FEATURE EXTRACTION AND CLASSIFICATION TOOLKIT – (MONTH 12)

Deliverable D4.3.1

Circulation:	PU - Public
Lead partner:	UCL
Contributing partners:	UCL, CNR-IMATI-GE, TUDelft
Authors:	Jan Boehm, Simon Julier, Kun Liu, Ziyi Jiang , Geoffrey Jones (UCL); Michela Spagnuolo, Silvia Biasotti, Andrea Cerri, Simone Pittaluga (CNR-IMATI-GE); Roderik Lindenbergh, Beril Sirmacek (TUDelft)
Quality Controller:	Ewald Quak (SINTEF)
Version:	1.0
Date:	01.11.2013

Consisting of

SINTEF	STIFTELSEN SINTEF, Department of Applied Mathematics, Oslo, Norway
Fraunhofer	Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
CNR-IMATI-GE	Institute for Applied Mathematics and Information Technologies of the National Research Council (CNR-IMATI), Genova, Italy
MOSS	M.O.S.S. Computer Grafik Systeme GmbH (MOSS), Munich, Germany
HRW	HR Wallingford Ltd (HRW), Wallingford, UK
FOMI	Hungarian National Mapping and Cadastral Agency (FÖMI), Institute of Geodesy, Cartography and Remote Sensing, Budapest, Hungary
UCL	University College London (UCL), Research centre for Photogrammetry, 3D Imaging and Metrology, London, UK
TUDelft	Delft University of Technology (TUDelft), Department of Geoscience & Remote Sensing, Delft, The Netherlands
IGN	Institut National de l'Information Géographique et Forestière (IGN), Paris, France
UBO	Université de Bretagne Occidentale (UBO), European Institute for Marine Studies, Brest, France
Ifremer	L'Institut Francais de Recherche pour l´Exploitation de la Mer (Ifremer), Brest, France
Liguria	Regione Liguria, Genova, Italy

DOCUMENT HISTORY

Version	Issue Date	Stage	Content and Changes
0.1	Sep 23 2013		First Draft
0.3	Oct 04 2013		Comments collected
0.5	Oct 13 2013		Integrate service metadata
0.7	Oct 24 2013		Merging Contributions
0.8	Oct 25 2013		Merging Contributions
0.9	Oct 26 2013		Proof reading
0.92	Oct 28 2013		Merging Contributions
0.94	Oct 31 2013		Quality Control
0.96	Nov 1 2013		Quality Control
1.0	Nov 1 2013		Version to be submitted to the Project Officer

EXECUTIVE SUMMARY

This report presents the progress made during the first year in the Task 4.3 *Feature extraction, classification and correlation*. It documents the development process, choices of tools and libraries. It lists the software prototypes which were implemented and released, documents their underlying algorithms and presents the results of their testing.

By the end of the first year 10 processing services were developed. The processing services contain two internal and expert services, which are needed for pre-processing of data and as an aid in development. These are the static tiling of LAS files and the quick visualization of point clouds. The remaining eight processing services are targeted towards the end user. They were motivated by the user requirements and were chosen to support the implementation of a full scenario work-flow. The processes include point cloud to regular grid conversion, peak point detection, outlier classification, 3D local keypoint extraction, tree-crown detection, detection of flow lines and drainage basins, extraction of critical points and filtering point clouds by attributes.

Two of the software tools are already implemented in the MapReduce paradigm, the choice for the IQmulus Cloud computing architecture. The processing services chosen for this are the static tiling of LAS files and the point cloud to regular grid conversion. These are typical example processes for mass data processing.

Mainly three data sets were used to test the processing services. They are all point cloud data sets. One data set is a mobile mapping data set and two are airborne LiDAR data sets.

TABLE OF CONTENTS

Executive summary 3

Table of contents 4

1 Introduction 5

2 Base Libraries & Development Tools 6

2.1 Hadoop 6

2.2 liblas 8

2.3 PCL 9

2.4 GitHub 10

2.5 CINDER 11

2.6 Programming Languages & Compilers 12

2.6.1 C++ 12

2.6.2 Boost 12

2.6.3 MATLAB 13

3 Test Data Sets 14

3.1 Bloomsbury 14

3.2 London 15

3.3 Liguria 16

4 Individual Services 18

4.1 Internal/Expert Services 18

4.1.1 Static tiling of LAS file 18

4.1.2 Quick Visualization for Point Clouds 21

4.2 End-User Services 22

4.2.1 Point Cloud to Regular Grid 22

4.2.2 Peak Points 24

4.2.3 Outlier Classification in Point Clouds 25

4.2.4 3D Local keypoint extraction from point clouds 28

4.2.5 Tree crown recognition from mobile mapping point clouds 29

4.2.6 Detection of flow lines and drainage basins 33

4.2.7 Extraction of Critical Points, lines and regions from grids and triangulations 35

4.2.8 Filter Point Cloud by Attribute & Coordinate 37

5 References 40

6 Appendix – Service Information Tables 41

1 INTRODUCTION

This report presents the progress made during the first year in the task 4.3 “Feature extraction, classification and correlation”. It describes the processing services developed for IQmulus as part of work package 4. Task 4.3 is targeted at a semantic enrichment of the data sets as laid out in the Description of Work. Semantic enrichment refers to automatic extraction and annotation of high-level information by segmentation or classification processes.

Within the first year ten processing services were developed and are now released at the end of the first period. The processing services contain two internal and expert services, which are needed for pre-processing of data and as an aid in development. The remaining eight processing services are targeted towards the end user. They were motivated by the user requirements and were chosen to support the implementation of a full scenario work-flow.

All processing services are described in a uniform way using standardized service information tables. These tables give the service name, a description of its functionality, a description of the algorithm, the connection to the use cases and further characteristics. These tables are contained in the Appendix of this document.

A more detailed description of the processing services is given in Section 4 of this document. Here the details of the implementation and results obtained on the test data sets are presented. For each process figures of the processed data and visualizations of the results are given to illustrate the algorithm and demonstrate its capabilities.

The test data sets used to obtain these results are described in detail in Section 3. Mainly three datasets were used to test the processing services. They are all point cloud datasets. One data set is a mobile mapping data set of the London Bloomsbury area. Two of the data sets are airborne LiDAR data sets. One data set was acquired over London, the other over the coastal area of Liguria. The size of the data sets is in the order of 4 million points.

To successfully implement the algorithms several base libraries and other tools for software development were used. To completely document the development process the main libraries and tools are introduced in Section 2. Hadoop is the choice for the IQmulus architecture and is therefore a crucial library for the services implemented in the MapReduce paradigm. Two of the software prototypes are already implemented in the MapReduce paradigm, the choice for the IQmulus cloud computing architecture. We chose two typical example processes for mass data processing, namely static tiling of LAS files and the point cloud to regular grid conversion for implementation in the MapReduce paradigm.

Other libraries that were used in the development of software in task 4.3 include libLAS a library to read point clouds in the standardized LAS format and PCL a general purpose point cloud processing library. Section 2.6 further discusses the situation on programming languages and compilers and recommendations made for the developments within task 4.3.

2 BASE LIBRARIES & DEVELOPMENT TOOLS

In the following, we briefly review the main libraries used to build the services for feature extraction, classification, and correlation. These libraries have been collected and extensively discussed in collaboration with other tasks in WP4 and will be further reviewed during the development of the processing services.

2.1 HADOOP

The Apache Hadoop library is an open-source software library that provides a framework for the distributed processing of large data sets across clusters of computers. It supports a simple programming model, known as MapReduce. MapReduce was originally derived from Google's paper on the programming model (Dean and Ghemawat, 2008). The strength of this framework is that it provides automated parallelization of the computation and automated distribution of data, while it possesses excellent scaling characteristics. Hadoop consists of a number of modules. For the processing in IQmulus, we focus on only two components - the Hadoop Distributed File System (HDFS) and the MapReduce engine.

HDFS is a distributed, scalable and rack-aware file system. It is therefore able to handle very large data sets, e.g. several terabytes. It is the default file system assumed for all processing jobs under Hadoop. Although Hadoop can support other distributed file systems from different cloud computing service vendors, such as Amazon S3 and IBM GPFS, HDFS is exclusively used for the current WP4 service implementations. The main programming API for HDFS is in Java. A C API is also provided, known as libhdfs, based on Java Native Interface (JNI), and implements a subset of the Java HDFS API.

For IQmulus, a distributed point cloud LAS file reader/writer for Hadoop has been implemented for Task 4.3 using the Java interface. Testing using currently available data shows stability. More testing of the file handler is expected in future once the processing framework has been decided.

Hadoop currently supports two versions of the MapReduce engine: MRv1 and MRv2. MRv1 is the default one used for most commercially supported Hadoop distributions. MRv2, also known as YARN, is under active development by the Hadoop community and is expected to be the future default. The main improvement in YARN is the replacement of the JobTracker and TaskerTracker in MRv1 with a new set of daemons such as ResourceManager and ApplicationManager. The new daemons are more agnostic to the application, which enables the application to implement programming models beyond MapReduce, such as machine learning and general cluster computing. YARN maintains to a large extent the API backwards compatibility with MRv1.

The current Task 4.3 implementation follows the API provided by MRv1, because no decision has yet been made for the specific processing framework and version at the point of writing this document. A small percentage of code adaptation for using the new YARN engine is expected for future work.

Hadoop is developed in Java, and Java offers the most flexible approach to implement distributed jobs under Hadoop. There are interfaces for other programming languages in Hadoop, known as Pipes and Streaming. While Pipes are designed for job implementations (map/reduce functions) in C++, Streaming is more intended for text processing using applications produced in other languages.

The C++ applications linked against Hadoop Pipes inherit similar class interfaces as in Java, and each application is run as a sub-process of the Java Virtual Machine. The communication is via sockets, and hence the key and value classes in Pipes are byte buffers, represented in C++ as STL strings. The Hadoop Pipes only support custom Mappers and Reducers in C++, more sophisticated distributed file handling and map output consolidation using Partitioner still need to be implemented in Java. A point cloud to regular grid demo has been produced using Hadoop Pipes for Task 4.3.

The Hadoop Streaming API uses UNIX standard streams as the interface between Hadoop and the application. Therefore any application that reads from standard input and writes to standard output can support Mapper or Reducer functions in a MapReduce job. A combination of Streaming and Pipes is expected for future service implementation depending on the specific form of service input.

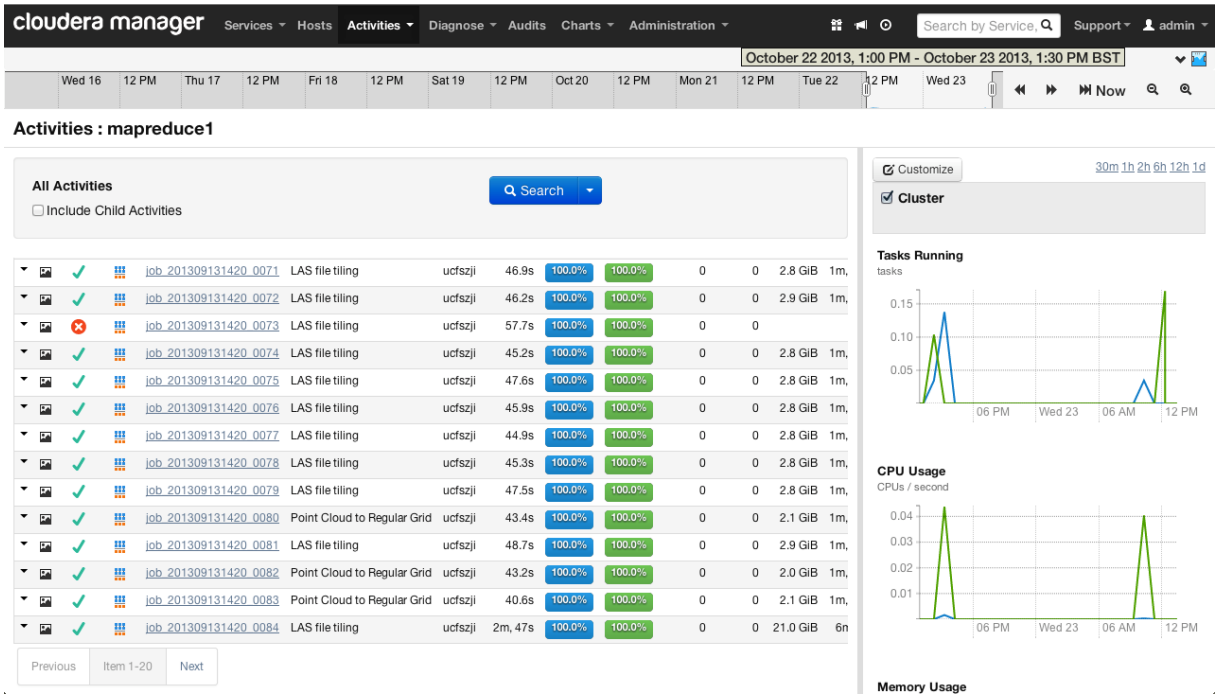

Figure 1: Cloudera Manager showing the services “LAS file tiling” and “Point Cloud to regular Grid” on a four node test cluster.

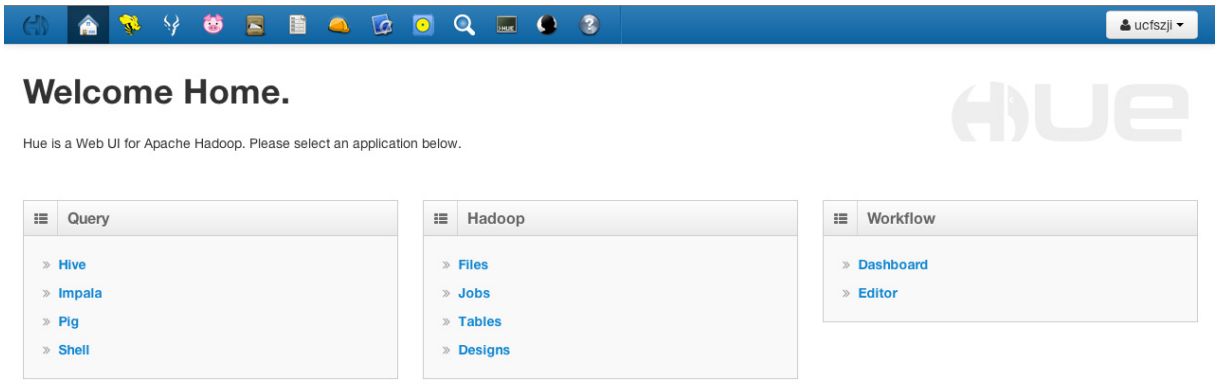

Figure 2: Hadoop start screen.

There are several existing commercially supported Hadoop distributions. Testing for Hadoop clusters and its upper infrastructure services is performed using the Cloudera CDH4 distributions with deployment through Vagrant. A 4-node Hadoop cluster running Hadoop 2.0 is used to test all current Task 4.3 service implementations (start screen in Figure 2). The Cloudera cluster management (see Figure 1) provides control through its own web-based interfaces, with rich access to job management, HDFS file management, etc., in addition to existing Hadoop command line functions and a web information page. WP3 will provide a Hadoop cluster for the future development. Still it can be reasonable to use a local cluster in the following phases of IQmulus for development and testing. The local cluster can even be connected with the main cluster to form a bigger cluster.

2.2 LIBLAS

libLAS is an open-source C/C++ library for reading and writing the LAS LiDAR point cloud format (Butler et al., 2013). The library is implemented in C/C++. In addition, libLAS also provides a Python API and can be used in various other languages such as C# and VB.NET. libLAS is distributed under the BSD license. libLAS currently provides full support for all LAS file formats up to version 1.2. Newer versions of the LAS file format include additions such as waveform data and a longer header. However, these are not currently supported by libLAS and therefore the further developments of this library need close monitoring. Potentially the library could be replaced with lidarformat ("lidarformat - C++ point cloud library," 2013) or both could be used in combination.

