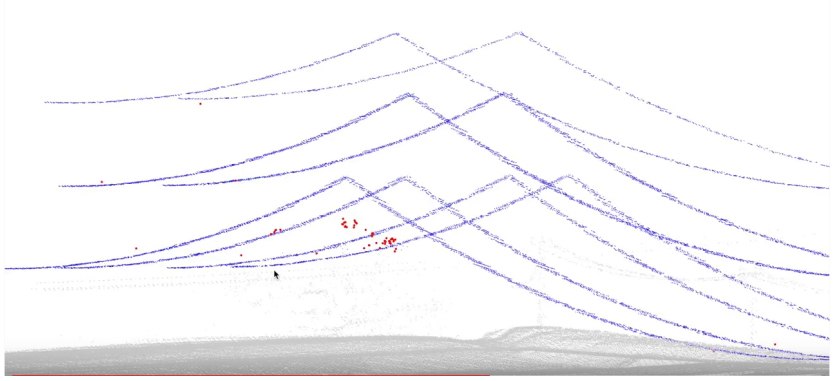


Figure 10.9: Tree incident points (red) which can be reach by some catenaries (blue) in a power line corridor. Grey colour indicates non-affected points.

An example of incidence points detection given by this prototype can be shown in Figure 10.9. More examples are available in a demonstrative video [1]. This prototype is still being developed in the moment of writing of this section, being the inclusion of changes in the catenary model according to estimated wind direction and temperature the future roadmap for this project.

10.2.7 Aerolaser LiDAR noise filtering tool

Finally, a third transference of knowledge from this research work was done in another project opened in the collaboration between the *Universidad de Las*

[1] `https://www.youtube.com/watch?v=3itb-XFyWGE`

Palmas de Gran Canaria and *Aerolaser System*. The project consist in a noise filtering software for airborne LiDAR point clouds, which is a task which is still dependent on human operators. As well as in the classifier case, the generated software should accept batch modes and the possibility of changing the noise classification values.

```
{
    "folders" : {
        "input_las" : "C:\Users\Lab2\Documents\Matlab\aerolaserproject\ruido\input/",
        "debug_mat" : "C:\Users\Lab2\Documents\Matlab\aerolaserproject\ruido\mats/",
        "output_las" : "C:\Users\Lab2\Documents\Matlab\\aerolaserproject\ruido\output/"
    },

    "process_modes" : {
        "subterranean" : true,
        "scattered_arc" : true,
        "tubular" : false,
        "striped" : false,
        "floating_blob" : false,
        "echo" : false
    },

    "general_settings" : {
        "unknown_class" : 0,
        "safe_noise_class" : 1,
        "unsafe_noise_class" : 6,

        "debug_class_low_intensity" : 3,
        "debug_class_high_intensity" : 16
    },

    "floating_blob_settings" : {
        "helicopter_folder" : "heli/",
        "helicopter_textfile" : "flight.txt",
        "decimal_step" : 0.002,
        "use_wgs84_coordinates": false
    }

}
```

Figure 10.10: Example of a noise filtering configuration file.

For this project, a full implementation of all the noise detectors discussed in Chapter 8 has been done. The operator should provide the input point clouds and, when necessary, the acquisition flight data. The detectors to be explored in each run of the final executable are also configurable, as it can be seen on Figure 10.10. As a final output, the noise classification is saved in each affected point data record of the original point cloud and optionally saved in a new file.

10.3 Future lines of work

The topics covered in this document are just a drop in the ocean of topics regarding spatial data prone to further research. Moreover, the contributions here presented also opened new lines of future work in the field of spatial data processing. In the following lines, some possible extensions to this work are highlighted.

- Regarding the quadtree-based level of detail structure for point sets and the architecture for vector data transmission, a natural continuation of the work is to adapt the methodology for the inclusion of line and polygon sets. The challenge of such a continuation comes on how to proceed on

node splitting when the division directly affects a line or a polygonal edge. Possible options are to take advantage of the sorting criteria to relate larger lines and polygons with coarser levels of detail, or to generate sub-lines in case the node division affects a line. Additionally, different strategies for node generation can be considered, e.g. splitting nodes based on the centroid of their related points instead of using the regular geographic criteria.

- Improvements on the topic of underground data visualization are also an interesting continuation of this work. On this matter, an open question is how changing the looks of the ditch and the excavation can affect the comprehension of the operator on the displayed data. Another possibility is to design the underground data display throughout cartographic principles, which have been proven to be effective for other types of mapping-oriented visualizations. Those new proposals should be validated by performing new user experience surveys before their adoptions. Moreover, the creation of new user interactions specific for underground data could also be welcome. A possibility on this line may be to implement a gesture for expanding and shrinking the size of the excavation. Finally, the addition of detail on the underground models generated for these visualizations can make them more suitable for VR/AR scenarios, where the user should be offered a complete immersive experience.

- In the topic of the 3D city model generation from LiDAR data, several lines of continuation can also be followed. One of them is to find rules which improve the prediction of pyramidal rooftops and the addition of new categories of rooftops within the system. For this objective, the addition of other data associated with each point, such as intensity, color or return values to the rule system may benefit the identification. The use of supervised techniques might ease this task as well. Another possibility consists in combining this work with the ideas proposed for building segmentation in urban clouds presented in Chapter 6 and the use cases section. After classifying the cloud, the corner-based algorithm for footprint generation can be applied directly to every building area found in the cloud, thus making the OSM input unnecessary. Other lines are related to the validation of the rooftop classification procedure: it can be extended to other cities, different LiDAR benchmarks and different point densities. A comparison against other methodologies from the state of the art is also desirable. A last future continuation of this work consists in analyzing the resulting model with the input data to ensure the topological correctness of the output, and applying corrections on the model generation if it becomes necessary. Having accurate building models is key in order to exploit the city model in smart city and energy oriented applications. Those applications can include simulations of heating and cooling needs, energy consumption, solar radiation and potential for photo-voltaic energy generation. Accurate models can also be used for visualization purposes in virtual globe applications, augmented reality and virtual reality.

- About ground classification and DEM generation, there are also open lines of potential work. From the comparison with methodologies from the state of the art, it was observed that color information benefits the ground classification and the approaches which exploit color outperform those which do not use it. A straightforward manner to continue is by studying how to integrate color in the patch decision tree. Additionally, slope cues can be considered in order to correct some misclassifications in mountainous terrain. Finally, a GPU version of the proposal on triangular regular network generation may be implemented, as it is mainly based on large matrix and vector operations.

- Regarding point segmentation in general, multiple continuations can be explored as well. One of them involves differentiation of subcategories on each of the proposed categories. As an example, ground classification can be divided into bare terrain, grass areas, roads and pavement, etc. Having these subcategories of ground, it is possible to introduce cues for better segmentation of cars and poles, as they normally are on or close to roads and paved surfaces. Another example is the vegetation, in which by combining the LiDAR data with RGB or infrared color features it might be possible to differentiate between multiple tree and crop species. This algorithm may be of interest on land use and land cover studies. It also may help the time estimation for future power line maintenance tasks, as each tree species has a known growth rate.

- Another possibility is to explore new features and architectures for the prediction of the already considered categories. It was seen in Chapter 6 that each of the different approaches applied to the Street3D benchmark, including our P4UCC proposal, explored very different descriptors and several learning-based and unsupervised architectures. Despite that, all the methodologies introduced improvements on the state-of-the-art. An open question is whether combining descriptors from different approaches has positive effects on the prediction. Another possibility is to introduce the progressive, class-by-class approach of the P4UCC methodology in a supervised architecture and, again, checks whether it benefits the prediction. This line of work can be extended not only to the pole, car, vegetation and building categories but also for power line and noise classification.

