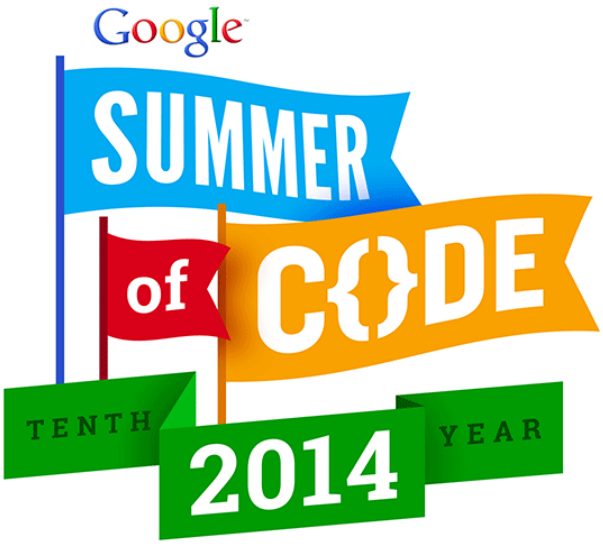

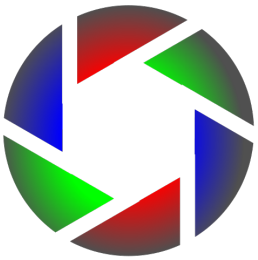

Opticks – LiDAR segmentation Plug-In based on RANSAC and PCA algorithms

Roberta Ravanelli

August 18, 2014

Contents

List of Figures

1 Introduction

1.1 Background

Opticks has recently added the point cloud data support to its software platform: in fact today more and more sensors (as LiDAR sensors) capable to provide high-quality 3D representations of the world - point clouds - become available, with a massively increased usage of 3-D data for perception tasks.

A point cloud is a data structure, a set of vertices, used to represent three-dimensional data. Oversimplifying, we can think about a point cloud as a collection of multiple points that provide computers the information (height, classification, intensity, color) necessary to recreate a physical object in a digital three dimensional coordinate system.

Now it is desirable that powerful processing tools and algorithms become available for handling and working with point clouds efficiently in Opticks. My project is precisely included in this background: the aim of this project is to wrap some of the PCL algorithms in an Opticks Plug-In, where PCL [1] is a software library which incorporates a multitude of 3D processing algorithms that operate on point cloud data, including: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration, etc.

[1] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). IEEE International Conference on Robotics and Automation (ICRA), May 2011.

1.2 Project Introduction

The aim of this project is to implement two of the Point Cloud Library (PCL) algorithms in an Opticks Plug-In, adding some point clouds processing capabilities to the Opticks software platform. In particular, tasks such as extracting interest features from dense accurate 3D point clouds acquired by Light Detection and Ranging (LiDAR) are not trivial and appropriate tools are required. LiDAR has become an important technology in topographic mapping, 3D city modelling and 3D building reconstruction. Buildings have a critical role for 3D city models, decision support systems, governments, telecommunication, disaster management. Thus, methods are needed to generate accurate and complete 3D building models with high degree of automation.

The original workflow of the project provided for the implementation, among all the PCL algorithms, of the RANdom SAmple Consensus (RANSAC) and Principal Component Analysis (PCA) algorithms, since they can be successfully applied to approach the problem of building roof segmentation as reported in [2], [3]. The first one is a robust method for model estimation widely used in the extraction of geometric primitives and 3D model reconstruction and it can be also used for roof facets extraction or segmentation. The second one can be adopted to determine the location of roof breaklines, which is an important element in building extraction and segmentation, because it implicitly follows the process of human perception.

During the GSoC, however, it has been decided not to apply the RANSAC algorithm directly to all the LiDAR point cloud, but only after having identified the zones in which the buildings are, in order to improve the quality of the results. In this way the RANSAC algorithm can be applied to the points included in each building in a recursive manner (for each building the outliers of the previous step are the input data for the next step and so on).

So the watershed segmentation algorithm (link) was selected to find the points which belong to the buildings and the connected components approach (link) was adopted to classify each identified building. Nevertheless, to use these two image processing techniques, the raw LiDAR data must be first converted in a raster file by interpolating it over a grid, generating a DEM. The watershed segmentation algorithm and the connected components approach have been implemented exploiting the OpenCV library.

However, after having implemeted this approach, there wasn't no more time to implement the PCA algorithm.

[2] J. Yan, W. Jiang and J, Shan. Quality analysis on ransac-based roof facets extraction from airborne lidar data.

International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXIX-B3, 2012 XXII ISPRS Congress.

[3] A. Sampath and J. Shan. Building roof segmentation and reconstruction from lidar point clouds using clustering techniques.