Advanced LAS data manipulation can be achieved both through the tool applications provided by libLAS or through its programmatic API. Functions provided by libLAS include input/output with LAS binary data, as well as various point record based operations including data filtering and statistics production. The prerequisite libraries for building libLAS include Boost version 1.38 or higher, with GDAL, libgeotiff and laszip to provide additional functionalities. CMake is used as the default building tool.

The latest stable release for libLAS is version 1.7. Services in Task 4.3 with single node implementations all use libLAS to deal with LAS file input/output. Version 1.7 of libLAS is stable for currently planned Task 4.3 services using the test data. Future development is expected to use the same version of libLAS.

2.3 PCL

The Point Cloud Library (PCL) is a BSD-licensed, open-source C++ library for n-D point cloud and 3D geometry processing. The library contains various state-of-the-art algorithms, which support a range of operations including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be efficiently used to implement the services in Task 4.3, e.g., outliner classification and feature points.

PCL is written in C++ and is fully templated. It is a cross-platform library that can be built under Windows, Linux and Mac OS X. In addition, the library heavily utilizes Streaming SIMD Extensions (SSE) to obtain high performance on modern CPUs. Most of mathematical operations are implemented using the C++ linear algebra library Eigen. Moreover, OpenMP is supported in PCL, which provides parallel implementation for many algorithms such as normal estimation and moving least square fitting. In geometry processing, k-nearest neighbour (KNN) search is frequently used and thus a fast KNN module is crucial to the whole framework. PCL mainly uses the library FLANN, i.e., Fast Library for Approximate Nearest Neighbours (Muja and Lowe, 2009), which is about one order of magnitude faster in query time than previously available approximate near neighbour search software (Mount and Arya, 1998). Algorithms for spatial data management, including octrees, are also provided. All the modules in PCL pass data by means of Boost shared pointer without explicitly copying. Currently the shared pointer is a part of the C++11 standard.

PCL is developed with efficiency and performance in mind, and it includes many techniques and libraries to optimise performance. Nonetheless, it can be very easily integrated into a user's own project. The same basic interface is used in all the algorithms as shown in Figure 3. First, an object is created for processing. One example would be an object to perform normal estimation. Subsequently the input point cloud is passed to the created object. Parameters are set thereafter. Finally the processing is executed and the results are obtained. A snippet of example code to estimate normals is presented in Figure 4. As illustrated in Figure 4, the basic interface is compact and simple. Moreover, since all the data structures in PCL are written using templates, this provides enough flexibility to modify the library to satisfy special requirements, e.g., introducing new point types, but without the need to create intermediate wrapper code.

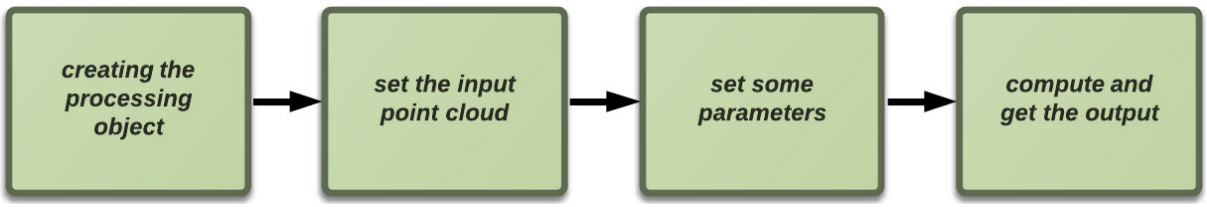

Figure 3: Basic interface scheme for PCL code.

Due to its efficiency and convenience, PCL is used as a core library for point cloud processing in Task 4.3. In our current implementation, PCL version 1.6.0 is used. The library is well maintained by a large developer group. The latest stable release is the version 1.7.0 and version 2.0 is expected possibly in 2014. Therefore, in the future we will continue using the library as a core library for point cloud processing.

```
// Create the normal estimation class, and pass the input dataset to it
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
ne.setInputCloud (cloud);

// Create an empty kdtree representation, and pass it to the normal estimation object.
// Its content will be filled inside the object, based on the given input dataset (as no other
search surface is given).
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());
ne.setSearchMethod (tree);

// Output datasets
pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>);

// Use all neighbors in a sphere of radius 3cm
ne.setRadiusSearch (0.03);

// Compute the features
ne.compute (*cloud_normals);
```

Figure 4: A C++ example which shows how to compute normals in a point could using PCL

2.4 GITHUB

During the first year of development it was necessary to exchange code with the partners. This was for example needed to distribute example code for the first code camp (Darmstadt, June 2013), which explains the combined use of libLAS and PCL, or the normal estimation from a point cloud. We chose to use GitHub to share code amongst partners. The same mechanism was used to distribute the first code demonstrating the use of MapReduce and Hadoop as a preparation of the second code camp (Paris, October 2013).

GitHub is a web-based software project hosting service that uses the Git version control system. GitHub offers both private and public repositories. Although GitHub actively encourages users to choose the open-source-licensing model for their project, the default copyright laws apply for projects in absence of a license.

GitHub provides rich code view and development functionality through its web interface. The basic concept of branching and version control for the code is based on its underlying version control system Git. With proper SSH setup, the GitHub repository can be accessed through local Git applications on any authorised development machine. In addition to the basic Git functionality, the GitHub web interface also provides other easy-to-access functions such as Wiki pages for code documentation; a live issue tracking system allows developer interaction and visual branch and pull request management.

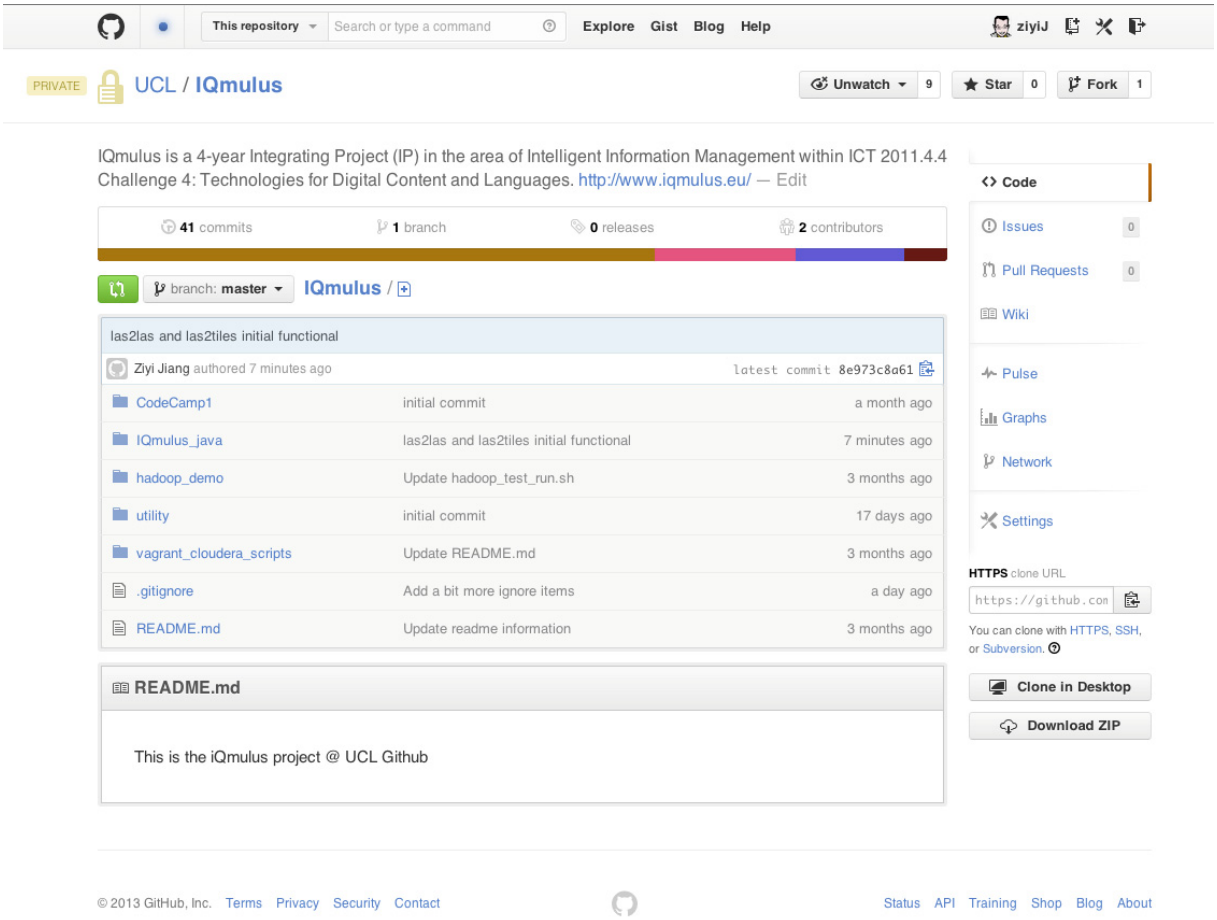

Figure 5: GitHub Web interface with Task 4.3 code samples

Git is a distributed version control system used by GitHub for its backbone code management. Git was originally designed and developed for Linux kernel development in 2005. For working with GitHub, Task 4.3 is currently using a private GitHub repository maintained by UCL to provide code sharing and hosting for the service development activities (Figure 5). Future code sharing activity is expected to continue through the GitHub service.

2.5 CINDER

"Cinder provides a powerful, intuitive toolbox for programming graphics, audio, video, networking, image processing and computational geometry. Cinder is cross-platform, and in general the exact same code works under Mac OS X, Windows and a growing list of other platforms — most recently the iPhone and iPad." – sourced from http://libcinder.org/about

Cinder in the context of IQmulus is a fast (yet elegant) way to visualise the point cloud data. Cinder has minimal dependence on third party libraries and is designed to leverage platform native capabilities where available, as such it is lighter weight than other cross-platform front-end frameworks such as Qt. Internally Cinder utilises aspects of the latest C++11 standard to create an internally memory managed framework for efficiently marshalling against memory leaks, both for framework objects and, via the use of transparent C++ wrappers, of OpenGL resources as well.

Cinder was originally designed as a cross-platform framework for graphics artists, and although it has matured into a more broadly applicable tool kit, the core premise of flexibility and ease of use has remained. This was apparent by the straightforward integration to make the Cinder

interface with PCL work for drawing and loading point clouds. There are also several optional, yet actively maintained, interface classes for leveraging other mainstream libraries such as OpenCV (current minimum version 2.1). The down sides of this open-ended design are that beyond text rendering, the UI construction and management is up to the user to implement, although a positive of this is that it is almost guaranteed not to look like a cookie cutter application which can happen when using frameworks that have more comprehensive(/inflexible) off-the-shelf offerings.

The Cinder library is an industry-strength tool originally developed for internal use by the Barbarian Group and it continues to be developed and maintained by an active community. It has recently received the endorsement of Herb Sutter for being a "flexible and fun" framework for rapid development. For the code developed in Task 4.3 version 0.8.5 was used, which depends on OpenGL and Boost.

2.6 PROGRAMMING LANGUAGES & COMPILERS

In the following sections we document the guidelines and recommendations we made for the software developed in Task 4.3 and we try to provide the rationales for the choices.

2.6.1 C++

In this phase of the project many processes were implemented in C++. This is perfectly reasonable for fast processes often implemented as single threaded processes. However, as the partners are using different platforms and consequently different compilers, this creates some problems when exchanging code. We try to give some recommendations to harmonize code development and make the resulting code more (re-)usable.

For one we recommend the use of a C++ style guide. We chose the Google C++ Style Guide ("Google C++ Style Guide," 2013), as it is a well-written, comprehensive and easily accessible style guide. It is important that we use this as a guide not a rule.

We further recommend the use of existing general purpose libraries such as Boost (see below) as it is peer-reviewed code and nicely documented. Many of the other libraries suggested for Task 4.3 use Boost already.

At the moment C++11 ("The Standard : Standard C++," 2013) is not supported by all compilers. Again in order to improve the exchange of code we recommend switching to a C++11 compatible compiler when they become available. In particular these are

1. Visual Studio C++ 2013
2. GNU GCC >= 4.8 (provides full C++11 support)

2.6.2 Boost

"Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing. Release 1.52 contains over eighty individual libraries". [Wikipedia page]

Given its broad range of functionality, Boost is required by libLAS, PCL and Cinder and is used by commercial packages including Matlab and Adobe Acrobat. The libraries are mostly released

under the Boost Software License, which allows for the use of the libraries in commercial products.

Given its widespread use by multiple IQmulus libraries, this library will continue to be used throughout the duration of the project.

2.6.3 MATLAB

MATLAB ("MATLAB - The Language of Technical Computing," 2013) is both a numerical computing tool and a programming language. Within Task 4.3 MATLAB will not be used to deliver processes. However, it is an important prototyping tool. Particularly the statistical toolbox provides operators that are useful for Task 4.3. The machine learning tools are an important reference and benchmark for processes developed within IQmulus. The class of operators with special importance are supervised classification algorithms. MATLAB provides the following state-of-the-art implementations:

- Boosted and bagged decision trees
- Support vector machines (SVMs)
- Naïve Bayes classifiers
- Nearest neighbours (kNN)
- Discriminant analysis
- Neural networks

Accessing readily implemented classification algorithms allows testing and benchmarking different classification approaches, without the implementation overhead. Since the MATLAB operators can currently be limited in performance and memory, they are not suited for dissemination. For the algorithms selected native implementations are preferred.

3 TEST DATA SETS

During development several data sets were used to test the implementations. The following sections give an overview of the main data sets. Where possible we give the source, extent and size of the data and describe the approximate location.

3.1 BLOOMSBURY

The Bloomsbury dataset is a mobile mapping LiDAR point cloud acquired by ABA Surveying. The ABA mapping van is equipped with three laser scanner heads. The trajectory is computed from GPS, IMU and wheel sensor data. Each scanner head can acquire up to 1,000,000 points per second. Figure 6 shows the mapping van and an overview of the point cloud. The test scene is a trajectory in the Bloomsbury area of London around the campus of UCL.

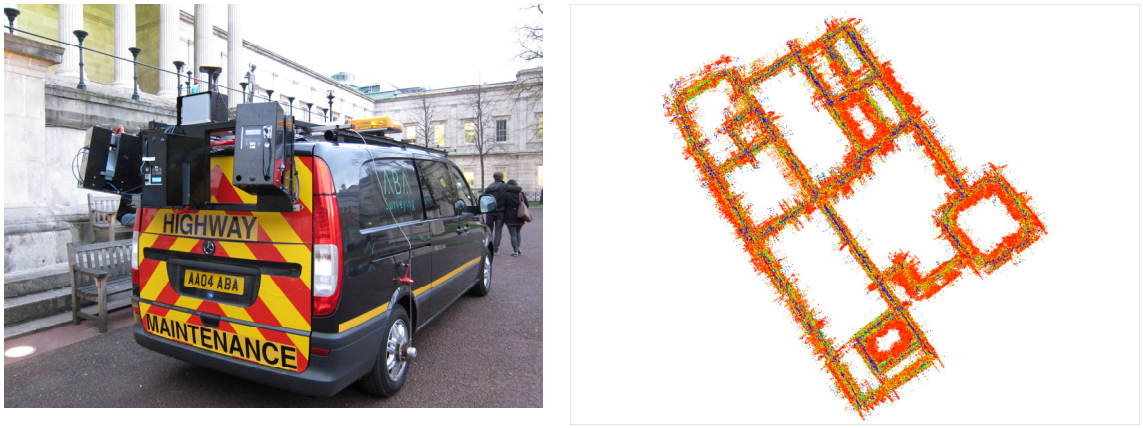

Figure 6: Mobile Mapping van (left) and captured data over Bloomsbury, London (right).