- An additional and interesting future combination of the work made in Chapters 7 and 9 about power corridor modelling and point cloud visualization is to display the vector models in the point cloud viewer, introduce different hypotheses of catenary movement and generate a relation of affected areas in the point cloud due to such a movement. In this regard, an initial Matlab prototype for the calculation of affected points has been introduced in the use cases section. By including this implementation into the viewer, the user may introduce in an easier manner the climate conditions on the wire, which are the key variables in the wire deformation and movement. This opens the possibility of instantaneously calculating

the new model position, showing it on the screen, generating alerts on the viewer for each affected point and generating an incidence report for the maintenance technicians.

- Finally, a new knowledge transference is also expected to be done in the near future, combining the work made in Chapters 2 and 9. The project, which will be included in the collaboration agreement between Aerolaser System and the ULPGC, aims to extend the capabilities of the *Megavisor* LiDAR viewer so country-sized point cloud repositories can be downloaded, displayed and edited efficiently. The quadtree-based point structures introduced in Chapter 2 will be used to progressively load the points of the repository in the scenario in a flawless manner. These structures should also be extended and adapted to load extra information about the scene, as color and thermal imagery or vector models of the power elements. A user interface will be also included to ease the common operator task, including tools for distance, area and volume measurement, configurable application of the noise removal, point segmentation and incidence report generation algorithms exposed in this work.

Annex I: Standard quality measurements

In this annex, the standard and common concepts that have been applied for the validation of all the methodologies that involve classification and point clouds are presented for a better understanding.

Confusion matrix, true and false positives-negatives

A confusion matrix, also called error matrix, is a table that relates the real data with the predictions performed by a given methodology on the data. Its use is extended in the fields of statistical classification and machine learning. In a confusion matrix, the sum of each row gives the total elements of a real class, and the sum of each column gives the total predictions made for a class.

If the order of classes in rows and columns is the same, the main diagonal of the matrix will offer the correct predictions of the methodology on each class, and the rest of elements of the matrix will offer the misclassification. Considering a sole class, four concepts are introduced on this regard for a better analysis:

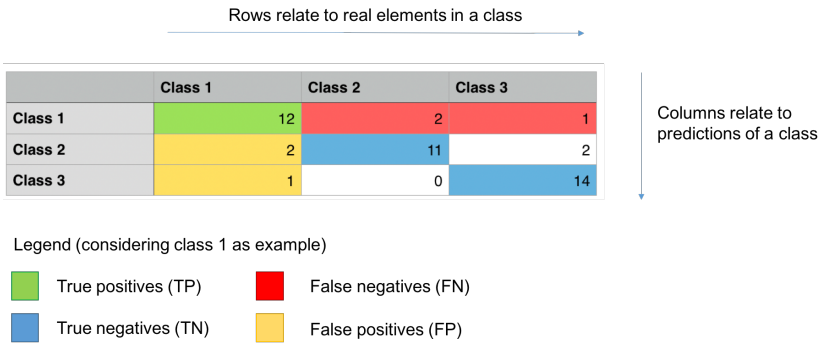


Figure 10.11: A confusion matrix example.

- *True positives* (TP), which are the correct predictions for the class;
- *True negatives* (TN), which are the correct rejections for the class;

- *False positives* (FP), also called *Type I errors*, which are erroneous predictions for the class;
- and *False negatives* (FN), also called *Type II errors*, which are missed predictions for the class.

In Figure 10.11, the concepts of confusion matrix, TP, TN, FP and FN can be seen graphically.

Recall

Also called *completeness* or *sensitivity*, it is the fraction of correctly classified elements in the total of elements of a given real class. It can be calculated following the expression:

$$recall = \frac{TP}{TP + FN}$$

Precision

Also called *correctness*, it is the fraction of correctly classified elements in the total of predictions of a class. It can be calculated following the expression:

$$precision = \frac{TP}{TP + FP}$$

F1-score

It is a harmonic mean of recall and precision and it offers a single score to evaluate the performance of the classifier, in which the two types of error have influence. It can be calculated following the expression:

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

IoU-score

The *Intersection over Union* (IoU) score was designed for experiments in the computer vision field and originally related the overlapping area with the combined area between two bounding boxes. Recently, it was adapted for machine learning as an alternative score for classification which takes into account the two types of error. It can be calculated following the expression:

$$IoU = \frac{TP}{TP + FP + FN}$$

Overall accuracy

The overall accuracy is a metric which considers all class predictions instead of a unique class and gives the general proportion of hits with respect to the total of elements in the dataset. It can be calculated following the expression:

$$OverallAccuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Mean of IoU

Finally, the mean of IoU is an alternative measure for the global performance of a multi-class prediction methodology. It consists in averaging all the single class IoU indicators.

Annex II: Scientific publications.

In this annex, the main scientific publications generated during the doctoral thesis are enumerated, as well as additional papers in which transferences of knowledge or minor contributions have been done:

Main publications

- T. Ku, R.C. Veltkamp, B. Boom, D. Duque-Arias, S. Velasco-Forero, J.E. Deschaud, F. Goulette, B. Marcotegui, S. Ortega, A. Trujillo, J.M. Santana, C. Ramírez, J.P. Suárez, K. Akadas, S. Gangisetty: **SHREC 2020 Track: 3D Point Cloud Semantic Segmentation for Street Scenes**. *Computers and Graphics*. (Accepted for publication: 29 July 2020, in press).

- Sebastián Ortega, José Miguel Santana, Jochen Wendel, Agustín Trujillo, Syed Monjur Murshed: **Generating 3D city models from open LiDAR point clouds: advancing towards smart city applications**. *In: Open Source Geospatial Science for Urban Studies: the value of open geospatial data. Springer.* (2020).

- Sebastián Ortega, Agustín Trujillo, José Miguel Santana, José Pablo Suárez, Jaisiel Santana: **Characterization and modeling of power line corridor elements from LiDAR point clouds**. *ISPRS Journal of Photogrammetry and Remote Sensing, 152, 24-33* (2019).

- Sebastián Ortega, Jochen Wendel, José Miguel Santana, Syed Monjur Murshed, Isaac Boates, Agustín Trujillo, Alexandru Nichersu, José Pablo Suárez: **Making the Invisible Visible—Strategies for Visualizing Underground Infrastructures in Immersive Environments**. *ISPRS International Journal of Geo-Information, 8(3), 152.* (2019).

- Jaisiel Santana, Sebastián Ortega, José Miguel Santana, Agustín Trujillo, José Pablo Suárez: **Noise reduction automation of liDAR point clouds for modeling and representation of high voltage lines in**

a 3D virtual globe. *In: Proceedings of the XXVIII Spanish Computer Graphics Conference (pp. 87-90).* (2018)

- Sebastián Ortega, Agustín Trujillo, José Miguel Santana, José Pablo Suárez: **An image-based method to classify power line scenes in LiDAR point clouds**. *In: Proceedings of the 12th International Symposium on Tools and Methods of Competitive Engineering, Las Palmas de Gran Canarias, Spain (pp. 7-11)* (2018).

- José Miguel Santana, Agustín Trujillo, José Pablo Suárez, Sebastián Ortega: **Accurate triangular regular network adjustment to large digital elevation models**. *In: Proceedings of the 25th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, in cooperation with EUROGRAPHICS Association, (pp. 107-115).* (2017)

- Sebastián Ortega, Agustín Trujillo, José Miguel Santana, José Pablo Suárez, Diego Gómez-Deck: **Rendering large datasets of georeferenced markers in mobile devices**. *In: Proceedings of the 32nd Spring Conference on Computer Graphics (pp. 67-74).* (2016)

Other contributions

- José Miguel Santana, Agustín Trujillo, Sebastián Ortega: **Visualization of Large Point Cloud in Unity**. *In Eurographics (Posters) (pp. 23-24)* (2019)

- José Miguel Santana, Jochen Wendel, Syed Monjur Murshed, Agustín Trujillo, Sebastián Ortega: **Towards augmented exploration of geospatial urban datasets**. *In: Adjunct Proceedings of the International Conference on Location Based Services.* (2019).