2 Plug-In documentation

2.1 LiDAR roof extraction Plug-In

This Plug-In implements a RANSAC-based technique for extracting roof planes of buildings from LiDAR point clouds. It consists of three different stages: raw LiDAR data are first interpolated over a grid with the nearest neighbor interpolation method, in order to generate a DEM raster; then the watershed segmentation algorithm and the connected components approach, which rispectively find and classify the DEM pixels which belong to the buildings, are applied (the DEM must be first divided into a rectangular grid of n $\times$ m tiles in order to improve the results of the segmentation process); finally, the extraction of roof planes is obtained by applying the RANSAC algorithm in a recursive manner.

2.1.1 Free dataset used for this tutorial

In order to test this Plug-In, a free LiDAR point cloud can be downloaded at the following link:

- San Diego sample point cloud (GSoC2014_test_point_cloud.las) link
- San Diego sample point cloud .kmz file, useful to have a reference (see Fig. 1) link

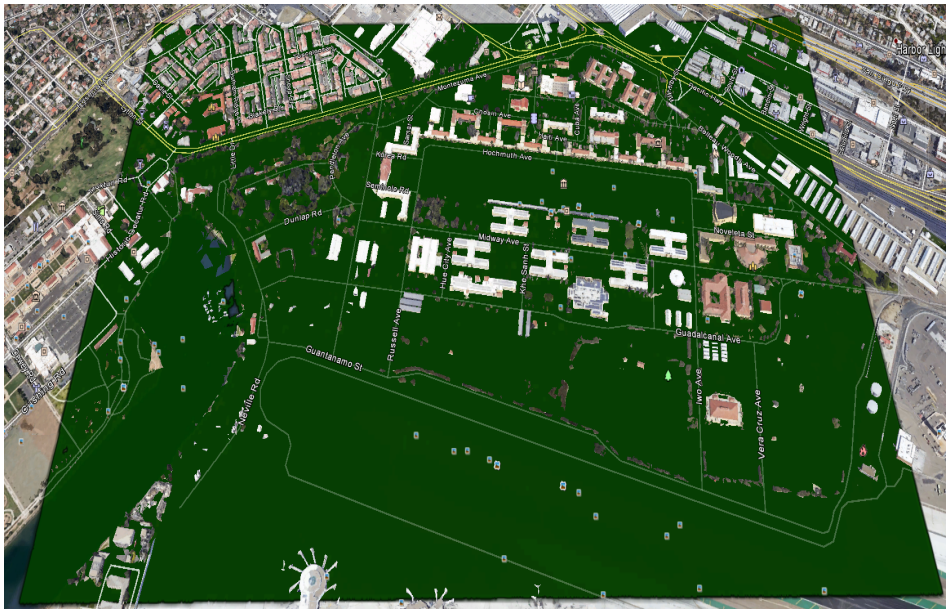

Figure 1: Site of the San Diego sample point cloud

It is also possible to use the LiDAR free sample datasets available through the Open Topography Portal (website), where several LiDAR .las files, recollected in different parts of the world, can be downloaded.

2.1.2 Tool workflow

The user can load the point cloud to be analized using the standard *Import data* method available from Opticks main menu. After the data are loaded and displayed in the main Opticks window, it is possible to access the *LiDAR Roof Extraction Plug-In* through the Point Cloud menu (i.e. Point Cloud->LiDAR Roof Extraction Plug-In). Fig. 2 shows how to launch the *Roof Extraction Plug-In* for the San Diego sample point cloud.

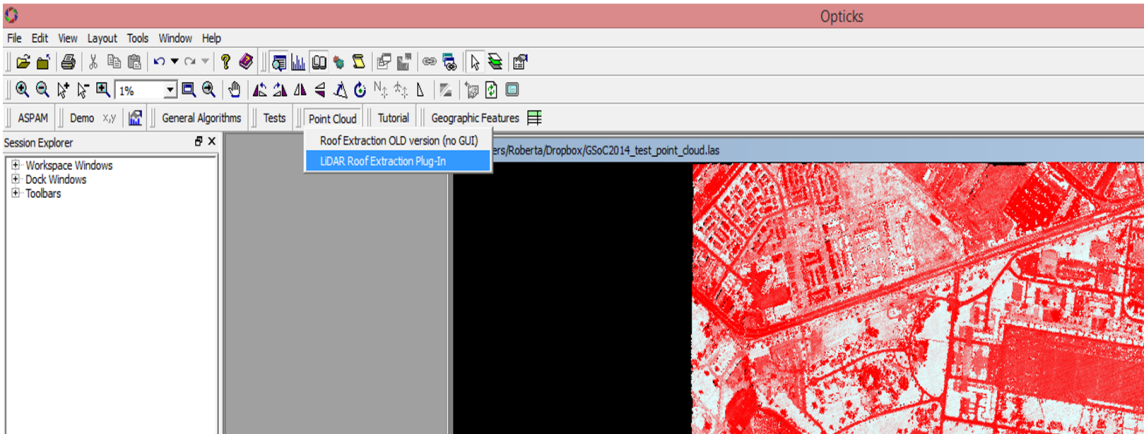

Figure 2: Launch the LiDAR Roof Extraction Plug-In through Point Cloud menu

The *LiDAR Roof Extraction Plug-In* Graphical User Interface (GUI) is displayed in Fig. 3.
The point cloud to be processed can be selected using the sliding tag *Select the input .las file*. This tag contains all the point clouds previously loaded into Opticks Workspace.