The point cloud has a spacing of roughly 10 cm along the trajectory of the van and on average of 2 cm across the driving direction. A detail of the point cloud demonstrating the high point density is given in Figure 7. Besides the 3D coordinates the backscattered energy of the laser beam is also recorded for each point. This can be displayed as an intensity value. For the test a single tile of the dataset with 4 448 634 points was used.

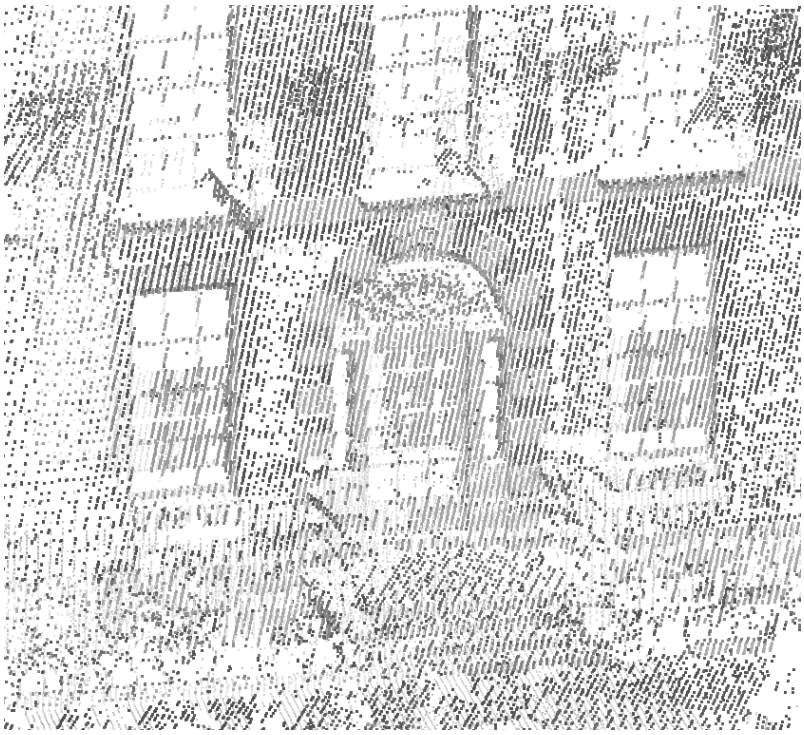

Figure 7: Detail of the point cloud.

3.2 LONDON

The London dataset is an airborne LiDAR point cloud over London. The data is stored in a LAS file version 1.2. A single tile contains about 2,000,000 points. The point density is about 4 points per square meter. The dataset represents the current standard product for commercially available fixed wing airborne data acquisition. The features stored per point are the attributes typically expected for an airborne dataset:

- 'X'
- 'Y'
- 'Z'
- 'Intensity'
- 'Return Number'
- 'Number of Returns'
- 'Scan Direction'
- 'Flightline Edge'
- 'Classification'
- 'Scan Angle Rank'
- 'User Data'
- 'Point Source ID'
- 'Time'

The Classification attribute conforms to the ASPRS standard. The visualization of a tile is given in Figure 8. As an example, the specific tile consists of the following classes (number of points given first):

- 5 840 Unclassified
- 584 307 Ground

- 316 287 Low Vegetation
- 225 053 Medium Vegetation
- 974 272 High Vegetation
- 2 752 Low Point (noise)
- 1 635 Reserved for ASPRS Definition

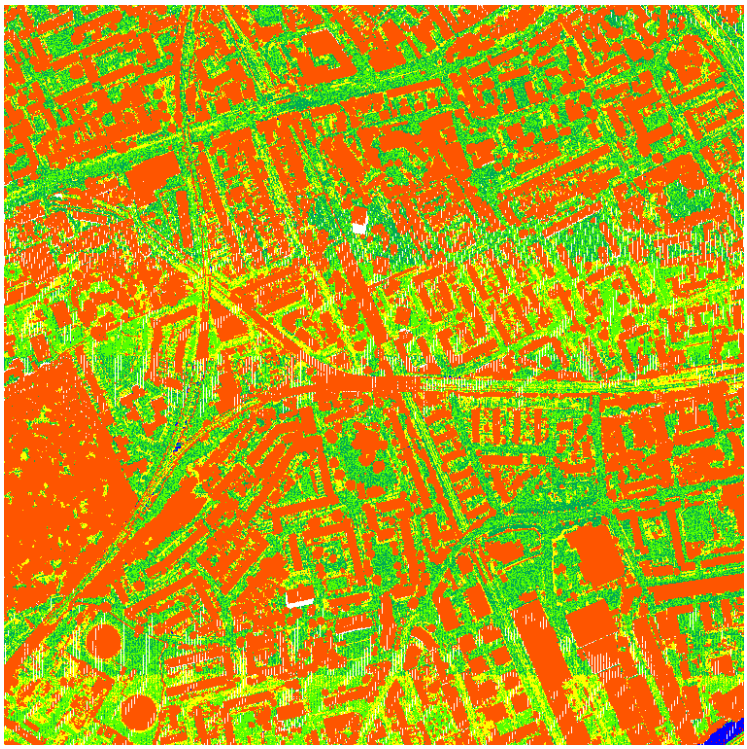

Figure 8: Single tile of London airborne LiDAR data set using Classification attribute for colouring.

3.3 LIGURIA

The Liguria dataset is another airborne LiDAR dataset. Figure 9 shows the dataset using the scan angle for colouring which visualizes the different flight strips used for the acquisition. The single tile dataset contains 4 134 193 points. The point cloud covers about 0.9 square kilometres. On average the point density is 4 points per square meter. The dataset contains only two distinct classes for classification:

- 3 726 654 Unclassified
- 407 539 Ground

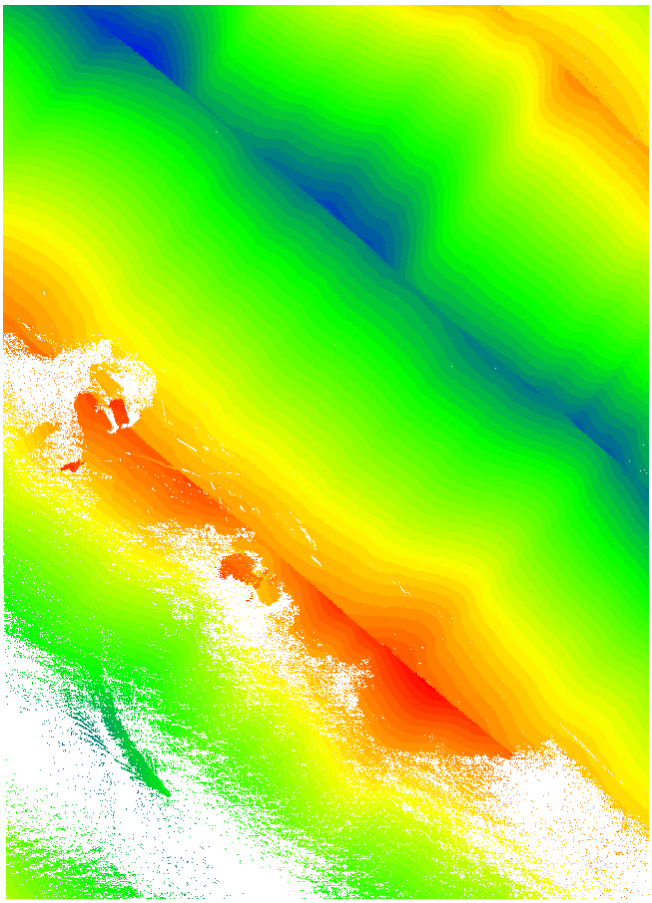

Figure 9: Airborne LiDAR data set over Liguria using Scan Angle for colouring.

4 INDIVIDUAL SERVICES

One of the core outputs of Task 4.3 in the first year is the development and implementation of a toolkit for feature extraction and classification. This section documents the development of the toolkit and motivates the decisions that were made during the implementation. For each process we demonstrate the results on one of the example data sets. The following table gives an overview of all ten processing services included in the toolkit.

Processing Task	Lead Partner
Static Tiling of a LAS File	UCL
Quick Visualization	UCL
Point Cloud to Regular Grid	UCL
Peak Points	UCL
Outlier Classification in Point Clouds	UCL
3D Local Keypoint Extraction from Point Clouds	TUDelft
Tree crown recognition from mobile mapping point clouds	UCL
Detection of Flow Lines and Drainage Basins	IMATI
Critical Points	IMATI
Filter Point Cloud by Attribute & Coordinate	UCL

4.1 INTERNAL/EXPERT SERVICES

A set of internal processes is needed for pre-processing of data and as an aid for development. Since the results of most processes can be easily assessed by visualization, a quick visualization tool is essential for the developers. This is outside of the scope of WP5, which targets the development of visualisation tools for end users. Static tiling of (potentially) large LAS files helps to keep data size for local nodes low. It is an essential tool for distributing load across a cluster.

4.1.1 Static tiling of LAS file

This service divides a single large LAS file into multiple smaller tiles, each of which is better suited for processing at an individual node in the cluster. The input is an unordered point cloud in LAS format. The input parameters are the tile size and the tile-overlapping ratio; both are indicated in the x- and y-axis direction. The output is a set of multiple unordered point clouds in LAS format, one for each tile.

This service is intended to facilitate other services. It is especially helpful for those services that are not able to handle data sizes beyond the memory size. It effectively reduces the size of individual LAS files. Using overlapping tiles makes it possible to address edge effects which can arise at the tile borders (e.g., filtering).

The service is implemented using the MapReduce framework in Java and therefore scales with the size of the Hadoop cluster. The LAS file I/O functionality is a standalone package that can be used by other Task 4.2 and Task 4.3 services which use LAS files for point cloud storage.

A brief description of the service implementation:

```
## Pre-Mapper:
Input: x_tile_size
Input: y_tile_size
Input: x_overlap_ratio
Input: y_overlap_ratio

# Read LAS file header for point cloud boundary
x_max, x_min, y_max, y_min = readHeader();

# Calculate number of tiles in x- and y-axis
x_tile_num = ceil((x_max - x_min) / x_tile_size)
y_tile_num = ceil((y_max - y_min) / y_tile_size)

set_num_of_reduce_task(x_tile_num * y_tile_num)

Output: x_tile_num
Output: y_tile_num

## Mapper: (Key1, Value1) -> (Key2, Value2)
Input: k1 # Key1, the offset of LAS point record in the file, Bytes
Input: v1 # Value1, the input LAS point record

# Calculate tile number in x- and y-axis
x_tile = ceil((v1.x - x_min) / x_tile_size)
y_tile = ceil((v1.y - y_min) / y_tile_size)

# Calculate the tile id
tile_id = x_tile_num * (y_tile - 1) + x_tile;

k2 = tile_id
v2 = (v1.x, v1.y, v1.z)

Output: k2 # Key2, the tile id for individual point
Output: v2 # Value2, the point

# Calculate tile number with inflated tile_size
x_overlap = ceil((v1.x - x_min) / (x_tile_size * (1 +
x_overlap_ratio)))
y_overlap = ceil((v1.y - y_min) / (y_tile_size * (1 +
y_overlap_ratio)))

# Calculate the inflated tile id
tile_id_overlap = x_tile_num * (y_overlap - 1) + x_overlap;

# If the point ends up in different tile, output the new id as well
```

```
if (tile_id_overlap != tile_id)
      k2' = tile_id_overlap
      v2' = v2
      Output: k2'
      Output: v2'

## Partitioner: (Key2, Value2) -> Reducer id
Input: k2
Output: k2 - 1 # tile id corresponds to the reducer id

## Reducer: (Key2, Value2s) -> Output
Input: k2
Input: v2s # All the values with the same k2

# Send point to the output file
print(v2)

Output: Tiled LAS file
```

The execution of the code is wrapped within Bash script to provide Hadoop job setup and after-job data retrieval in addition to the main code functionality. Figure 10 and Figure 11 demonstrate a tiling example of the Liguria data set with 500m by 500m tile size and 10% overlapping ratio. The original point cloud is coloured in grey in the middle of the figure, whereas the tiles are coloured and scatter around the original data for demonstration.

The performance of current service implementation depends on the specific Hadoop cluster deployment and user inputs. For example, the execution time for a single-node Hadoop run on the Liguria data set with the user inputs as shown in Figure 11 is about 50 seconds.

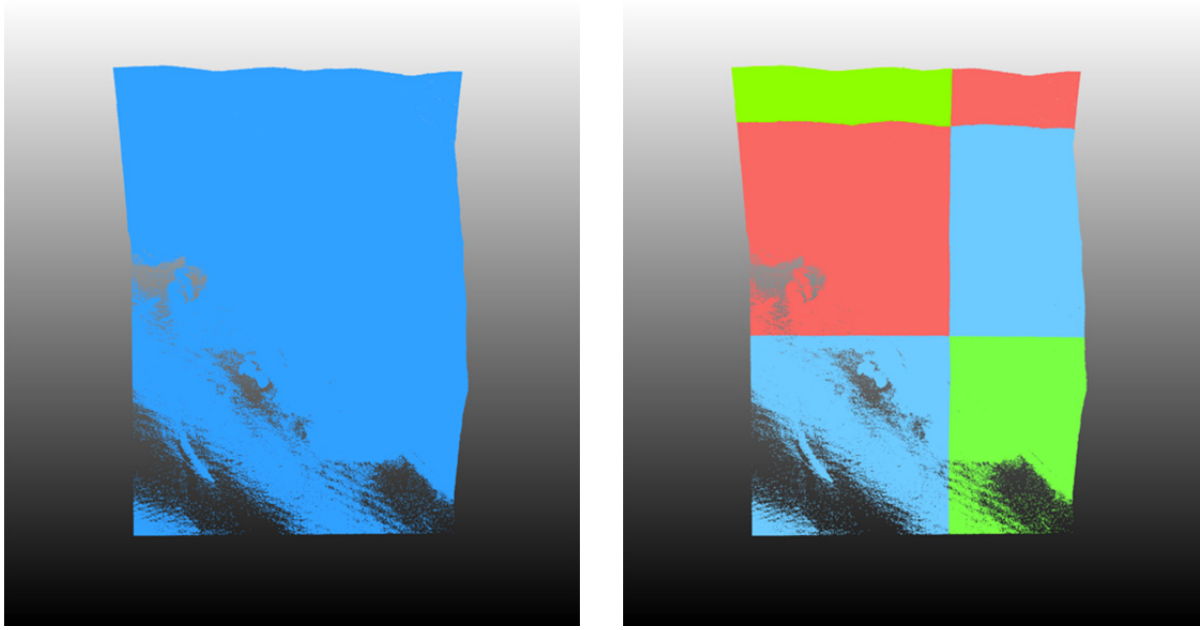

Figure 10: Dividing a point cloud dataset (left) into six separate tiles (right).