- Sebastián Ortega, Jochen Wendel, José Miguel Santana, Alexandru Nichersu, Alexander Simons, Agustín Trujillo, José Pablo Suárez, Andreas Koch: **Location Based Services for the Underground–Showcasing a multimodal visualization approach for underground infrastructure** . *In Adjunct Proceedings of the 14th International Conference on Location Based Services (pp. 227-228). ETH Zurich.* (2018)

- Sebastián Ortega, José Miguel Santana, Agustín Trujillo, José Pablo Suárez, Pablo Fernández: **Explora Gran Canaria - A mobile sense-based recommender system for hiking trails**. *In: 2nd Global Conference on Applied Computing in Science and Engineering (ACSE2)* (2017)

- José Miguel Santana, Agustín Trujillo, José Pablo Suárez, Sebastián Ortega: **Efficient atmospheric rendering for mobile virtual globes**. *In: Proceedings of the 25th International Conference in Central Europe on*

Computer Graphics, Visualization and Computer Vision, in cooperation with EUROGRAPHICS Association. (2017)

- Pablo Fernández, José Miguel Santana, Sebastián Ortega, Agustín Trujillo, José Pablo Suárez, Jaisiel Santana, Alejandro Sánchez, Conrado Domínguez: **Web-based GIS through a big data open source computer architecture for real time monitoring sensors of a seaport**. *In: The Rise of Big Spatial Data (pp. 41-53). Springer, Cham.* (2017)

- Manuel Ramírez, José Pablo Suárez, Agustín Trujillo, Pablo Fernández, José Miguel Santana, Sebastián Ortega: **Digital Epigraphic Heritage Made Simple: an Android App for Exploring 3D Roman Inscriptions**. *In Proceedings of the 14th Eurographics Workshop on Graphics and Cultural Heritage (pp. 179-182).* (2016)

- Pablo Fernández, José Miguel Santana, Sebastián Ortega, Agustín Trujillo, José Pablo Suárez, Conrado Domínguez, Jaisiel Santana, Alejandro Sánchez: **SmartPort: A platform for sensor data monitoring in a seaport based on FIWARE**. *Sensors, 16(3), 417.* (2016)

Annex III: Funding.

This research work would have not been possible without the help of the grant *Programa de personal investigador predoctoral en formación - 2015 - PIF-ULPGC-2015-ING-ARQ-2.*. I want to thank the *Vicerrectorado de Investigación, Innovación y Transferencia* of the *Universidad de Las Palmas de Gran Canaria* for this grant.

Bibliography

[1] Agustín Trujillo et al. "An efficient architecture for automatic shaders management on virtual globes". In: *Computing for Geospatial Research and Application (COM. Geo), 2014 Fifth International Conference on*. IEEE. 2014, pp. 38–42.

[2] Sebastián Ortega et al. "Rendering large datasets of georeferenced markers in mobile devices". In: *Proceedings of the 32nd Spring Conference on Computer Graphics*. 2016, pp. 67–74.

[3] Wei Peng, Matthew O Ward, and Elke A Rundensteiner. "Clutter reduction in multi-dimensional data visualization using dimension reordering". In: *IEEE Symposium on Information Visualization*. IEEE. 2004, pp. 89–96.

[4] Ayush Shrestha, Ying Zhu, and Yan Zhu. "Visual cluttering reduction for visualizing large spatio-temporal data sets". In: *2014 IEEE 17th International Conference on Computational Science and Engineering*. IEEE. 2014, pp. 137–143.

[5] Geoffrey Ellis and Alan Dix. "A taxonomy of clutter reduction for information visualisation". In: *IEEE transactions on visualization and computer graphics* 13.6 (2007), pp. 1216–1223.

[6] Paulo Miguel Pombinho, Maria Beatriz Carmo, and Ana Paula Afonso. "Evaluation of overcluttering prevention techniques for mobile devices". In: *2009 13th International Conference Information Visualisation*. IEEE. 2009, pp. 127–134.

[7] George W Furnas. "Generalized fisheye views". In: *ACM SIGCHI Bulletin* 17.4 (1986), pp. 16–23.

[8] Daniel A Keim and Hans-Peter Kriegel. "Visdb: A system for visualizing large databases". In: *ACM SIGMOD Record* 24.2 (1995), p. 482.

[9] Luke Mahe and Chris Broadfoot. *Too many markers!* 2010. URL: `https://web.archive.org/web/20160411214820/https://developers.google.com/maps/articles/toomanymarkers` (visited on 06/17/2020).

[10] Jean-Yves Delort. "Hierarchical cluster visualization in web mapping systems". In: *Proceedings of the 19th international conference on World Wide Web*. ACM. 2010, pp. 1241–1244.

[11] Che-An Lu, Chin-Hui Chen, and Pu-Jen Cheng. "Clustering and visualizing geographic data using geo-tree". In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*. IEEE Computer Society. 2011, pp. 479–482.

[12] Vojtech Krızek. "Intelligent Visualization of data from Multi-agent Simulations". In: (2012).

[13] Craig Allison, Richard Treves, and Edward Redhead. *The Usability of Online Data Maps: An Ongoing Web Based Questionnaire Investigation into Users' Understanding and Preference for Geo-Spatial Visualisations*. Last accessed: 2016-02-11. 2013. URL: `http://www.geos.ed.ac.uk/~gisteac/proceedingsonline/GISRUK2013/gisruk2013_submission_26.pdf`.

[14] Jari Korpi and Paula Ahonen-Rainio. "Clutter reduction methods for point symbols in map mashups". In: *The Cartographic Journal* 50.3 (2013), pp. 257–265.

[15] Michela Bertolotto and Max J Egenhofer. "Progressive transmission of vector map data over the world wide web". In: *GeoInformatica* 5.4 (2001), pp. 345–373.

[16] Szymon Rusinkiewicz and Marc Levoy. "Streaming QSplat: A viewer for networked visualization of large, dense models". In: *Proceedings of the 2001 symposium on Interactive 3D graphics*. 2001, pp. 63–68.

[17] Wei Cheng. "Streaming of 3D progressive meshes". In: *Proceedings of the 16th ACM international conference on Multimedia*. 2008, pp. 1047–1050.

[18] Junho Kim, Seungyong Lee, and Leif Kobbelt. "View-dependent streaming of progressive meshes". In: *Proceedings Shape Modeling Applications, 2004.* IEEE. 2004, pp. 209–220.

[19] Shai Dekel and Nitzan Goldberg. *System and method for the lossless progressive streaming of images over a communication network*. US Patent 7,024,046. 2006.

[20] Michael Waschbüsch et al. "Progressive Compression of Point-Sampled Models." In: *SPBG*. 2004, pp. 95–102.

[21] Xiaochen Kang, Jiping Liu, and Xiangguo Lin. "Streaming progressive TIN densification filter for airborne LiDAR point clouds using multi-core architectures". In: *Remote sensing* 6.8 (2014), pp. 7212–7232.

[22] David Luebke et al. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[23] Enrico Gobbetti and Fabio Marton. "Layered Point Clouds." In: *SPBG* 4 (2004), pp. 113–120.

[24] Ruwen Schnabel and Reinhard Klein. "Octree-based Point-Cloud Compression." In: *Spbg* 6 (2006), pp. 111–120.

[25] Yan Huang et al. "Octree-Based Progressive Geometry Coding of Point Clouds." In: *SPBG* 6 (2006), pp. 103–110.

[26] Julius Kammerl et al. "Real-time compression of point cloud streams". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 778–785.

[27] Jason Smith, G Petrova, and Scott Schaefer. "Progressive encoding and compression of surfaces generated from point cloud data". In: *Computers & Graphics* 36.5 (2012), pp. 341–348.

[28] Kevin Sahr, Denis White, and A Jon Kimerling. "Geodesic discrete global grid systems". In: *Cartography and Geographic Information Science* 30.2 (2003), pp. 121–134.

[29] Michael F Goodchild. "Discrete global grids for digital earth". In: *International Conference on Discrete Global Grids, California: Santa Barbara*. Citeseer. 2000.