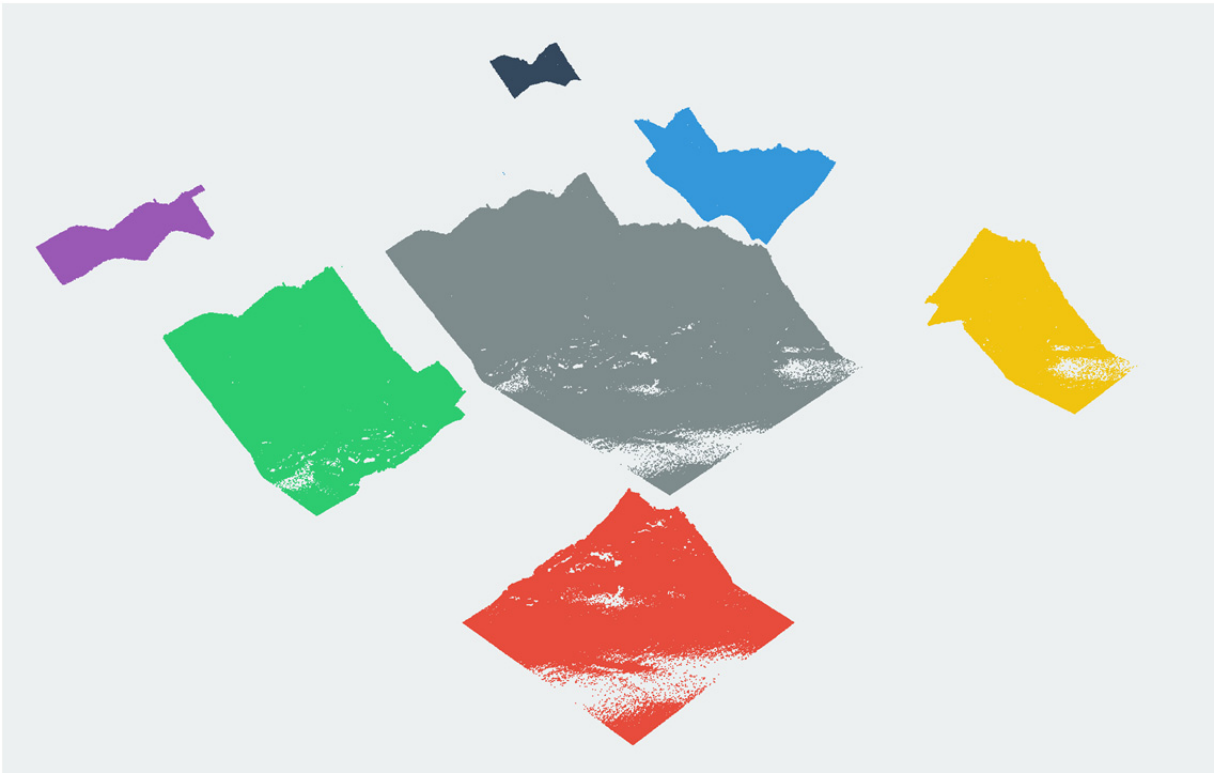

Figure 11: Three-dimensional rendering of tiling (original data grey, resulting tiles in colour).

4.1.2 Quick Visualization for Point Clouds

As explained above, for algorithm development there is a requirement to provide techniques to rapidly visualize data such as intermediate results. This is not the same as the full visualization system which will be developed under WP5 to support end users. Rather, we require simple tools which will support debugging and rapid prototyping. As stated previously in Section 3.3, PCL is utilized as the core library for point cloud processing. PCL supports several ways to visualize clouds in its native format. However, there is no support when an external format (such as LAS) must be visualized.

Given these issues, a tool for quick visualization has been implemented using the light C/C++ library AntTweakBar. Visualization is achieved using OpenGL and GLUT. The framework is such that it is easy to introduce custom code to provide further visualization and interaction capabilities. Moreover, AntTweakBar facilitates the creation of graphical user interfaces (GUI) for the application. By the means of AntTweakBar, bar, slider, and button can be added to the parameter panel without a lot of coding work, which accelerates the procedure of prototyping.

Figure 12 shows the graphical user interface (GUI) provided by the current visualization tool. The figure in the LIGURIA point cloud data set is visualized in the central window of the UI. The semi-transparent blue panel at the top-left can be used to set parameter values. Three ways to set parameter values are provided: rotating slider, increasing or decreasing by buttons and typing in input fields. In the bottom-left, the help panel is displayed including shortcuts and a simple description of the UI.

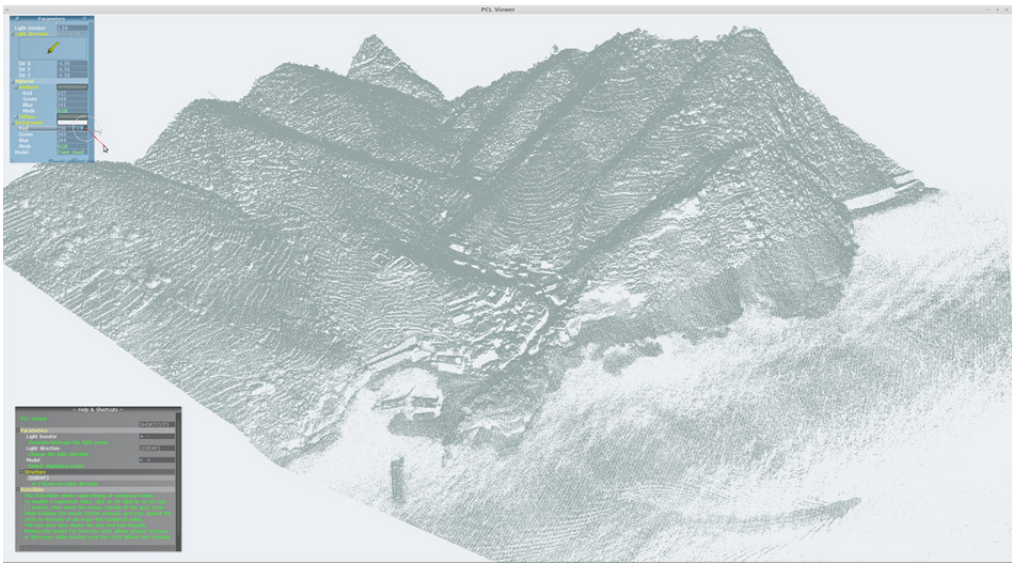

Figure 12: Visualization of an example point cloud.

4.2 END-USER SERVICES

The following processes are targeted at the end users. They draw their motivation from the user requirements and they are building blocks in accommodating the required work flows.

4.2.1 Point Cloud to Regular Grid

This service converts an unordered point cloud into a grid representation in 2.5 D. The input is an unordered point cloud in LAS format. The input parameters are the grid sampling distance ("spacing") in both x- and y-direction. The output is a gridded point cloud in LAS format. It is also possible to output GeoTIFF for compatibility with certain GIS software packages.

This service is implemented using the MapReduce framework in Java and therefore scales with the size of the Hadoop cluster. The approach is to generate keys based on point coordinates which identify the grid positions. The LAS file I/O functionality is implemented using the shared library. This service has also been implemented in C++ using Hadoop Pipes to test the combination of C++ and Java for Hadoop parallelisation usage.

A brief description of the service implementation:

```
## Mapper: (Key1, Value1) -> (Key2, Value2)
Input: k1 # Key1, the offset of LAS point record in the file, Bytes
Input: v1 # Value1, the input LAS point record
Input: grid_res_x # the x grid sampling distance, in meters
Input: grid_res_y # the x grid sampling distance, in meters

# Calculate lower tile coordinates in x- and y-axis
lower_grid_x = floor(v1.x / grid_res_x) * grid_res_x
lower_grid_y = floor(v1.y / grid_res_y) * grid_res_y

k2 = (lower_grid_x, lower_grid_y)
v2 = v1.z

Output: k2 # Key2, the tile coordinates
Output: v2 # Value2, the height information
```

```
## Reducer: (Key2, Value2s) -> Output
Input: k2
Input: v2s # All the values with the same k2

# Loop through all the value with the same key
for v2 in v2s:
      sum_z += v2.z

z_out = sum_z / v2s.size() # Taking average

# Send point to the output file
print(k2.lower_grid_x, k2.lower_grid_y, z_out)

Output: Gridded LAS file
```

The execution of the code is wrapped with Bash script to provide Hadoop job setup and after-job data retrieval in addition to the main code functionality. Figure 13 demonstrates a gridding example of the Liguria data set with a sampling distance of 10m by 10m. The original point cloud is coloured in grey, and the generated grid is coloured in yellow.

The performance of the current service implementation depends on the specific Hadoop cluster deployment, and user inputs. For example, the execution time for a single-node Hadoop run on the Liguria data set with the user inputs as shown in Figure 13 is about 40 seconds.

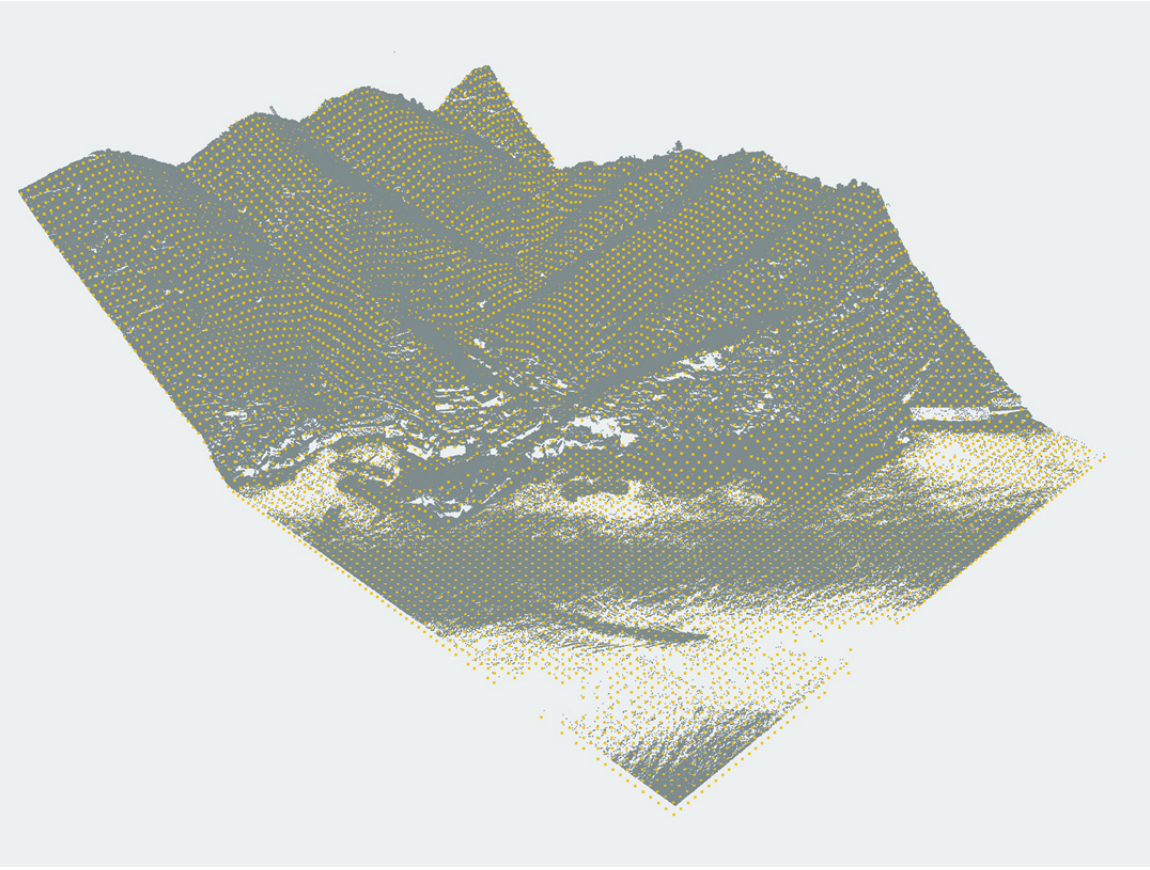

Figure 13: Regular grid displayed on top of dense point cloud.

4.2.2 Peak Points

This processing service detects local peak points. For this process a peak is not defined by its local shape, but by the relation of its height with respect to its neighbours. The detector for peak points can best be described as a non-maximum suppression approach. Each point in the local neighbourhood is inspected. If the height of the point is larger than any of its neighbours it is kept as a peak. It is discarded otherwise.

The implementation of the approach is based on a kd-tree using the FLANN library. To build a kd-tree requires O(n log n) time with respect to the size of the input cloud (n). The search of the tree is of complexity log n for a fixed search radius (Preparata and Shamos, 1985). The parameters for the search are the radius and a sensitivity parameter. This is a heuristic introduced to avoid detecting peaks in predominantly flat regions. Results are shown in Figure 14.

The pseudo code description of the implemented approach is the following

```
Foreach point p in cloud
      Q = RadiusSearch(p, Radius)
      If height(p)> height(every q in Q) AND
         height(p)> height(lowest q in Q + Sensitivity)
            Add p to Peaks
      Endif
Endfor
```

The operator is currently implemented for a single CPU in a sequential manner. The operator can be extended to large datasets, using tiling and distribution over several nodes. The Operator synopsis on the command line is

```
C:\>iQmulus_peak.exe --help
Allowed options:
  --help                    produce help message
  --input arg (=test.las)   set input file
  --output arg (=test.shp)  set output file
  --radius arg (=5)         set search radius
  --sensitivity arg (=1)    set sensitivity
```

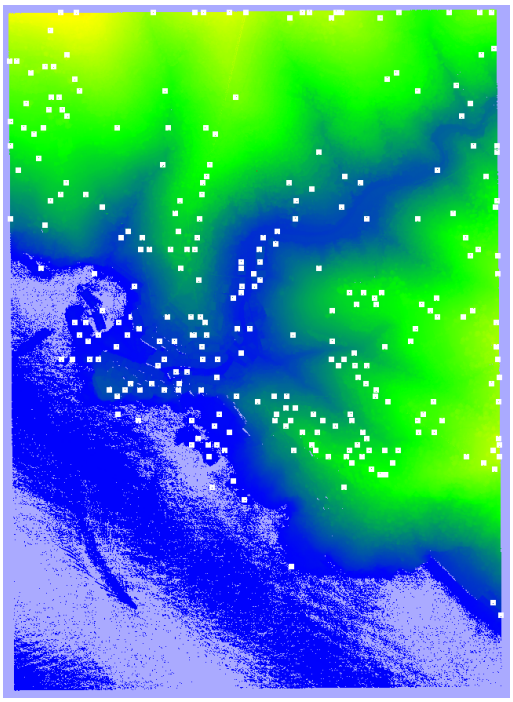

Figure 14: Peak points shown in white from an unordered point cloud.

4.2.3 Outlier Classification in Point Clouds

This processing service performs statistical outlier detection to identify isolated points which are usually spurious and arise from errors in the capture process. One input is the threshold of how many standard deviations away from the global mean neighbourhood distance is acceptable to qualify inliers. The size k refers to the k neighbours to sample in order to generate the statistical profile of the input data. This separates the input point cloud into inliers and outliers. Results of this processing are shown in Figure 16 and Figure 17.

Algorithm in pseudo code:

```
Input k          # nearest neighbourhood size
Input sd_tol     # tolerance threshold
Input cloud      # input point could

# calculate the mean distance between points in all neighbourhoods
For all p_i in cloud
        u_i = mean_dist( p_i , nearest_neighbours( k , p_i ) )

# calculate global statistics for neighbourhoods in this cloud
U = mean( u )
SD = standard_deviation( u )

Threshold = U + ( SD * sd_tol )

# divide  the cloud into inliers and outliers
For all u_i in u
        If u_i <= Threshold
                inliers.push_back( p_i )
```

```
        else
                outliers.push_back( p_i )

Output inliers   # points whose neighbourhood is within tolerance
Output outliers  # points whose neighbourhood is outside tolerance
```

Results

Figure 15: Problematic outliers in the Bloomsbury data set, due to laser beam reflection (points below street level).

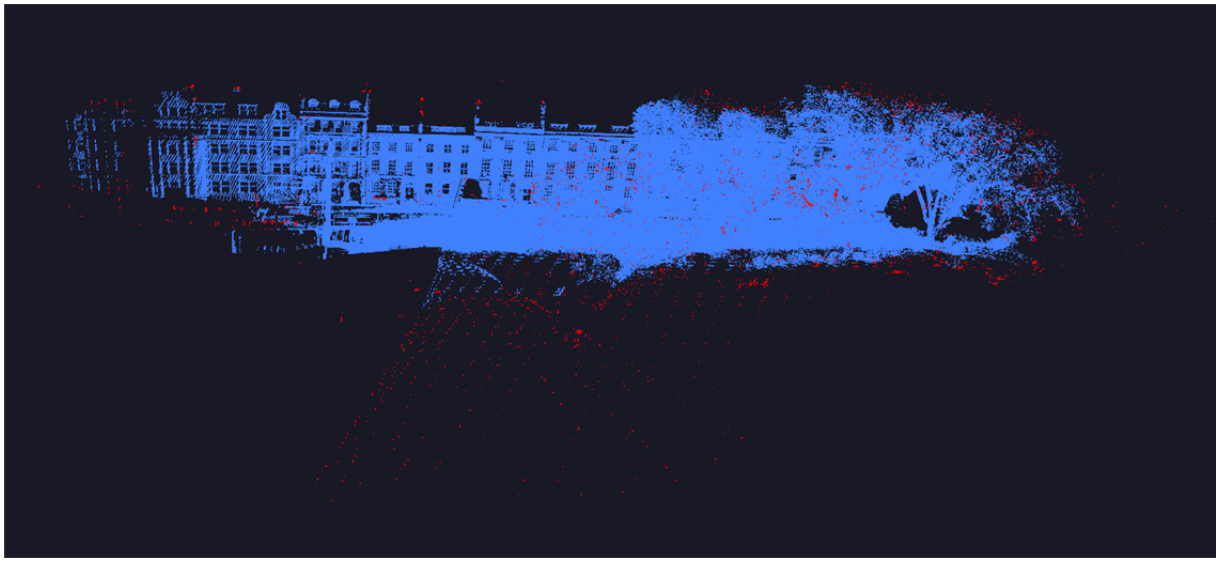

Figure 16: Automatic removal of outliers (marked red). Some vegetation points were falsely classified as outliers.

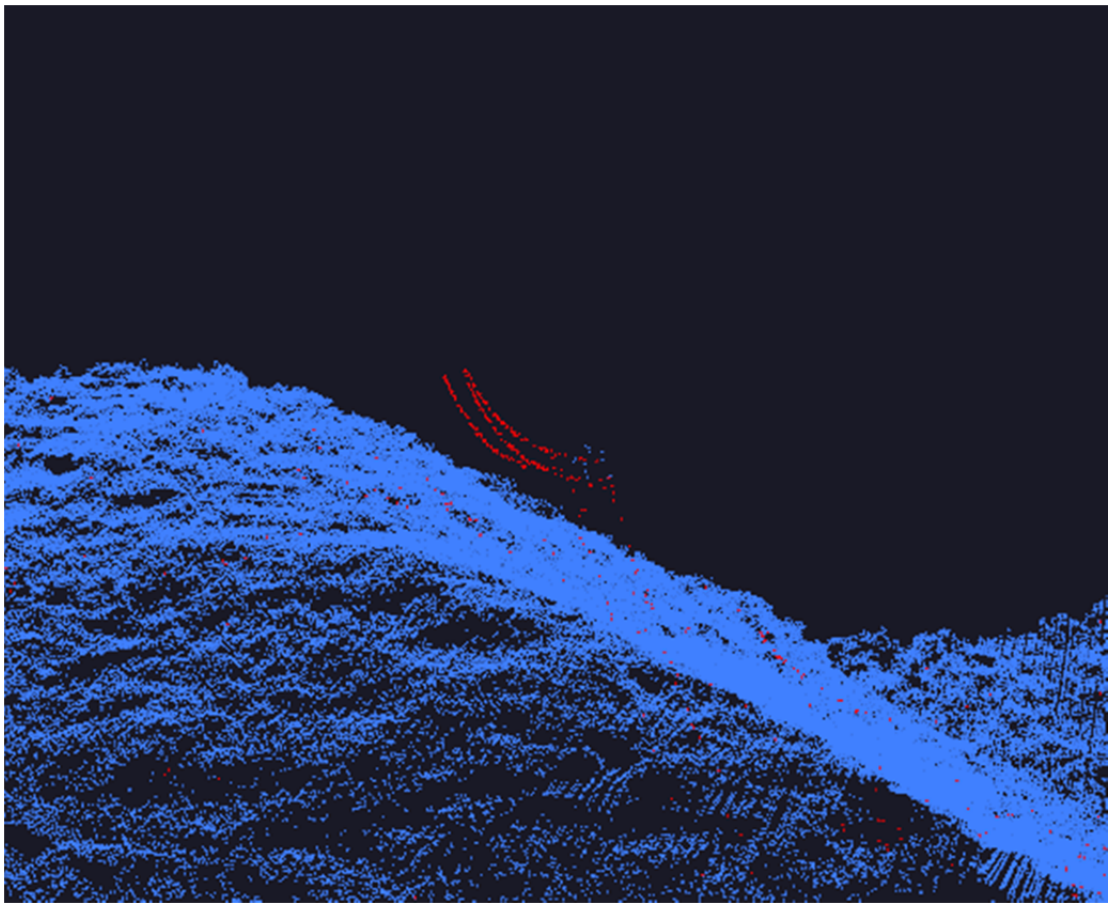

Figure 17: Example of removing power-lines over the terrain by automated outlier detection.

Implementation

The service is implemented in C++. The core algorithm uses PCL which in turn depends on

- Boost > 1.46
- Flann > 1.71
- Eigen > 3
- Vtk > 5.6

The service is connected with an optional visualization tool for development and debugging which depends on Cinder and requires

- OpenGL
- Boost

The code of the service builds and runs on Linux and Windows.

The computational time in relation to data size scales approximately at (n log n) based on the time taken for the construction of the kd-tree and nearest neighbour queries.

Since the algorithm computes the global mean distance for the input data this could result in varying identification of outliers across different input data sets. This constrains the locality of the algorithm and complicates distribution of the data.

To overcome this problem either the mean and standard deviation (SD) could be calculated as sample mean and SD. Alternatively a synchronisation step could gather all the tile local means and variance and combine them to create a global threshold which is then used for all tiles. This results in:

1) Tile data
2) Compute mean + SD of neighbourhoods on each tile
3) Gather + combine to get the global mean + SD
4) Compute global tolerance threshold
5) For each tile separate outliers using global threshold

For a 4 million point data set the execution time is in the range of 5-20 seconds depending on machine capabilities.

The current implementation runs on a single node single thread implementation and could potentially process larger data sets in shorter times through parallelisation across tile configurations.

Future Development

nanoflann (https://code.google.com/p/nanoflann/) may be an alternative to using the PCL kd-tree construction and approximate nearest neighbour (ANN) search interfaces. This would reduce the dependencies (nanoflann has only STL dependencies), further it could possibly offer up to 2x speed up on search and construction of the kd-tree.

This would be reliant on a successful tiling of the input data into contiguous tiles and on the synchronisation step mentioned above.

4.2.4 3D Local keypoint extraction from point clouds

Features in general are distinctive locations of both 2D and 3D input data. They can be either points given by certain coordinates, or small regions. In our case we consider points to be represented by their 3D coordinates and are thus referred to as keypoints. We refer to them as low-level features to indicate that the extracted keypoint might be meaningless for the end user, however it contains crucial information for other internal modules of the IQmulus system for further processing and to generate end results. One such module is the registration of two data sets for which corresponding keypoints are crucial input.

The procedure can be summarized as follows:

1. Using the LibLAS library to read the point cloud data which is in ".las" file format
2. Calculate the surface normals using the PCL library
3. Extract the local maxima of the surface normal values as local features
4. Use the VTK library to display the input point cloud (optional)
5. Use the VTK library to label the locations on the 3D display where the local features are extracted (optional)

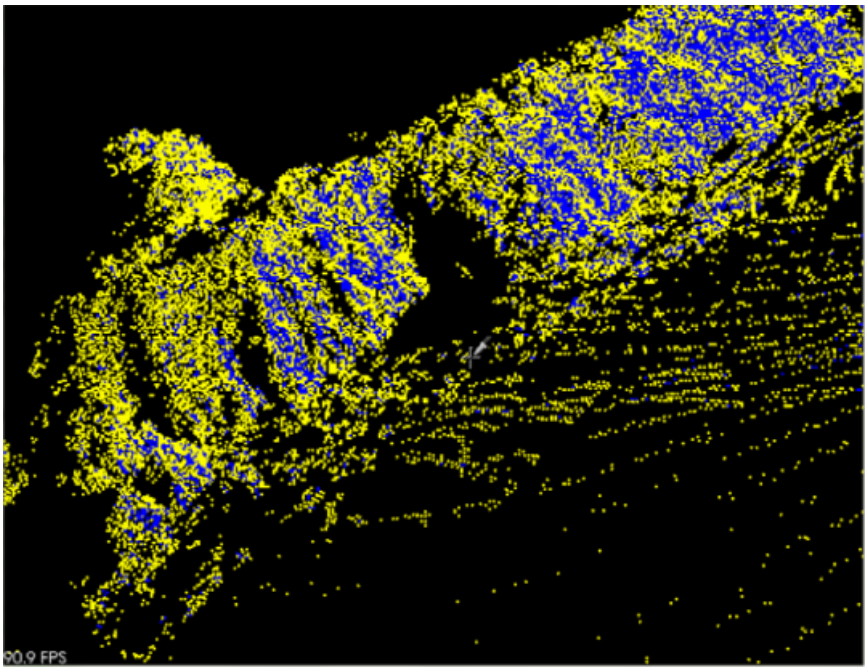

Figure 18: Example output of local keypoints on the Brittany point cloud based on analysis of local normals (DoN).

4.2.5 Tree crown recognition from mobile mapping point clouds

The purpose of this processing service is to classify the point cloud data according to underlying object type. For this initial implementation, the classification attempts to separate man made from naturally occurring features. It exploits the fact that manmade objects tend to be more planar.

A standard approach to inspect the local spatial distribution of points is to compute a Principle Component Analysis (PCA) (Demantké et al., 2011). This results in three Eigenvectors and the corresponding Eigenvalues λ_i. The Eigenvalues relate to the semi axis lengths $\sigma_i = \sqrt{\lambda_i}$ of the corresponding ellipsoid. There are three extreme cases:

- $\sigma_1 >> \sigma_2 \geq \sigma_3$: mainly distributed along one axis direction
- $\sigma_1 \approx \sigma_2 >> \sigma_3$: mainly distributed along two axis directions
- $\sigma_1 \approx \sigma_2 \approx \sigma_3$: distributed along all directions

Dimensionality descriptors normalize these eigenvalue relationships, and define how close the individual neighbourhood shape is to one of the three cases described above:

- $D_1 = (\sigma_1 - \sigma_2)/\sigma_1$
- $D_2 = (\sigma_2 - \sigma_3)/\sigma_1$
- $D_3 = \sigma_3/\sigma_1$

Two classification algorithms have been developed. Classification algorithm A is a hand-crafted decision tree using descriptors D_2 and D_3. This serves as a baseline algorithm to see if machine

learning can actually improve classification results. The classification algorithm A was shown to evaluate the two manually labelled test data sets with only planar surfaces or only tree-crowns. Results on the full data set are shown in Figure 21.

	Planar Surface	Tree Crowns
Total points	77566	58955
Correctly labelled	96%	83%
Falsely labelled	4%	17%

Classification algorithm B is a random forest classification using all three descriptors. We currently use the MATLAB implementation to perform random forest training and classification. The classifier B was evaluated on the same test data sets.

	Planar Surface	Tree Crowns
Total points	77566	58955
Correctly labelled	99%	96 %
Falsely labelled	1%	4%

For random forest classification the training phase already gives inside into classification accuracy. The out-of-bag error (Breiman, 2001) is a vital indicator of this and also helps to choose the number of trees which are needed. Figure 19 shows that the graph saturates for around 15 trees. While the classification results on the test data look satisfactory, we can observe misclassifications when the classifier B is applied to the full tile (Figure 22). We attribute this to over-fitting, but further investigations are needed. At the moment we deliver the classical decision tree, but we see a strong potential in the random forest approach.

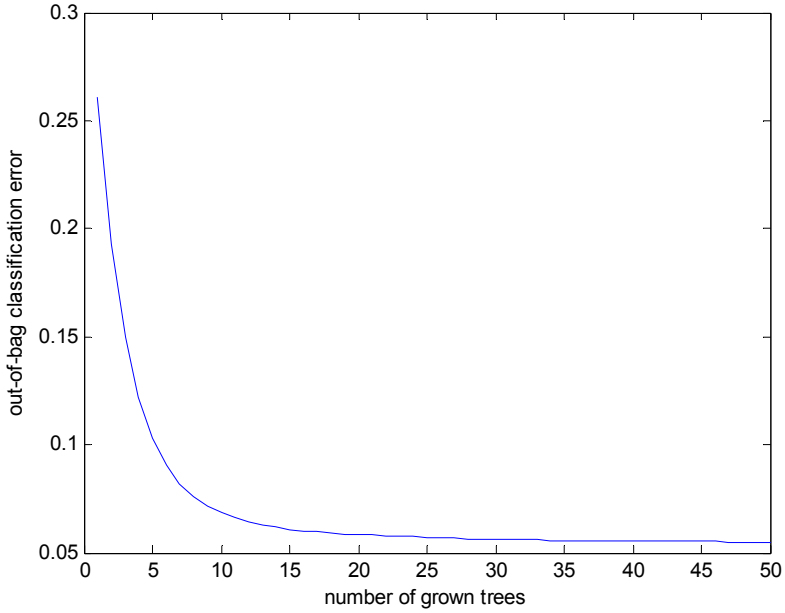

Figure 19: Out-of-bag error versus the number of trees grown.

Figure 20: Mobile mapping dataset Bloomsbury. The grey values correspond to the amount of backscattered energy roughly similar to intensity.

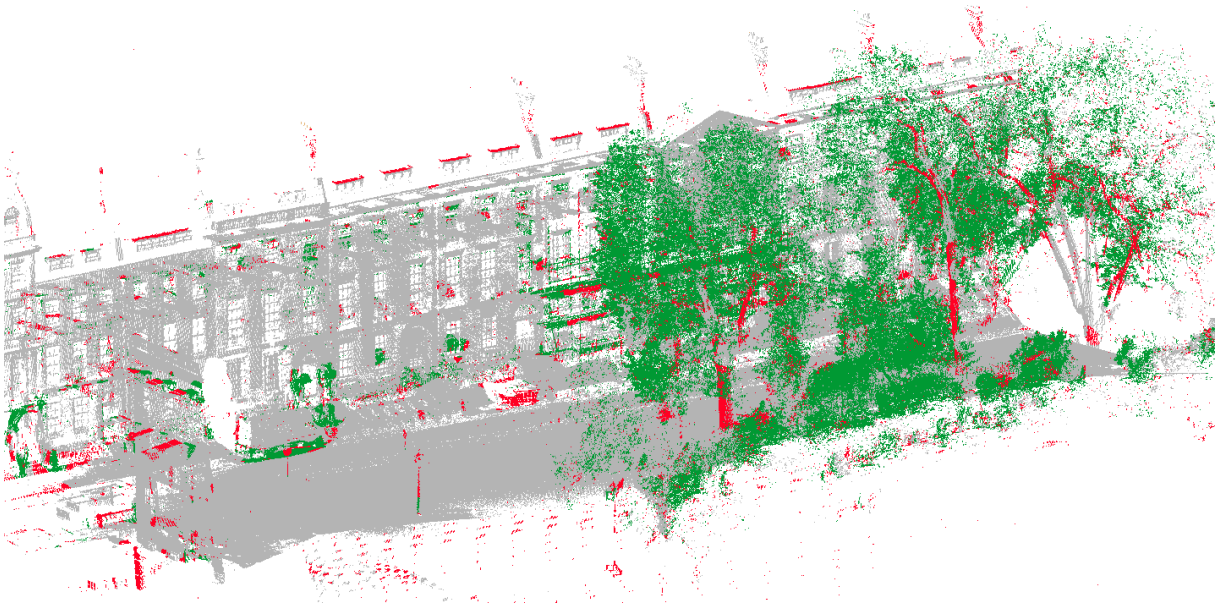

Figure 21: Classification using Decision Tree. Man-made planar surfaces (Grey), tree crowns (Green) and unclassified (Red).