[30] Ali Mahdavi-Amiri, Troy Alderson, and Faramarz Samavati. "A survey of digital earth". In: *Computers & Graphics* 53 (2015), pp. 95–117.

[31] Open Source Geospatial Foundation. *What is Geoserver?* 2016. URL: `http://geoserver.org/about/` (visited on 06/16/2020).

[32] Agustín Trujillo et al. "Glob3 mobile: An open source framework for designing virtual globes on ios and android mobile devices". In: *Progress and New Trends in 3D Geoinformation Sciences*. Springer, 2013, pp. 211–229.

[33] José P Suárez et al. "An open source virtual globe framework for iOS, Android and WebGL compliant browser". In: *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*. 2012, pp. 1–10.

[34] José Miguel Santana Núñez. *New methodologies and techniques on the development of virtual globes for mobile devices and the web*. Universidad de las Palmas de Gran Canaria, 2017.

[35] Shreiner Dave et al. *Opengl Programming Guide: The Official Guide To Learning Opengl, Version 2.1, 6/E*. Pearson Education India, 2008.

[36] Marc Wick and Unxos GmbH. *Geonames download area website*. 2016. URL: `http://download.geonames.org/export/dump/` (visited on 06/16/2020).

[37] Gerhard Schall, Stefanie Zollmann, and Gerhard Reitmayr. "Smart Vidente: advances in mobile augmented reality for interactive visualization of underground infrastructure". In: *Personal and ubiquitous computing* 17.7 (2013), pp. 1533–1549.

[38] Thomas Becker, Claus Nagel, and Thomas H Kolbe. "Semantic 3D modeling of multi-utility networks in cities for analysis and 3D visualization". In: *Progress and New Trends in 3D Geoinformation Sciences*. Springer, 2013, pp. 41–62.

[39] Jukka Sirkiä et al. “Data utilization at finnish water and wastewater utilities: Current practices vs. state of the art”. In: *Utilities Policy* 45 (2017), pp. 69–75.

[40] Paolo Neirotti et al. “Current trends in Smart City initiatives: Some stylised facts”. In: *Cities* 38 (2014), pp. 25–36.

[41] Siyka Zlatanova, Alias Abdul Rahman, and Morakot Pilouk. “Trends in 3D GIS development”. In: *Journal of Geospatial Engineering* 4.2 (2002), pp. 71–80.

[42] Y Du and Sisi Zlatanova. “An approach for 3D visualization of pipelines”. In: *Innovations in 3D geo information systems*. Springer, 2006, pp. 501–517.

[43] Dhiraj Amin and Sharvari Govilkar. “Comparative study of augmented reality SDKs”. In: *International Journal on Computational Science & Applications* 5.1 (2015), pp. 11–26.

[44] M Christen, U Clement, and A Meyer. *Mixed Reality Anwendungen mit 3D-Stadtmodellen.* 2018.

[45] Riccardo Palmarini et al. “A systematic review of augmented reality applications in maintenance”. In: *Robotics and Computer-Integrated Manufacturing* 49 (2018), pp. 215–228.

[46] Kangsoo Kim et al. “Revisiting Trends in Augmented Reality Research: A Review of the 2nd Decade of ISMAR (2008–2017)”. In: *IEEE transactions on visualization and computer graphics* 24.11 (2018), pp. 2947–2962.

[47] Manoela Rogofski Brum and Rafael Rieder. “Virtual reality applications for smart cities in health: A systematic review”. In: *Virtual and Augmented Reality (SVR), 2015 XVII Symposium on*. IEEE. 2015, pp. 154–159.

[48] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. “Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR”. In: *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society. 2008, pp. 193–202.

[49] Sebastián Ortega et al. “Making the Invisible Visible—Strategies for Visualizing Underground Infrastructures in Immersive Environments”. In: *ISPRS International Journal of Geo-Information* 8.3 (2019), p. 152.

[50] Gethin W Roberts et al. “The use of augmented reality, GPS and INS for subsurface data visualization”. In: *FIG XXII International Congress*. 2002, pp. 1–12.

[51] Ryan Bane and Tobias Hollerer. “Interactive tools for virtual x-ray vision in mobile augmented reality”. In: *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society. 2004, pp. 231–239.

[52] Benjamin Avery, Christian Sandor, and Bruce H Thomas. "Improving spatial perception for augmented reality x-ray vision". In: *2009 IEEE Virtual Reality Conference*. IEEE. 2009, pp. 79–82.

[53] Xing Su et al. "Uncertainty-aware visualization and proximity monitoring in urban excavation: a geospatial augmented reality approach". In: *Visualization in engineering* 1.1 (2013), p. 2.

[54] Shuai Li, Hubo Cai, and Vineet R Kamat. "Uncertainty-aware geospatial system for mapping and visualizing underground utilities". In: *Automation in Construction* 53 (2015), pp. 105–119.

[55] Xiaolei Zhang et al. "ARGIS-based outdoor underground pipeline information system". In: *Journal of Visual Communication and Image Representation* 40 (2016), pp. 779–790.

[56] Léon olde Scholtenhuis et al. "Spying the underground: visualizing subsurface utilities' location uncertainties with fuzzy 3D". In: *SPOOL* 4.2 (2017), pp. 61–64.

[57] Gerhard Schall et al. "Handheld augmented reality for underground infrastructure visualization". In: *Personal and ubiquitous computing* 13.4 (2009), pp. 281–291.

[58] Jiazhou Chen et al. "On-line visualization of underground structures using context features". In: *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*. ACM. 2010, pp. 167–170.

[59] Stefanie Zollmann et al. "Image-based X-ray visualization techniques for spatial understanding in Outdoor Augmented Reality". In: *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design*. ACM. 2014, pp. 194–203.

[60] Sangho Lee, Jangwon Suh, and Hyeong-Dong Park. "BoreholeAR: A mobile tablet application for effective borehole database visualization using an augmented reality technology". In: *Computers & Geosciences* 76 (2015), pp. 41–49.

[61] Domenica Mirauda et al. "Applications of Mobile Augmented Reality to Water Resources Management". In: *Water* 9.9 (2017), p. 699.

[62] José Miguel Santana et al. "Multimodal location based services—semantic 3D city data as virtual and augmented reality". In: *Progress in location-based services 2016*. Springer, 2017, pp. 329–353.

[63] Chiara Delmastro, Evasio Lavagno, and Laura Schranz. "Underground urbanism: master plans and sectorial plans". In: *Tunnelling and Underground Space Technology* 55 (2016), pp. 103–111.

[64] Xiaoming Li et al. "XEarth: A 3D GIS Platform for managing massive city information". In: *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[65] European Comission. *Utility and governmental services*. 2019. URL: http://inspire.ec.europa.eu/theme/us.

[66] buildingSmart. *IFC Overview*. 2019. URL: http://www.buildingsmart-tech.org/specifications/ifc-overview.

[67] ESRI. *What is a utility network?* 2019. URL: https://pro.arcgis.com/es/pro-app/help/data/utility-network/what-is-a-utility-network-.htm.

[68] Tatjana Kutzner and Thomas H Kolbe. "Extending semantic 3D city models by supply and disposal networks for analysing the urban supply situation". In: *Lösungen für eine Welt im Wandel, Dreiländertagung der SGPF, DGPF und OVG, 36. Wissenschaftlich-Technische Jahrestagung der DGPF*. 2016, pp. 382–394.

[69] Tatjana Kutzner. *CityGML-UtilityNetwork-ADE Github repository*. 2019. URL: https://github.com/TatjanaKutzner/CityGML-UtilityNetwork-ADE.

[70] Gerhard Gröger et al. "OGC city geography markup language (CityGML) encoding standard, version 2.0". In: *OGC Doc* 12-019 (2012).

[71] Xander den Duijn. "A 3D data modeling approach for integrated management of below and above ground utility network features". In: (2018).

[72] I Boates, G Agugiaro, and A Nichersu. "Network modelling and semantic 3D city models: testing the maturity of the Utility Network ADE for CityGML with a water network test case." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4.4 (2018).