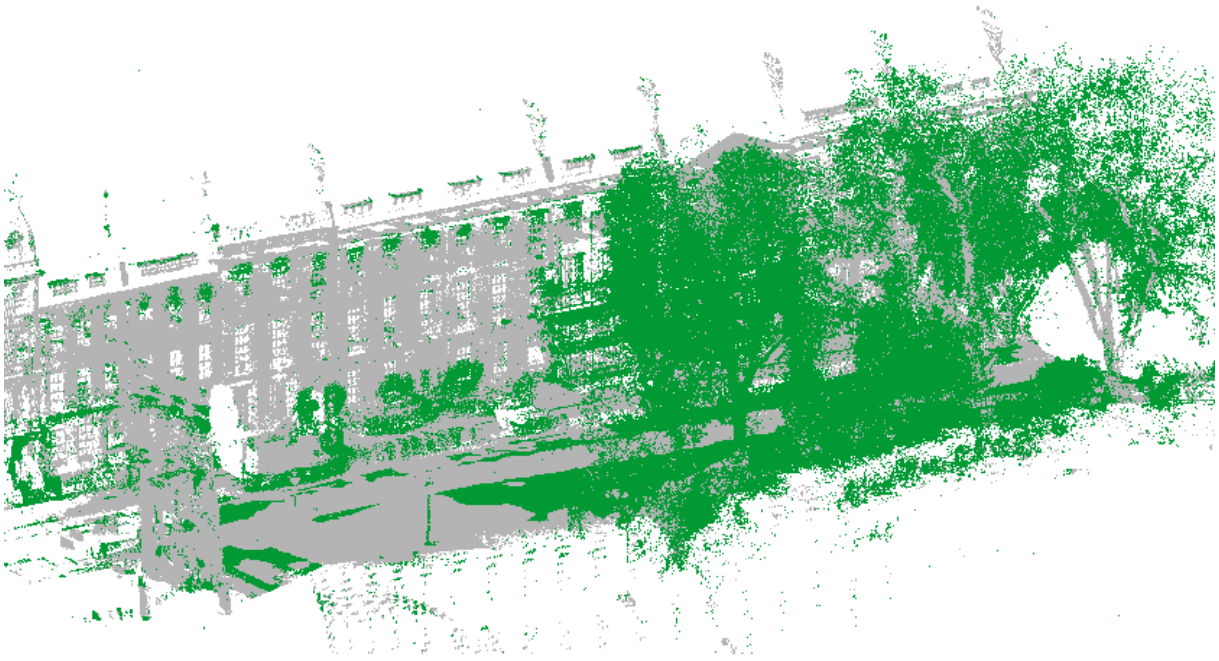

Figure 22: Classification using Random Forest. Man-made planar surfaces (Grey) and tree crowns (Green). Misclassification can be observed on the Street Plane

The current implementation is a single-threaded sequential implementation of the decision tree in C++. While the MATLAB random forest shows better statistics on the test data, we observe some artefacts on the full data set. This requires further investigation.

There are documented approaches in the literature using parallelization to speed up classification (Low et al., 2012) with some using MapReduce as a paradigm (Lin and Kolcz, 2012). An Apache machine learning library is available that is implemented on top of Hadoop which provides high-level machine learning algorithms ("Apache Mahout: Scalable machine

learning and data mining," 2013). The use of Hadoop would make it an ideal fit for the IQmulus architecture. We will investigate the use of the library to bring this algorithm to the Cloud.

4.2.6 Detection of flow lines and drainage basins

This service answers the specific needs of hydrologists to know the extent and the shape of the area that drains towards an outlet section. The knowledge of drainage basins is a key step in the hydrological modelling for protection from flood. This is especially true for small basins, where the flooding is fast and the overlap of rainfall data with the shape of catchment can help the decision making pipeline in civil protection. This specific requirement comes from Regione Liguria and its departments in charge of civil protection and environment monitoring. Given its peculiar orography, the Liguria region is indeed particularly critical as most of the dangerous rivers flow out from small basins, usually with a surface less of 10 km2.

The reference user story is the following: "I would like to have a shape file of the Ligurian small basins (< 10 km2) derived from a high resolution DTM exploiting the available Lidar data of Regione Liguria to derive the DTM and a cut interpolated rainfall field over the shape file."

Functionality

The service released with the "watershedDEM" provides the functionality to extract flow lines and drainage basins from terrain models. In the current release of the service, we have considered as input the terrain represented as a gridded DEM, and in the next project period we will develop the service providing the same functionality when starting from triangulations. The service takes a threshold parameter that filters the minimum size of the basin that the service will extract.

Considering the agreed data representation types, the service accepts as input gridded point clouds (format: geoTIFF). The output is a gridded point cloud (format: geoTIFF), where the cells store the X, Y coordinates of the point and the Z value stores the unique integer identifier of the basin the cell belongs to. As output, the service also produces a shapefile (format: .shp .shx .dbf) storing the mainstream reaches. The services can save, as intermediate output, slope, accumulation area and flow directions map.

Libraries used

The service is based on the TauDEM toolkit for gridded data released under GPL by D.G. Tarboton (Tarboton, 2005). The original software is implemented in C++ and uses parallelization based on MPI. The TauDEM toolkit does not need external libraries except the standard C++ and the MPI compiler. To get the shapefile of basins it requires the GDAL library.

Underlying Algorithm

The service execution can be described by the following pseudo code:

```
#Read and load the input DEM, and the optional threshold following
#the constant drop law (Broscoe, 1959)

Read_input (DEM, threshold);

Find_and_remove_pits_from (DEM);

#For each cell in the DEM, compute the flow directions, using a 8-
#window neighbourhood. This step writes two geoTiff files: one for
```

```
#direction map and one for slope activity.
Find_flow_directions (DEM);

#For each cell in the DEM, compute the cumulative number of cells
#that drains into downslope cells. The output is a geoTIFF with
#the computed value as attribute.
Compute_contributing_area (DEM);

Run a Peuker-Douglas filter over topography (DEM);

With this step produces a skeleton of streams network as a grid of
{0,1} values.

Apply a threshold to delineate stream network;

#The threshold is related to the similarity between the mean stream
#drop of first order streams and the mean stream drop of higher
#streams according to the constant drop law.

Extract and write watershed and streamnet (output a watershed
geoTiff and a streamnet shapefile);
```

Execution architecture of current implementation

The different steps explained in the previous section are implemented in parallel using a Message Passing Interface (MPI). The input gridded data is divided in equally sized horizontal strips. The number of strips corresponds to the number of processes. For each strip, the correct size of memory is allocated with an overlap of two more lines for each strip in order to exchange data with the neighbouring strip; this task is performed by a specific function (Tesfa et al., 2011).

Example Output

As shown in Figure 23 each extracted basin represents the contributing area for river reaches between intersections. Each colour displays a different drainage area. The black thick lines are the main streams. This basin extraction has been tested on the Liguria dataset and in more detail using a Digital Surface Model derived from an inverse distance weighting interpolation of LiDAR data of the Vernazza area.

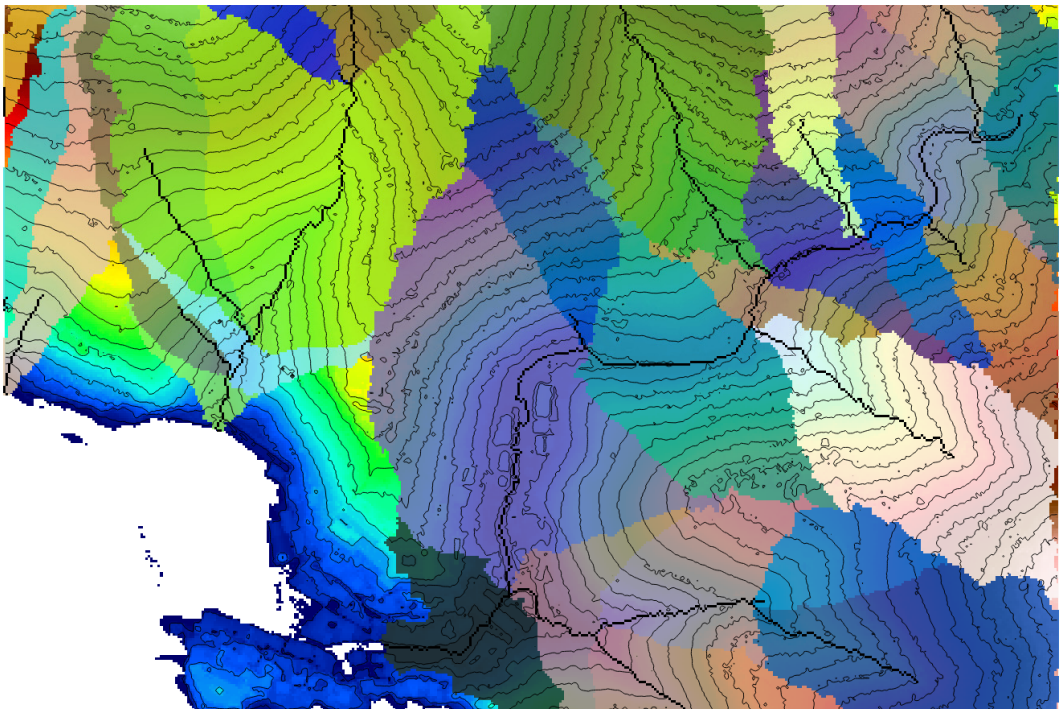

Figure 23: Basins extraction on the Liguria dataset.

Performance on Test Data set

The parallelization of the code allows executing the code on data sets of tens of millions of cells in less than a minute on an eight core desktop computer. The service has been tested on the Liguria data set and on the dataset provided for the IQmulus Processing Contest (see D8.8.1). Some results obtained from the contest data sets are not as good as expected. When the process is confronted with a large flat area (6.4*10^7 cells), such as encountered over the sea, it spends a lot of time (tens of minutes) to find a possible downslope direction. This issue has to be fixed in the next release to improve the behaviour over flat areas.

Suitability for Map/Reduce

This algorithm may be implemented following a MapReduce paradigm. A mapper may emit each cell index as a key and pass as value the neighbourhood list of characteristic values, e.g., the elevation value. The reducer may reduce each cell index with a value, for example, of flow direction and so on for every local computation step (direction map, contributing area ...).

4.2.7 Extraction of Critical Points, lines and regions from grids and triangulations

This service for surface characterization is based on the extraction of the critical points (in the case of a differentiable function points where all partial derivatives are zero) from either a triangle mesh or a grid. The knowledge about critical points is crucial for understanding and organizing the topological structure of a surface. Unfortunately the hypothesis that a surface is only continuous does not guarantee that the associated height function is Morse nor that it is derivable.

The approach proposed in this service is based on the popular characterisation of triangle meshes proposed by (Banchoff, 1970). Such a method uses a local point-wise criterion to detect

and classify critical points: for example, triangle meshes are analysed by checking the height difference (z-value) between a vertex and the adjacent ones in its star-neighbourhood.

The original algorithm is then extended by:

- considering critical lines and triangles that are maxima or minima with respect to the z-value;
- adding an (optional) threshold to establish if two points can be considered at the same height or not.

The algorithm extracts a list of critical points (maxima, minima and saddles) with respect to the z-coordinates and flat non-isolated maxima/minima lines and regions represented in a shape file. The computational cost of the service for the surface characterization is O(n), where n is the number of input vertices. Figure 24 shows two examples of the surface characterization the service extracts from a triangle mesh.

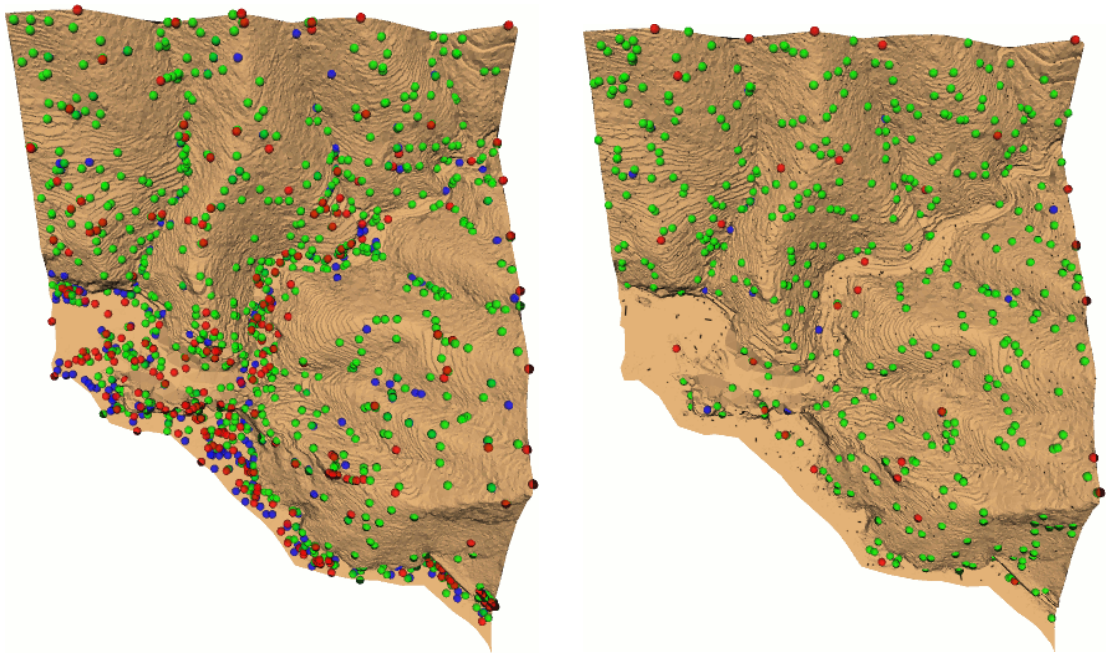

Figure 24 Two examples of feature extraction with two different choices of the threshold value for critical points.

The number of the detected critical points is usually very high and pruning or simplification steps are necessary to make the resulting structures understandable. For this reason, a further service is planned to filter out the irrelevant critical points up to a threshold, expressed in term of "persistence".

The algorithm is outlined by the following pseudo code:

```
Foreach vertex v in triangulation

      Star = StarofaVertex(v);

      Foreach edge e in Star

         If (abs(height(e->v[0] - height(v)) < -threshold && (abs(e-
>v[1] - height(v)) > threshold) up++
```

```
        If (abs(height(e->v[0] - height(v)) > -threshold && (abs(e-
>v[1] - height(v)) < threshold) down++

    Endfor

    If (up == 0)  v is a Pits

    If (down == 0)  v is a Peaks

    If (down >= 2) && (up >=2)  v is a saddle

Endfor
```

The service currently runs on triangulations, but will be extended to run on grids. The Input is a Triangle mesh (in PLY format) or a Gridded point cloud (geoTIFF). An optional parameter is the threshold for points to be considered at the same height. The Output is a List of feature points (in shape file format). The service is implemented in C++ and does not have dependencies on any other libraries. It is compiled under Linux and it is planned to work on Windows and Linux operative systems. Further properties are:

- Visualization: the input model, with a colour map of the height, and visualization (possibly in colour) of the features in the shape file.
- Accuracy: features are detected at the same level of the machine precision used to store the height function values.
- Robustness: the robustness is measured by the threshold used, that is, small perturbations of the input within the threshold do not change the result
- Scalability: The method is local and therefore the characterization can run with domain partitioning (on a grid trivial, on triangle meshes more complex); therefore, it is easily scalable.