[73] Isaac Boates. *Demonstrating Utility Network Interdependency Modelling Using the UtilityNetwork Application Domain Extension for CityGML*. Master Thesis at Hochschule Karlsruhe Technik und Wirtschaft. 2018.

[74] X Den Duijn, Giorgio Agugiaro, and Sisi Zlatanova. "Modelling below- and above-ground utility network features with the CityGML Utility Network ADE: Experiences from Rotterdam". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4.4/W7 (2018).

[75] Jochen Wendel, José Miguel Santana, and Alexander Simons. "Semantic 3D city data as virtual and augmented reality Urban Energy Modelling". In: *GIM International-The Worldwide Magazine for Geomatics* 31.12 (2017), pp. 30–33.

[76] Jose Pablo Suárez et al. "An efficient terrain Level of Detail implementation for mobile devices and performance study". In: *Computers, Environment and Urban Systems* 52 (2015), pp. 21–33.

[77] David S Ebert et al. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[78] P. F. Verhulst. *A note on population growth*. Correspondence Mathematiques et Physiques, 10:113-121. 1838.

[79] David L Elliott. *A better activation function for artificial neural networks*. Tech. rep. 1993.

[80] Patrick Cozzi and Kevin Ring. *3D Engine Design for Virtual Globes*. 1st. Natick, MA, USA: A. K. Peters, Ltd., 2011, p. 520. ISBN: 1568817118. URL: http://www.amazon.com/3D-Engine-Design-Virtual-Globes/dp/1568817118.

[81] Patrick Cozzi and Christophe Riccio. *OpenGL insights*. AK Peters/CRC Press, 2012.

[82] Autodesk. *Maya Render Pass concepts and techniques*. 2010. URL: http://images.autodesk.com/adsk/files/maya_render_pass_concepts_and_techniques_whitepaper_us.pdf.

[83] T Becker and G König. "Generalized cartographic and simultaneous representation of utility networks for decision-support systems and crisis management in urban environments." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* (2015).

[84] Thomas H Kolbe, Gerhard Gröger, and Lutz Plümer. "CityGML: Interoperable access to 3D city models". In: *Geo-information for disaster management*. Springer, 2005, pp. 883–899.

[85] Filip Biljecki et al. "The most common geometric and semantic errors in CityGML datasets." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4 (2016).

[86] Jochen Wendel et al. "Rapid development of semantic 3D city models for urban energy analysis based on free and open data sources and software". In: *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. 2017, pp. 1–7.

[87] M Lang et al. "Light detection and ranging: New information for improved wetland mapping and monitoring". In: *National Wetlands Newsletter* 32.5 (2010), pp. 10–13.

[88] The American Society of Photogrammetry and Remote Sensing. *Laser File Format Exchange Activities*. 2019. URL: https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities.

[89] Sebastián Ortega et al. "Generating 3D city models from open LiDAR point clouds: advancing towards smart city applications". In: *Open Source Geospatial Science for Urban Studies: the value of open geospatial data (in press)*. Springer, 2020.

[90] George Vosselman and Hans-Gerd Maas. *Airborne and terrestrial laser scanning*. CRC press, 2010.

[91] André Henn et al. "Model driven reconstruction of roofs from sparse LIDAR point clouds". In: *ISPRS Journal of photogrammetry and remote sensing* 76 (2013), pp. 17–29.

[92] Michael Ying Yang and Wolfgang Förstner. "Plane detection in point cloud data". In: *Proceedings of the 2nd int conf on machine control guidance, Bonn*. Vol. 1. 2010, pp. 95–104.

[93] Yuanfan Zheng and Qihao Weng. "Model-driven reconstruction of 3-D buildings using LiDAR data". In: *IEEE geoscience and remote sensing letters* 12.7 (2015), pp. 1541–1545.

[94] Wuming Zhang et al. "3D building roof modeling by optimizing primitive's parameters using constraints from LiDAR data and aerial imagery". In: *Remote Sensing* 6.9 (2014), pp. 8107–8133.

[95] Martin Kada and Laurence McKinley. "3D building reconstruction from LiDAR based on a cell decomposition approach". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 38.Part 3 (2009), W4.

[96] Jeremy Castagno and Ella Atkins. "Roof shape classification from LiDAR and satellite image data fusion using supervised learning". In: *Sensors* 18.11 (2018), p. 3960.

[97] A Mahphood and H Arefi. "A data driven method for flat roof building reconstruction from LiDAR point clouds." In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 42 (2017).

[98] Yanming Chen et al. "Multiscale grid method for detection and reconstruction of building roofs from airborne LiDAR data". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7.10 (2014), pp. 4081–4094.

[99] Yongjun Wang et al. "Three-dimensional reconstruction of building roofs from airborne LiDAR data based on a layer connection and smoothness strategy". In: *Remote Sensing* 8.5 (2016), p. 415.

[100] Yusheng Xu et al. "Segmentation of building roofs from airborne LiDAR point clouds using robust voxel-based region growing". In: *Remote Sensing Letters* 8.11 (2017), pp. 1062–1071.

[101] Aparajithan Sampath and Jie Shan. "Segmentation and reconstruction of polyhedral building roofs from aerial lidar point clouds". In: *IEEE Transactions on geoscience and remote sensing* 48.3 (2009), pp. 1554–1567.

[102] Jingwei Song, Jianwei Wu, and Yongyao Jiang. "Extraction and reconstruction of curved surface buildings by contour clustering using airborne LiDAR data". In: *Optik* 126.5 (2015), pp. 513–521.

[103] Rujun Cao et al. "3D building roof reconstruction from airborne LiDAR point clouds: A framework based on a spatial database". In: *International Journal of Geographical Information Science* 31.7 (2017), pp. 1359–1380.

[104] Jaewook Jung, Yoonseok Jwa, and Gunho Sohn. "Implicit regularization for reconstructing 3D building rooftop models using airborne LiDAR data". In: *Sensors* 17.3 (2017), p. 621.

[105] Liying Wang et al. "Voxel segmentation-based 3D building detection algorithm for airborne LIDAR data". In: *PloS one* 13.12 (2018), e0208996.

[106] Mohammad Awrangjeb and Clive Fraser. "Automatic segmentation of raw LiDAR data for extraction of building roofs". In: *Remote Sensing* 6.5 (2014), pp. 3716–3751.

[107] Parham Pahlavani, Hamed Amini Amirkolaee, and Behnaz Bigdeli. "3D reconstruction of buildings from LiDAR data considering various types of roof structures". In: *International journal of remote sensing* 38.5 (2017), pp. 1451–1482.

[108] Bisheng Yang et al. "Automated reconstruction of building lods from airborne lidar point clouds using an improved morphological scale space". In: *Remote Sensing* 9.1 (2017), p. 14.

[109] Jianhua Yan et al. "Automatic construction of 3-D building model from airborne LIDAR data through 2-D snake algorithm". In: *IEEE Transactions on Geoscience and Remote Sensing* 53.1 (2014), pp. 3–14.

[110] Liang Cheng et al. "Integration of LiDAR data and optical multi-view images for 3D reconstruction of building roofs". In: *Optics and Lasers in Engineering* 51.4 (2013), pp. 493–502.

[111] Viji Varghese, Dimple A Shajahan, and Aneesh G Nath. "Building boundary tracing and regularization from LiDAR point cloud". In: *2016 International Conference on Emerging Technological Trends (ICETT)*. IEEE. 2016, pp. 1–6.

[112] Jing-zhong Xu, You-chuan Wan, and Fang Yao. "A method of 3D Building Boundary Extraction from airborne LIDAR points cloud". In: *2010 Symposium on Photonics and Optoelectronics*. IEEE. 2010, pp. 1–4.

[113] M Awrangjeb. "Using point cloud data to identify, trace, and regularize the outlines of buildings". In: *International Journal of Remote Sensing* 37.3 (2016), pp. 551–579.