4.2.8 Filter Point Cloud by Attribute & Coordinate

This service filters an unordered point cloud by the attributes attached to each point. The LAS files contain many attributes (including pulse number and classification). The point cloud can be filtered according to any of the attributes (including by coordinate value and point classification). The Input is an unordered point cloud in LAS format. The input parameters are the point attribute name and the point attribute boundary conditions to be selected. The output is a new unordered point cloud in LAS format in which only a subset of points, which match the filtering conditions, are maintained.

This service is implemented using the MapReduce framework in Java and can therefore scale with the Hadoop cluster. The LAS file I/O functionality is implemented as a sharing package with other Task 4.3 services that use LAS files for point cloud storage. In addition a single CPU implementation depending on libLAS is also done using C++. The main loop for this service iterates through all point records in the LAS file. Only points meeting the attributes specified by the user will be written to the output LAS file (or multiple layers of different LAS files as for the Hadoop implementation). The complexity is linear in the size of the input cloud.

A brief description of the service implementation:

```
## Mapper: (Key1, Value1) -> (Key2, Value2)
```

```
Input: k1 # Key1, the offset of LAS point record in the file, Bytes
Input: v1 # Value1, the input LAS point record
Input: attribute # the user specified point attribute
Input: attribute_bound # the attribute boundary

# Examine point attribute
switch (attribute):
      if ( v1.attribute not meets attribute_bound) break

k2 = (attribute, attribute_bound)
v2 = (v1.x, v1.y, v1.z)

Output: k2 # Key2, the tile coordinates
Output: v2 # Value2, point record

## Partitioner: (Key2, Value2) -> Reducer id
Input: k2
Output: k2 # each reducer writes points with one attribute

## Reducer: (Key2, Value2s) -> Output
Input: k2
Input: v2s # All the values with the same k2

# Loop through all the value with the same key
for v2 in v2s:
      # Send point to the output file
      print(v2)

Output: Filtered LAS files
```

The execution of the code is wrapped with a Bash script to provide Hadoop job setup and after-job data retrieval in addition to the main code functionality. Figure 25 demonstrates a filtering example of the London LiDAR data set using point classification. The original point cloud is coloured in grey on the top left, and layers of different point classification are coloured differently.

The performance of the current service implementation depends on the specific Hadoop cluster deployment and user inputs. For example, the execution time for a single-node Hadoop run on the London LiDAR data set using point classification as shown in Figure 25 is about 30 seconds.

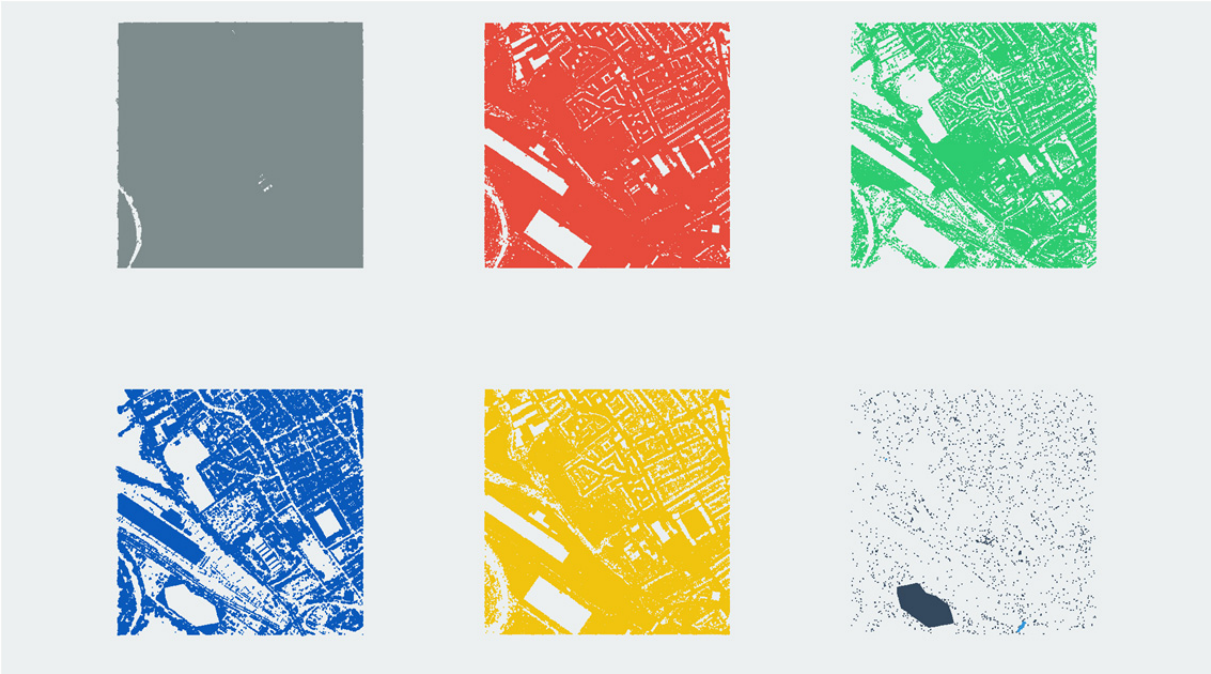

Figure 25: Example output of attribute filtering. The classification label stored in the LAS file is used for selection. Labels include ground, low vegetation, medium vegetation and high vegetation.

5 REFERENCES

Apache Mahout: Scalable machine learning and data mining [WWW Document], 2013. URL http://mahout.apache.org/ (accessed 10.28.13).

Banchoff, T.F., 1970. Critical points and curvature for embedded polyhedral surfaces. Am Math Mon. 77, 475–485.

Breiman, L., 2001. Random forests. Mach. Learn. 45, 5–32.

Broscoe, A.J., 1959. Quantitative analysis of longitudinal stream profiles of small watersheds. DTIC Document.

Butler, H., Loskot, M., Vachon, P., Vales, M., Warmerdam, F., 2013. libLAS: ASPRS LAS LiDAR Data Toolkit [WWW Document]. URL http://www.liblas.org/ (accessed 10.28.13).

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 107–113.

Demantké, J., Mallet, C., David, N., Vallet, B., 2011. Dimensionality based scale selection in 3d lidar point clouds. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. Laser Scanning.

Google C++ Style Guide [WWW Document], 2013. URL http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml (accessed 10.28.13).

lidarformat - C++ point cloud library [WWW Document], 2013. URL http://code.google.com/p/lidarformat/ (accessed 10.28.13).

Lin, J., Kolcz, A., 2012. Large-scale machine learning at twitter, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, Scottsdale, Arizona, USA, pp. 793–804.

Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M., 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proc VLDB Endow 5, 716–727.

MATLAB - The Language of Technical Computing [WWW Document], 2013. URL http://www.mathworks.co.uk/products/matlab/ (accessed 10.28.13).

Mount, D.M., Arya, S., 1998. ANN: Library for Approximate Nearest Neighbour Searching.

Muja, M., Lowe, D.G., 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. Presented at the VISAPP (1), pp. 331–340.

Preparata, F.P., Shamos, M.I., 1985. Geometric Searching, in: Computational Geometry. Springer.

Tarboton, D.G., 2005. Terrain analysis using digital elevation models (TauDEM). Utah Water.

Tesfa, T.K., Tarboton, D.G., Watson, D.W., Schreuders, K.A., Baker, M.E., Wallace, R.M., 2011. Extraction of hydrological proximity measures from DEMs using parallel processing. Environ. Model. Softw.

The Standard : Standard C++ [WWW Document], 2013. URL http://isocpp.org/std/the-standard (accessed 10.28.13).

6 APPENDIX – SERVICE INFORMATION TABLES

IQmulus Service information #7			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Static tiling of LAS file		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Break up a large LAS file into tiles		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Point cloud, format: LAS	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Tile Size; Tile Overlapping Percentage;	
	Output: <data representation and format>	Multiple point clouds, format: LAS	
	Functionality of the service: <text>	Reduce the size of individual LAS files	
Algorithm	Map inputs: point record from LAS file Map outputs/ Reduce inputs: Key – tile id, Value – point record Reduce outputs: multiple xyz files Final outputs: translating xyz into LAS		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	JAVA, C++, Bash	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc.); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the output.*
	Dependencies with other libraries	Hadoop, libLAS	
	Operating system	Unix/Linux	

	Visualization modalities of the output	pcl_viewer, meshlab	
IQmulus Data	**Available IQmulus input data**: <ID of the dataset as in the eRoom tables> 23, Liguria		*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** N/A **Robustness:** Robust **Computational time in relation to data size:** O(n)**,** Linear **Locality/globality of the algorithm:** Native Hadoop implementation under MapReduce framework.		*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
Alternatives	<List of Service IDs>		*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	<Use cases ID as in RedMine>		*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	UCL		*Partner ID and responsible person (include email)*
Involved Partners			

IQmulus Service information #16			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Quick visualization		*Quick display and rendering of the 3D input data.*
Description	The 3D data is read from the folder path which is provided by the user. Data is automatically displayed on the screen and 3D rendering, zooming like mouse actions are taken for visualization of the different views and the details.		*User can have a quick overview of the data to be processed.*
Service functionality	**Input:**	.pcd point cloud file format	*Quick display of the 3D input data. Very large point cloud data can be displayed after down sampling in order to increase the speed of zooming and rendering.*
	Input parameters: *[optional]*	Path of the folder where the input 3D data is located.	
	Output: <data representation and format>	Screen display	
	Functionality of the service: <text>	Quick visualization of the 3D data.	
Algorithm	Reading point cloud data using Point Cloud Library (PCL) functions. Creating 3D display screen by using Visualization Kit (VTK) library and adding data rendering features.		*PCL and OpenGL also provides functions for 3D display.*
Implementation details	**Implementation language**	C++	*It is also possible to display data by using commercial software like QuickTerrain, MeshLab, CloudCompare, etc.*
	Dependencies with other libraries	PCL, VTK, Boost Libraries are needed.	
	Operating system	Windows	
	Visualization modalities of the output	Visualization and rendering of the point cloud. Rendering of triangle meshes option can be provided.	
IQmulus Data	**Available IQmulus input data**: For tests we have used the following data;		*eRoom id's are provided.*

	(1) UBO – TLS Topography of a small beach in Brittany (ID-10-11) (2) UBO – Bathymetry in Brittany (ID-12-13) (3) Linguria – Lidar data for the study area of Vernazza, Liguria, Italy (ID-22-23)	
Service characteristics	**Accuracy:** -	
	Robustness: reliable to work with large point clouds	
	Computational time in relation to data size: == less than two seconds even for point cloud data with ~600000 points.	
	Locality/globality of the algorithm: the program can be used to see the input data or the processed data quickly.	
Alternatives	Using 3D display functions from other libraries like PCL and OpenGL at the code generation. Another option is using commercial software packages for point cloud display; i.e. QuickTerrain, MeshLab, CloudCompare.	
Related use cases	Each user story in the RedMine list can benefit from quick 3D visualization application for displaying the input or the output data.	
Responsible Partner	UCL	
Involved Partners	TUDelft, Beril Sirmacek [b.sirmacek@tudelft.nl]	

IQmulus Service information #2			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Point Cloud to Regular Grid		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	From an unordered point cloud develop a grid representation in 2.5 D		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Unordered point cloud, format: LAS	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Grid size (resolution)	
	Output: <data representation and format>	Gridded point cloud, format: LAS; 2.5D raster, format: GeoTIFF	
	Functionality of the service: <text>		
Algorithm	Map inputs: LAS point records Map outputs/Reduce inputs: Key – Grid id; Value: Point records; Reduce outputs: multiple xyz files Final outputs: translating xyz into LAS, and GeoTIFF		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	JAVA, C++, Bash	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the output.*
	Dependencies with other libraries	Hadoop, libLAS	
	Operating system	Unix/Linux	
	Visualization modalities of the output	pcl_viewer meshlab	

IQmulus Data	**Available IQmulus input data**: <ID of the dataset as in the eRoom tables> 23, Liguria	*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** N/A **Robustness:** Robust **Computational time in relation to data size:** O(n), Linear **Locality/globality of the algorithm:** Native Hadoop implementation under MapReduce framework.	*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
Alternatives	<List of Service IDs>	*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	<Use cases ID as in RedMine>	*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	UCL	*Partner ID and responsible person (include email)*
Involved Partners		

IQmulus Service information #3			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Peak points		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Detect peak point (highest point) in an unordered point cloud, with respect to a given region size.		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Unordered point cloud in LAS file format	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Radius of region, sensitivity	
	Output: <data representation and format>	Sparse point cloud (i.e. list of) peak points as ASCII or shape file	
	Functionality of the service: <text>	Detect highest point within a set search radius.	
Algorithm	The algorithm loops over all points of the input cloud (O(n)). For each point the k-neighbourhood is inspected within the given radius (O(log m)). If the point is the highest point it is stored, otherwise discarded.		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	C++	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the output.*
	Dependencies with other libraries	PCL 1.6, Shapelib, libLAS	
	Operating system	Microsoft Windows	
	Visualization modalities of the output	Sparse feature points together with dense point cloud	

IQmulus Data	**Available IQmulus input data**: Liguria	*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** Only points from the original point cloud will be returned, no interpolation is performed.	*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
	Robustness: The current implementation is not robust against outliers. Outliers need to be removed prior to this process. See Process "outlier removal".	
	Computational time in relation to data size: O(n log m) with n the size of the input cloud and m the size of the region (# of points)	
	Locality/globality of the algorithm: The algorithm works locally on a subset of the point cloud there are no global interdependencies**.**	
	Currently the service is available only as a single CPU implementation. External tiling is required to enable parallelism.	
Alternatives	#44 Critical Points	*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	User story (IQmulus) #1072	*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	UCL Jan Boehm (j.boehm@ucl.ac.uk)	*Partner ID and responsible person (include email)*
Involved Partners		

<table>
<tr><th colspan="4">IQmulus Service information #10</th></tr>
<tr><th>Name of the metadata</th><th colspan="2">Content expected</th><th>Motivation/comments</th></tr>
<tr><td>Service Acronym</td><td colspan="2">Outlier Detection</td><td>Unique identifier of the service; necessary to call it within user-defined workflows</td></tr>
<tr><td>Description</td><td colspan="2">Statistical outlier detection of points in laser scan data, useful in removing erroneous data caused by a variety of effect inherent in the capture process</td><td>Brief textual description of the service: what it provides, what can be used for. This text could be used as a short “help text” in the User Interfaces of IQmulus</td></tr>
<tr><td rowspan="4">Service functionality</td><td>Input: <data representation and format></td><td>pcl::PointCloud<pcl::PointXYZ>

a point cloud in pcl point cloud format

Will be changed to LAS</td><td rowspan="4">Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.</td></tr>
<tr><td>Input parameters:
[optional]</td><td>The threshold of how many standard deviations away from the global mean neighbourhood distance is acceptable to qualify as inliers.

The k neighbours to sample in order to generate the statistical profile of the input data</td></tr>
<tr><td>Output: <data representation and format></td><td>Two pcl::PointCloud<...> (as above) relating to the ‘outliers’ and the ‘inliers’. Will be changed to LAS</td></tr>
<tr><td>Functionality of the service: <text></td><td>This separates the input point cloud into inliers and outliers</td></tr>
<tr><td>Algorithm</td><td colspan="2">Input k # nearest neighbourhood size
Input sd_tol # tolerance threshold
Input cloud # input point could

calculate the mean distance between points in all neighbourhoods
For all $\mathbf{p}_i$ in cloud
$\mathbf{u}_i$ = mean_dist($\mathbf{p}_i$, nearest_neighbours(k , $\mathbf{p}_i$))

# calculate global statistics for neighbourhoods in this cloud
U = mean(u)
SD = standard_deviation(u)

Threshold = U + (SD * sd_tol)</td><td>The same functionality may in principle be implemented by different algorithms.</td></tr>
</table>

<table>
<tr><td></td><td colspan="2"># divide the cloud into inliers and outliers
For all u_i in u
If u_i <= Threshold
inliers.push_back(p_i)
else
outliers.push_back(p_i)

Output inliers # points who's neighbourhood are within tolerance
Output outliers # points who's neighbourhood are outside tolerance</td><td></td></tr>
<tr><td rowspan="4">Implementation details</td><td>Implementation language</td><td>C++</td><td rowspan="4">Include information related to the implementation of the service, such as language (e.g., C, C++, etc); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the output.</td></tr>
<tr><td>Dependencies with other libraries</td><td>Core algorithm
Pcl + Requires
Boost > 1.46
Flann > 1.71
Eigen > 3
Vtk > 5.6

Debug/development viewer
Cinder + Requires
OpenGL
Boost</td></tr>
<tr><td>Operating system</td><td>Core algorithm
Build & runs on Linux and Windows

Debug/development viewer
Build and runs on windows, however Cinder not yet supported for linux.</td></tr>
<tr><td>Visualization modalities of the output</td><td>A rudimentary debugging/development GUI visualizer to eyeball the data input and generated</td></tr>
<tr><td>IQmulus Data</td><td colspan="2">Available IQmulus input data:
Can read PLY and LAS files drag-n-drop into the visualisation window to load (may take a few moments as outlier detection is performed on load).</td><td>If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]</td></tr>
<tr><td>Service characteristics</td><td colspan="2">Accuracy: variable depending on parameters and innate characteristics of the input cloud</td><td>These fields are necessary to document all the characteristics of the services that are important to assess</td></tr>
</table>

	Robustness: variable depending on parameters and innate characteristics of the input cloud	*the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
	Computational time in relation to data size: approximately scales at **(n log n)** based on the time taken for construction of the kdtree and nearest neighbour queries.	
	Locality/globality of the algorithm: since the algorithm computes the global mean distance for the input data this could result in varying identification of outliers across different input data sets. To overcome this problem either the mean and standard deviation (SD) could be calculated as sample mean and SD. Alternatively a synchronisation step could gather all the tile local mean and variance and combine them to create a global threshold which is then used for all tiles. Resulting in: 1) Tile data 2) Compute mean + SD of neighbourhoods on each tile 3) Gather + combine to get the global mean + SD 4) Compute global tolerance threshold 5) For each tile separate outliers using global threshold	
	For a 4 million point data set the execution time is in the range of 5-20 seconds depending on machine capabilities. The current implementation runs on a single node single thread implementation and could potentially process larger data sets in shorter times through parallelisation or multimode configurations. This would be reliant on a successful tiling of the input data into contagious tiles and on the synchronisation step mentioned above.	
Alternatives	nanoflann (https://code.google.com/p/nanoflann/) may be an alternative to using the pcl kdtree construction and approximate nearest neighbour (ANN) search interfaces. This would reduce the dependencies (nanoflann has only STL dependencies), further it could possibly offer up to 2x speed up on search and construction of the kdtree.	*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	Data clean up pre-processing step for use as part of other algorithms.	
Responsible Partner	UCL	*Partner ID and responsible person (include email)*
Involved Partners		Other Partners and responsible persons (if any, emails) involved in the development and/or testing of the service

<table>
<tr><th colspan="4">IQmulus Service information #17</th></tr>
<tr><th>Name of the metadata</th><th colspan="2">Content expected</th><th>Motivation/comments</th></tr>
<tr><td>Service Acronym</td><td colspan="2">3D Local keypoint extraction from point clouds (low-level feature)</td><td>Extracting low-level features from 3D data which does not give valuable information to the end user however can be used at recognition, detection, registration, modelling, data compression, etc. modules of the IQmulus system.</td></tr>
<tr><td>Description</td><td colspan="2">Features are distinctive and information containing locations of an 2D or 3D input data. They can be either points given by certain coordinates, or small regions. In our case we consider points represented by their coordinates and we call them keypoint. Low-level term is used to indicate that the extracted keypoints might be meaningless for the end user, however it contains crucial for other modules of the IQmulus system for further process and to generate further end results.</td><td></td></tr>
<tr><td rowspan="4">Service functionality</td><td>Input:</td><td>.las point cloud file format</td><td rowspan="4">Quick display of the 3D input data with keypoint locations.</td></tr>
<tr><td>Input parameters: [optional]</td><td>Path of the folder where the input 3D data is located.</td></tr>
<tr><td>Output: <data representation and format></td><td>.mat matrix file which contains x, y, z coordinates of the extracted keypoints.</td></tr>
<tr><td>Functionality of the service: <text></td><td>Extracting low-level features of the input point cloud file.</td></tr>
<tr><td>Algorithm</td><td colspan="2">Reading the point cloud data using LibLas functions.

Creating 3D display screen by using Visualization Kit (VTK) library and adding data rendering features.

Extracting low-level features the point cloud using PCL and other libraries which are going to be explored by experiments.

Visualization of the extracted keypoints on the original point cloud.</td><td></td></tr>
</table>

<table>
<tr><td></td><td colspan="2">Saving the keypoint locations into a .mat file.</td><td></td></tr>
<tr><td rowspan="4">Implementation details</td><td>Implementation language</td><td>C++</td><td rowspan="4">It is also possible to display data and the extracted keypoint locations by using commercial software like QuickTerrain, MeshLab, CloudCompare, MatLab etc.</td></tr>
<tr><td>Dependencies with other libraries</td><td>PCL, VTK, Boost Libraries are needed. (More libraries might be added.)</td></tr>
<tr><td>Operating system</td><td>Windows</td></tr>
<tr><td>Visualization modalities of the output</td><td>Visualization and rendering of the point cloud with the extracted keypoints.</td></tr>
<tr><td>IQmulus Data</td><td colspan="2">Available IQmulus input data:

For tests we have used internal data set.

We continue to the experiments by using the following data sets;

(4) UBO – TLS Topography of a small beach in Brittany (ID-10-11)
(5) UBO – Bathymetry in Brittany (ID-12-13)
(6) Linguria – Lidar data for the study area of Vernazza, Liguria, Italy (ID-22-23)</td><td>eRoom id’s are provided.</td></tr>
<tr><td rowspan="4">Service characteristics</td><td colspan="2">Accuracy: Depends on the correct scale selection. Automatic scale selection will be developed in the future.</td><td rowspan="4"></td></tr>
<tr><td colspan="2">Robustness: reliable to work with large point clouds</td></tr>
<tr><td colspan="2">Computational time in relation to data size: ~5 minutes for the point cloud data with ~100000 points.</td></tr>
<tr><td colspan="2">Locality/globality of the algorithm: the program can be used with object detection, reconstruction, classification, registration, etc. modules of the IQmulus project.</td></tr>
<tr><td>Alternatives</td><td colspan="2">Another option is using commercial software packages for point cloud display and keypoint extraction; i.e. QuickTerrain, Terrasolid.</td><td></td></tr>
<tr><td>Related use cases</td><td colspan="2">User Story (IQmulus) #1037, Land Showcase User Story 2 (1.2.2_SC2_2); User can register up-to-date scene on to the 3D models. The keypoints are needed for the registration process.</td><td></td></tr>
<tr><td>Responsible Partner</td><td colspan="2">TUDelft, Beril Sirmacek [b.sirmacek@tudelft.nl]</td><td></td></tr>
<tr><td>Involved Partners</td><td colspan="2">• TUDelft, Roderik Lindenbergh (algorithm development support)
• TUDelft, Jinhu Wang (testing the developed algorithms)</td><td></td></tr>
</table>

IQmulus Service information #27			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Tree crown recognition from mobile mapping point clouds		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Detecting tree crowns and separate them from planar surfaces in point cloud data collected from mobile mapping		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Unordered point clouds from LiDAR in LAS format	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Radius of influence for feature computation	
	Output: <data representation and format>	Labelled unordered point cloud	
	Functionality of the service: <text>	The service reads an unordered point cloud captured from mobile mapping and classifies the data into tree crown and planar surface classes.	
Algorithm	In the first stage the algorithm computes per point features based on PCA to estimate local point entropy. In the second stage these features are used to classify data into tree crown and planar surface classes. The classifier must be trained on training data prior to that.		*Service #66 "Point Cloud Dimensionnality (PCD)" can be used to compute features from year 2 on.*
Implementation details	**Implementation language**	C++ using propriety decision tree implementation	*In the next period a cloud-based classification framework will be used, e.g. Mahout.*
	Dependencies with other libraries	PCL for feature computation	
	Operating system	Microsoft	

		Windows	
	Visualization modalities of the output	Point cloud and scalar label	
IQmulus Data	**Available IQmulus input data**:		*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** Classification correctness (currently 99% and 96% in limited tests)		*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
	Robustness: Misclassification, false positives (currently 1% and 4% in limited tests)		
	Computational time in relation to data size: Feature computation is the bottle neck, but complexity is generally O(n) albeit with large constant.		
	Locality/globality of the algorithm: Per point classification is independent for all points, so maximum parallelism can be exploited. Feature computation depends on a region surrounding the point, which is controlled by the radius parameter. Again good parallelization can be achieved.		
	At the moment only a single CPU implementation is provided. Tiling is required to be performed externally.		
Alternatives	#27 "Tree recognition from point clouds" and #59 "Multi-object classification of 3D point clouds [object must be specified]"		*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	User story (IQmulus) #1097 "As a GIS expert (geodata provider/integrator) working at local level, I want to automatically detect individual trees from a LiDAR point cloud in a urban area, so that I can monitor growth and foresee pruning works."		*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	UCL - Jan Boehm j.boehm@ucl.ac.uk		
Involved Partners			

IQmulus Service information #42			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	watershedDEM		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Extract watershed and flow directions from digital elevation model. As an optional output, it is possible to get the accumulation area and the slope map.		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	DEM gridded point cloud (GeoTIFF)	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Threshold size of the basin. Degree of the river network. Shapefile representing the outlet point of the basin	
	Output: <data representation and format>	Shapefile, Gridded point cloud (GeoTIFF)	
	Functionality of the service: <text>	Extraction of watershed and flow directions	
Algorithm	This algorithm uses both D8 – Dinf flow directions for routing the flow to outlet. (D8 refers to flow direction calculated in the steepest direction and only cell centre to cell centre – Dinf refers to a more flexible routing that give a more real direction)		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	C++	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc.); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization*
	Dependencies with other libraries	Read and write libraries for GeoTIFF and shapefile, MPI, Taudem	
	Operating system	Linux-like	
	Visualization modalities of the	3D visualization of the DEM with watersheds and each different basin visualized with	

	output	different colours, possibly in relation to their size; visualization of the river net, possibly with different colours according to their degree	*modalities of the output.*
IQmulus Data	**Available IQmulus input data**: Dataset 22; Dataset 6;		*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** **Robustness:** **Computational time in relation to data size:** **Locality/globality of the algorithm:**		*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
Alternatives			*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	• User story (IQmulus) #1076		*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	• CNR-IMATI, Simone Pittaluga (simone.pittaluga@ge.imati.cnr.it)		*Partner ID and responsible person (include email)*
Involved Partners	• Regione Liguria		

IQmulus Service information #64			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	PersistentCriticalPoints		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Extraction of critical points (isolated and grouped in lines/regions) from grids and triangulations		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Triangle mesh (PLY) Gridded point clouds (geoTIFF)	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	The threshold, expressed in term of "persistence", which is used to filter out irrelevant critical points. If no threshold is given, a default value is chosen.	
	Output: <data representation and format>	List of features (Shapefile)	
	Functionality of the service: <text>	Persistent critical points	
Algorithm	The local characterization uses Banchoff's definition of critical points, extended to non-isolated critical points. Critical points within the z-value threshold are filtered out using persistence/significance evaluation.		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	C++	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc.); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the*
	Dependencies with other libraries	==	
	Operating system	Windows, Linux	
	Visualization modalities of the output	• Visualization of the input model, in case with a colour map of the height • Visualization	

		(possibly in colour) of the features in the shape file	*output.*
IQmulus Data	**Available IQmulus input data**: • 7 (FOMI) • 17, 22, 23 (RegioneLiguria) • 8, 9, 12, 13 (UBO)		*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** features are detected at the same level of the machine precision used to store the height function values, and within the threshold indicated. **Robustness:** depends on the method implemented for the persistence evaluation **Computational time in relation to data size:** The computational cost of the service for the surface characterization is O(n)+O(c log c), where n is the number of input vertices and c are the critical points (they are ordered for the persistence evaluation). **Locality/globality of the algorithm:** • the choice of a threshold makes the algorithm not local anymore		*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
Alternatives	• Service #3: Peak points • Service #44: CriticalPoints: computation of the critical points without persistence filtering. The following services have similar goal but with low level features: • Service #17: 3D Local keypoint extraction from point clouds (low-level feature) • Service #37: Low-level feature Point		*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases	• User story (IQmulus) #1035 • User story (IQmulus) #1074? • User story (IQmulus) #1086		*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	CNR-IMATI, Silvia Biasotti silvia@ge.imati.cnr.it, Andrea Cerri cerri@ge.imati.cnr.it		*Partner ID and responsible person (include email)*
Involved Partners	Possible tests on data sets provided by Regione Liguria, UBO, and FOMI. Possible comparison with the services provided by TuDelft and UCL.		

IQmulus Service information #54			
Name of the metadata	**Content expected**		**Motivation/comments**
Service Acronym	Filter Point Cloud by Attribute & Coordinate		*Unique identifier of the service; necessary to call it within user-defined workflows*
Description	Filter an unordered point cloud by the attributes attached to each point. LAS files contain various attributes (pulse number, classification). The point cloud can be filtered according to any of the attributes (including by coordinate value).		*Brief textual description of the service: what it provides, what can be used for. This text could be used as a short "help text" in the User Interfaces of IQmulus*
Service functionality	**Input: <data representation and format>**	Point cloud, format: LAS	*Include here the input/output of the service, the name of the functionality (e.g., registration, fusion), and input parameters (if any). In the future, we may consider defining a taxonomy of the functionalities to harmonize the terminology used to fill this field.*
	Input parameters: *[optional]*	Point Attribute Name; Point Attribute Boundary Conditions;	
	Output: <data representation and format>	Point cloud, format: LAS	
	Functionality of the service: <text>	Filtering points in a LAS file based on its point format attributes.	
Algorithm	Looping through the LAS file point record section, only points meeting the attributes specified by user will be write to the output LAS file.		*The same functionality may in principle be implemented by different algorithms.*
Implementation details	**Implementation language**	C++, BASH	*Include information related to the implementation of the service, such as language (e.g., C, C++, etc); dependencies with other libraries (e.g., ANN library); constraints on the operating systems, and visualization modalities of the output.*
	Dependencies with other libraries	libLAS	
	Operating system	Unix/Linux	
	Visualization modalities of the output	pcl_viewer meshlab	

IQmulus Data	**Available IQmulus input data**: <ID of the dataset as in the eRoom tables> 23, Liguria	*If there are examples of data that could serve as an example, then include here their identifiers as in the data table in eRoom (useful to test the service]*
Service characteristics	**Accuracy:** N/A **Robustness:** Robust **Computational time in relation to data size:** O(n), linear **Locality/globality of the algorithm:** Can be run as a Hadoop streaming utility.	*These fields are necessary to document all the characteristics of the services that are important to assess the quality of the results. Also, these fields could be used to select a specific service among more services that implement the same functionality but with different characteristics. See the following box.*
Alternatives		*List other services (if any) that have the same functionality, have the same input/output but use a different algorithm or have different features*
Related use cases		*Mention the user stories related to the usage of the service (see D4.1.1) and related use cases uploaded on Red Mine*
Responsible Partner	UCL	*Partner ID and responsible person (include email)*
Involved Partners	Other Partners and responsible persons (if any, emails) involved in the development and/or testing of the service	