[114] Bin Wu et al. "A graph-based approach for 3D building model reconstruction from airborne LiDAR point clouds". In: *Remote Sensing* 9.1 (2017), p. 92.

[115] Hongchao Fan, Wei Yao, and Qing Fu. "Segmentation of sloped roofs from airborne LiDAR point clouds using ridge-based hierarchical decomposition". In: *Remote Sensing* 6.4 (2014), pp. 3284–3301.

[116] Ruisheng Wang, Jiju Peethambaran, and Dong Chen. "LiDAR Point Clouds to 3-D Urban Models : A Review". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11.2 (2018), pp. 606–627.

[117] David H Douglas and Thomas K Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122.

[118] Keqi Zhang, Jianhua Yan, and S-C Chen. "Automatic construction of building footprints from airborne LIDAR data". In: *IEEE Transactions on Geoscience and Remote Sensing* 44.9 (2006), pp. 2523–2533.

[119] Philip HS Torr and Andrew Zisserman. "MLESAC: A new robust estimator with application to estimating image geometry". In: *Computer vision and image understanding* 78.1 (2000), pp. 138–156.

[120] Government of La Rioja (Spain). *Datos de vuelo LiDAR PNOA*. 2015. URL: `https://www.iderioja.larioja.org/vct/index.php?c=46757a356c32636e39766143325861395a2b352b4c773d3d`.

[121] M. Raifer. *Overpass Turbo*. 2019. URL: `https://overpass-turbo.eu/`.

[122] Hugues Hoppe et al. *Surface reconstruction from unorganized points*. Vol. 26. 2. ACM, 1992.

[123] William HE Day and Herbert Edelsbrunner. "Efficient algorithms for agglomerative hierarchical clustering methods". In: *Journal of classification* 1.1 (1984), pp. 7–24.

[124] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. "On the shape of a set of points in the plane". In: *IEEE Transactions on information theory* 29.4 (1983), pp. 551–559.

[125] AS Antonarakis, Keith S Richards, and James Brasington. "Object-based land cover classification using airborne LiDAR". In: *Remote Sensing of Environment* 112.6 (2008), pp. 2988–2998.

[126] Sebastián Ortega et al. "Characterization and modeling of power line corridor elements from LiDAR point clouds". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 152 (2019), pp. 24–33.

[127] Ali H Omar et al. "The CALIPSO automated aerosol classification and lidar ratio selection algorithm". In: *Journal of Atmospheric and Oceanic Technology* 26.10 (2009), pp. 1994–2014.

[128] Yu Wang et al. "Building point detection from vehicle-borne LiDAR data based on voxel group and horizontal hollow analysis". In: *Remote Sensing* 8.5 (2016), p. 419.

[129] Ziyue Chen, Bingbo Gao, and Bernard Devereux. "State-of-the-art: DTM generation using airborne LIDAR data". In: *Sensors* 17.1 (2017), p. 150.

[130] José M Santana et al. "Accurate triangular regular network adjustment to large digital elevation models". In: (2017).

[131] Karl Kraus and Norbert Pfeifer. "Determination of terrain models in wooded areas with airborne laser scanner data". In: *ISPRS Journal of Photogrammetry and remote Sensing* 53.4 (1998), pp. 193–203.

[132] Peter Axelsson. "DEM generation from laser scanner data using adaptive TIN models". In: *International archives of photogrammetry and remote sensing* 33.4 (2000), pp. 110–117.

[133] Pejman Rashidi and Heidar Rastiveis. "Extraction of ground points from LiDAR data based on slope and progressive window thresholding (SPWT)". In: *Earth Observation and Geomatics Engineering* 2 (2018), pp. 36–44.

[134] ES Anderson, JA Thompson, and RE Austin. "LIDAR density and linear interpolator effects on elevation estimates". In: *International Journal of Remote Sensing* 26.18 (2005), pp. 3889–3900.

[135] Donald Shepard. "A two-dimensional interpolation function for irregularly-spaced data". In: *Proceedings of the 1968 23rd ACM national conference*. 1968, pp. 517–524.

[136] K Kraus, Mikhail Em, et al. "Linear least-squares interpolation". In: (1972).

[137] Jack PC Kleijnen. "Kriging metamodeling in simulation: A review". In: *European journal of operational research* 192.3 (2009), pp. 707–716.

[138] Qi Chen et al. "Filtering airborne laser scanning data with morphological methods". In: *Photogrammetric Engineering & Remote Sensing* 73.2 (2007), pp. 175–185.

[139] Masaomi Okagawa. "Algorithm of multiple filter to extract DSM from LiDAR data". In: *2001 ESRI International User Conference*. 2001, pp. 193–203.

[140] Keqi Zhang et al. "A progressive morphological filter for removing non-ground measurements from airborne LIDAR data". In: *IEEE transactions on geoscience and remote sensing* 41.4 (2003), pp. 872–882.

[141] Yong Li et al. "An improved top-hat filter with sloped brim for extracting ground points from airborne lidar point clouds". In: *Remote sensing* 6.12 (2014), pp. 12885–12908.

[142] Shuwei Li, Huabo Sun, and Lei Yan. "A filtering method for generating DTM based on multi-scale mathematic morphology". In: *2011 IEEE International Conference on Mechatronics and Automation*. IEEE. 2011, pp. 693–697.

[143] Keqi Zhang and Dean Whitman. "Comparison of three algorithms for filtering airborne lidar data". In: *Photogrammetric Engineering & Remote Sensing* 71.3 (2005), pp. 313–324.

[144] Pejman Rashidi and Heidar Rastiveis. "Ground filtering lidar data based on multi-scale analysis of height difference threshold". In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* (2017), pp. 225–229.

[145] George Sithole and George Vosselman. "Filtering of laser altimetry data using a slope adaptive filter". In: *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* 34.3/W4 (2001), pp. 203–210.

[146] Junichi Susaki. "Adaptive slope filtering of airborne LiDAR data in urban areas for digital terrain model (DTM) generation". In: *Remote Sensing* 4.6 (2012), pp. 1804–1819.

[147] M Uysal and N Polat. "Investigating performance of airborne lidar data filtering with triangular irregular network (tin) algorithm". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 40.7 (2014), p. 199.

[148] Yining Quan et al. "Filtering LiDAR data based on adjacent triangle of triangulated irregular network". In: *Multimedia Tools and Applications* 76.8 (2017), pp. 11051–11063.

[149] Yanfeng Gu, Qingwang Wang, and Bingqian Xie. "Multiple kernel sparse representation for airborne LiDAR data classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 55.2 (2017), pp. 1085–1105.

[150] Joachim Niemeyer et al. "Hierarchical higher order crf for the classification of airborne lidar point clouds in urban areas". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives 41 (2016)* 41 (2016), pp. 655–662.

[151] Lukas Winiwarter and Gottfried Mandlburger. "Classification of 3D Point Clouds using Deep Neural Networks". In: (2019).

[152] Ruibin Zhao, Mingyong Pang, and Jidong Wang. "Classifying airborne LiDAR point clouds via deep features learned by a multi-scale convolutional neural network". In: *International Journal of Geographical Information Science* 32.5 (2018), pp. 960–979.

[153] Zhishuang Yang et al. "A convolutional neural network-based 3D semantic labeling method for ALS point clouds". In: *Remote Sensing* 9.9 (2017), p. 936.

[154] Zhen Wang et al. "A deep neural network with spatial pooling (dnnsp) for 3-d point cloud classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 56.8 (2018), pp. 4594–4604.

[155] Jinming Zhang et al. "DEM Extraction from ALS Point Clouds in Forest Areas via Graph Convolution Network". In: *Remote Sensing* 12.1 (2020), p. 178.

[156] Frédéric Bretar and Nesrine Chehata. "Terrain modeling from lidar range data in natural landscapes: A predictive and bayesian framework". In: *IEEE Transactions on Geoscience and Remote Sensing* 48.3 (2009), pp. 1568–1578.

[157] Sebastián Ortega et al. "An image-based method to classify power line scenes in LiDAR point clouds". In: *12th International Symposium on Tools and Methods of Competitive Engineering* (2018), pp. 585–593.

[158] R Blomley et al. "Shape distribution features for point cloud analysis-a geometric histogram approach on multiple scales". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.3 (2014), p. 9.

[159] Internacional Society for Photogrammetry and Remote Sensing. *3D Building Reconstruction and Semantic Labeling*. 2018. URL: `http://www2.isprs.org/commissions/comm3/wg4/tests.html`.

[160] Joachim Niemeyer, Franz Rottensteiner, and Uwe Soergel. "Contextual classification of LiDAR data and building object detection in urban areas". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 87 (2014), pp. 152–165.

[161] Yanfeng Gu, Qingwang Wang, and Bingqian Xie. *MKSRC for Airborne LiDAR Data Classification*. `https://www.researchgate.net/publication/307508843_Multiple_Kernel_Sparse_RepresentationMKSRC_for_Airborne_LiDAR_Data_Classification`. 2017.

[162] Jihad El-Sana and Amitabh Varshney. "Generalized view-dependent simplification". In: *Computer Graphics Forum*. Vol. 18. 3. Wiley Online Library. 1999, pp. 83–94.

[163] Enrico Puppo. "Variable Resolution Terrain Surfaces." In: *CCCG*. 1996, pp. 202–210.

[164] Hugues Hoppe. "Smooth view-dependent level-of-detail control and its application to terrain rendering". In: *Proceedings Visualization'98 (Cat. No. 98CB36276)*. IEEE. 1998, pp. 35–42.

[165] JC Aguero, A Feuer, and GC Goodwin. "Terrain modelling via triangular regular networks". In: *MODSIM 2003* 33 (2003).

[166] Peter Lindstrom et al. "Real-time, continuous level of detail rendering of height fields". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 109–118.

[167] Renato Pajarola. "Large scale terrain visualization using the restricted quadtree triangulation". In: *Proceedings Visualization'98 (Cat. No. 98CB36276)*. IEEE. 1998, pp. 19–26.

[168] Ron Goldman. *Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling*. Elsevier, 2002.

[169] Christopher C Paige and Michael A Saunders. "LSQR: An algorithm for sparse linear equations and sparse least squares". In: *ACM Transactions on Mathematical Software (TOMS)* 8.1 (1982), pp. 43–71.

[170] Jianhua Chang et al. "Noise reduction in Lidar signal using correlation-based EMD combined with soft thresholding and roughness penalty". In: *Optics Communications* 407 (2018), pp. 290–295.

[171] Yanjun Wang et al. "A Survey of Mobile Laser Scanning Applications and Key Techniques over Urban Areas". In: *Remote Sensing* 11.13 (2019), p. 1540.

[172] Yuan Gao and MC Li. "Airborne LIDAR Point Cloud Classification Based on Multilevel Point Cluster Features". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2020), pp. 1231–1237.

[173] Nesrine Chehata, Li Guo, and Clément Mallet. "Airborne lidar feature selection for urban classification using random forests". In: 2009.

[174] K Fukano and H Masuda. "Detection and classification of pole-like objects from mobile mapping data." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 2 (2015).

[175] Martin Weinmann et al. "Contextual classification of point cloud data by exploiting individual 3D neigbourhoods". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences II-3 (2015), Nr. W4* 2.W4 (2015), pp. 271–278.

[176] Chisheng Wang et al. "A random forest classifier based on pixel comparison features for urban LiDAR data". In: *ISPRS journal of photogrammetry and remote sensing* 148 (2019), pp. 75–86.

[177] Jeremie Papon et al. "Voxel cloud connectivity segmentation-supervoxels for point clouds". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2027–2034.

[178] Z Sun et al. "Classification of MLS point clouds in urban scenes using detrended geometric features from supervoxel-based local contexts." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4.2 (2018).

[179] Alexander Velizhev, Roman Shapovalov, and Konrad Schindler. "Implicit shape models for object detection in 3D point clouds". In: *International Society of Photogrammetry and Remote Sensing Congress*. Vol. 2. 2012, p. 2.

[180] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[181] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.

[182] M Zaboli et al. "Classification of mobile terrestrial Lidar point cloud in urban area using local descriptors". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 42 (2019), pp. 1117–1122.

[183] Borja Rodríguez-Cuenca et al. "Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm". In: *Remote Sensing* 7.10 (2015), pp. 12680–12703.

[184] Sherif Ibrahim El-Halawany and Derek D Lichti. "Detecting road poles from mobile terrestrial laser scanning data". In: *GIScience & remote sensing* 50.6 (2013), pp. 704–722.

[185] Beril Sirmacek and Roderik Lindenbergh. "Automatic classification of trees from laser scanning point clouds." In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 2 (2015).

[186] T. Ku et al. "SHREC 2020 Track 3: 3D Point Cloud Semantic Segmentation for Street Scenes". In: *Eurographics Workshop on 3D Object Retrieval, 3DOR 2020, Graz, Austria, September 3-5*. 2020, In press.

[187] Qingyong Hu et al. "RandLA-Net: Efficient semantic segmentation of large-scale point clouds". In: *arXiv preprint arXiv:1911.11236* (2019).

[188] Yuee Liu et al. "Classification of airborne LiDAR intensity data using statistical analysis and Hough transform with application to power line corridors". In: *Digital Image Computing: Techniques and Applications, 2009. DICTA'09.* IEEE. 2009, pp. 462–467.

[189] Y Jwa, G Sohn, and HB Kim. "Automatic 3D powerline reconstruction using airborne LiDAR data". In: *Int. Arch. Photogramm. Remote Sens* 38.Part 3 (2009), W8.

[190] HB Kim and G Sohn. "3D classification of power-line scene from airborne laser scanning data using random forests". In: *Int. Arch. Photogramm. Remote Sens* 38 (2010), pp. 126–132.

[191] HB Kim and G Sohn. "Random forests based multiple classifier system for power-line scene classification". In: *ISPRS Int. Arch. Photogramm., Remote Sens. Spat. Inf. Sci* (2011), pp. 253–258.

[192] Jing Liang et al. "A new power-line extraction method based on airborne LiDAR point cloud data". In: *Image and Data Fusion (ISIDF), 2011 International Symposium on.* IEEE. 2011, pp. 1–4.

[193] Marcel Ritter and Werner Benger. "Reconstructing power cables from lidar data using eigenvector streamlines of the point distribution tensor field". In: (2012).

[194] Bo Guo et al. "Classification of airborne laser scanning data using JointBoost". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 100 (2015), pp. 71–83.

[195] Lingli Zhu and Juha Hyyppä. "Fully-automated power line extraction from airborne laser scanning point clouds in forest areas". In: *Remote Sensing* 6.11 (2014), pp. 11267–11282.

[196] Mohammad Awrangjeb and Mohammad Khairul Islam. "Classifier-Free Detection of Power Line Pylons from Point Cloud Data". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* (2017), pp. 81–87.

[197] Leena Matikainen et al. "Remote sensing methods for power line corridor surveys". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 119 (2016), pp. 10–31.

[198] Robert A McLaughlin. "Extracting transmission lines from airborne LIDAR data". In: *IEEE Geoscience and Remote Sensing Letters* 3.2 (2006), pp. 222–226.

[199] Bo Guo et al. "An improved method for power-line reconstruction from point cloud data". In: *Remote sensing* 8.1 (2016), p. 36.

[200] Liang Cheng et al. "Extraction of urban power lines from vehicle-borne LiDAR data". In: *Remote Sensing* 6.4 (2014), pp. 3302–3320.

[201] Haiyan Guan et al. "Extraction of power-transmission lines from vehicle-borne lidar data". In: *International Journal of Remote Sensing* 37.1 (2016), pp. 229–247.

[202] M Arastounia and DD Lichti. "Automatic extraction of insulators from 3D LiDAR data of an electrical substation". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.5/W2 (2013), pp. 19–24.

[203] E. Viatger. *Postes de alta y baja tensión (1)*. 2010. URL: `https://electricidad-viatger.blogspot.com/2010/05/postes-de-alta-y-baja-tension-1.html/`.

[204] Herbert Edelsbrunner and Ernst P Mücke. "Three-dimensional alpha shapes". In: *ACM Transactions on Graphics (TOG)* 13.1 (1994), pp. 43–72.

[205] Ting On Chan, Derek D Lichti, and Craig L Glennie. "Multi-feature based boresight self-calibration of a terrestrial mobile mapping system". In: *ISPRS journal of photogrammetry and remote sensing* 82 (2013), pp. 112–124.

[206] Russell Eberhart and James Kennedy. "A new optimizer using particle swarm theory". In: *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE. 1995, pp. 39–43.

[207] Radu Bogdan Rusu et al. "Towards 3D point cloud based object maps for household environments". In: *Robotics and Autonomous Systems* 56.11 (2008), pp. 927–941.

[208] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[209] J Li, Wei Gong, and Y Ma. "Atmospheric lidar noise reduction based on ensemble empirical mode decomposition". In: *Int Arch Photogr Remote Sensing Spatial Inf Sci* (2012), pp. 127–129.

[210] A Ullrich and M Pfennigbauer. "Noisy lidar point clouds: impact on information extraction in high-precision lidar surveying". In: *Laser Radar Technology and Applications XXIII*. Vol. 10636. International Society for Optics and Photonics. 2018, p. 106360M.

[211] ES Ferdinandov, VI Tsanev, and BO Todorov. "Turbulence-noise in infrared lidar sensing". In: *Infrared Physics & Technology* 36.1 (1995), pp. 105–111.

[212] MultiMedia LLC. *MS Windows NT Kernel Description*. 1999. URL: `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm` (visited on 09/30/2010).

[213] Somkiat Lerkvarnyu, K Deijhan, and F Cheevasuvit. "Moving average method for time series lidar data". In: *Available: http//www. gisdevelopment. net/aars/acrs/1998/ps3016. shtml* (1998).

[214] Hai-Tao Fang and De-Shuang Huang. "Noise reduction in lidar signal based on discrete wavelet transform". In: *Optics Communications* 233.1-3 (2004), pp. 67–76.

[215] TS Reddy et al. "Noise reduction in LIDAR signal using wavelets". In: *International Journal of Engineering and Technology* 2.1 (2009), pp. 21–28.

[216] M Sarvani, K Raghunath, and S Vijaya Bhaskara Rao. "Lidar signal denoising methods-application to NARL Rayleigh lidar". In: *Journal of Optics* 44.2 (2015), pp. 164–171.

[217] Xuezhen Qin and Jiandong Mao. "Noise reduction for lidar returns using self-adaptive wavelet neural network". In: *Optical Review* 24.3 (2017), pp. 416–427.

[218] Carla Nardinocchi, Gianfranco Forlani, and Primo Zingaretti. "Classification and filtering of laser data". In: *International Archives of Photogrammetry and Remote Sensing* 34.3/W13 (2003).

[219] Rodrigo AA Nobrega, Jose A Quintanilha, and Charles G O'Hara. "A noise-removal approach for Lidar intensity images using anisotropic diffusion filtering to preserve object shape characteristics". In: *Proceedings of the ASPRS 2007 Annual Conference*. 2007.

[220] Huang Zuowei, Huang Yuanjiang, and Huang Jie. "A method for noise removal of lidar point clouds". In: *2013 Third International Conference on Intelligent System Design and Engineering Applications*. IEEE. 2013, pp. 104–107.

[221] Xuelian Cheng et al. "Noise-Aware Unsupervised Deep Lidar-Stereo Fusion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6339–6348.

[222] Jaisiel Santana et al. "Noise reduction automation of liDAR point clouds for modeling and representation of high voltage lines in a 3D virtual globe". In:

[223] Yue Gao et al. "3-D object retrieval with Hausdorff distance learning". In: *IEEE Transactions on industrial electronics* 61.4 (2013), pp. 2088–2098.

[224] Rapidlasso GmbH. *LasTools*. 2020. URL: `https://rapidlasso.com/lastools/` (visited on 06/02/2020).

[225] The Mathworks Company. *Lidar and Point Cloud Processing*. 2020. URL: `https://es.mathworks.com/help/vision/lidar-and-point-cloud-processing.html` (visited on 06/02/2020).

[226] Bogdan Radu Rusu and Steve Cousins. *PCL: Point Cloud Library*. 2020. URL: `https://pointclouds.org/` (visited on 06/02/2020).

[227] Bogdan Radu Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[228] Epic Games. *Unreal Engine - LiDAR Point Cloud Plugin*. 2020. URL: https://docs.unrealengine.com/en-US/Engine/Content/index.html (visited on 06/02/2020).

[229] Markus Schütz. "Potree: Rendering large point clouds in web browsers". In: *Technische Universität Wien, Wiedeń* (2016).

[230] Claus Scheiblauer and Michael Wimmer. "Out-of-core selection and editing of huge point clouds". In: *Computers & Graphics* 35.2 (2011), pp. 342–351.

[231] Howard Butler et al. "plas. io: Open Source, Browser-based WebGL Point Cloud Visualization". In: *AGU Fall Meeting Abstracts*. 2014.

[232] Fugro group. *FugroViewer*. 2020. URL: https://www.fugro.com/about-fugro/our-expertise/technology/fugroviewer (visited on 06/03/2020).

[233] Electricité de France - R&D Division. *CloudCompare - 3D point cloud and mesh processing software. Open Source Project*. 2020. URL: http://www.cloudcompare.org/ (visited on 06/03/2020).

[234] Daniel Girardeau-Montaut et al. "Change detection on points cloud data acquired with a ground laser scanner". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36.part 3 (2005), W19.

[235] Nicolas Brodu and Dimitri Lague. "3D terrestrial LiDAR data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 68 (2012), pp. 121–134.

[236] U.S. Department of Agriculture - Forest Service. *Fusion/LDV Lidar analysis and visualization software*. 2020. URL: http://forsys.sefs.uw.edu/fusion/fusion_overview.html (visited on 06/03/2020).

[237] Terrasolid Company. *Terrascan - Software for LiDAR Data Processing and 3D Vector Data Creation*. 2020. URL: http://www.terrasolid.com/products/terrascanpage.php (visited on 06/03/2020).

[238] José M Santana, Agustín Trujillo, and Sebastián Ortega. "Visualization of Large Point Cloud in Unity." In: *Eurographics Technical Report Series*. 2019.

[239] Technodigit - Hexagon. *3D Reshaper Free Viewer: The 3D scanner software*. 2020. URL: https://www.3dreshaper.com/en/software-en/download-software/free-viewer (visited on 06/06/2020).

[240] Limon20. *Terrascan - Software for LiDAR Data Processing and 3D Vector Data Creation*. 2020. URL: http://www.terrasolid.com/products/terrascanpage.php (visited on 06/03/2020).

[241] ULPGC and Cabildo de Gran Canaria. *Explora Gran Canaria*. 2015. URL: http://193.145.147.50/explora-web/web/ (visited on 06/19/2020).

[242] Sebastián Ortega et al. "Explora Gran Canaria - A mobile sense-based recommender system for hiking trails". In: *2nd Global Conference on Applied Computing in Science and Engineering (ACSE2)*. 2017.

[243] Pablo Fernández et al. "SmartPort: A platform for sensor data monitoring in a seaport based on FIWARE". In: *Sensors* 16.3 (2016), p. 417.

[244] Abbas Harati-Mokhtari et al. "Automatic Identification System (AIS): data reliability and human error implications". In: *The Journal of Navigation* 60.3 (2007), p. 373.

[245] Manuel Ramírez et al. "Digital epigraphic heritage made simple: an android app for exploring 3D roman inscriptions". In: *Proceedings of the 14th Eurographics Workshop on Graphics and Cultural Heritage*. 2016, pp. 179–182.