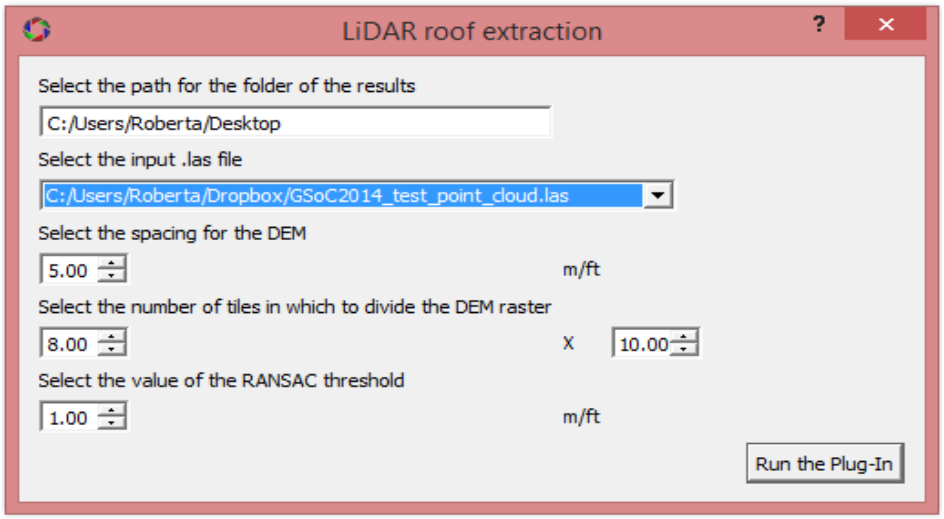

Figure 3: Graphical User Interface of the LiDAR Roof Extraction Plug-In

In addition it is necessary to underline that the user must create a *Results* folder (it will collect all the text and image files generated by the Plug-In) and an additional *Tiles* folder (it will contain the DEM tiles needed to the segmentation process) inside it. In fact the path for the *Results* folder has to be inserted inside *Select the path for the folder of the results* tag of the GUI. Fig. 4 shows how to select the path for the *Results* folder (the line highlighted in blue is the string that must be inserted in the path tag; if the *Results* folder is in the user's desktop, he/she must only change the name Roberta with his/her user name).

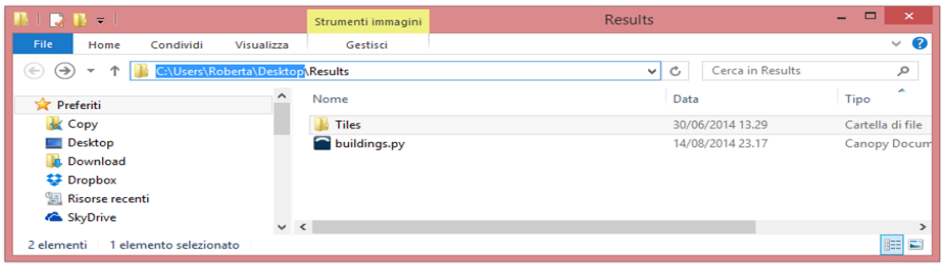

Figure 4: Retrieve the path for the *Results* folder: the blue string

At this point, before starting the processing, the user must set all the other input parameters in their respective sliding tags:

- the DEM spacing, the pixel spacing of the DEM matrix, in meters or feet depending on the reference system used in the .las file;
- the number of horizontal (first sliding tag) and vertical tiles (second sliding tag) in which the DEM raster is divided in order to process it with the watershed algorithm;
- the value of the RANSAC threshold.

The GUI parameters are initialized with default values, optimal for the San Diego sample point cloud. Once all input parameters have been selected, *LiDAR Roof Extraction Plug-In* can be run by pressing the *Run the Plug-In* button.

The Plug-In generates several text and image files:

- *Building_i.txt*: the file contains the image coordinates and 3D coordinates of the points that belong to the i^{th} building;
- *buildings_inliers.txt*: each row contains the ID of the inliers that belong to the first plane detected by RANSAC (iteration number 0) for a single identified building (e.g. row 0 contains the ID of the inliers for the first plane detected of the building number 0, row 1 contains the ID of the inliers for the first plane detected of building number 1 and so on);
- *buildings_parameters.txt*: each row contains the four plane parameters of the first detected plane by RANSAC (iteration number 0) for a single identified building (e.g. row 0 contains the parameters of the first detected plane of the building number 0, row 1 contains the parameters of the first detected plane of the building number 1 and so on);
- *Inliers_building_i.txt*: the file contains the ID of inliers of the i^{th} building for every RANSAC iteration (e.g. row 0 contains the inliers for RANSAC iteration number 1, row 1 contains the inliers for RANSAC iteration number 2 and so on);
- *plane_parameters_building_i.txt*: the file contains the plane parameters of the i^{th} building for every RANSAC iteration (e.g. row 0 contains the plane parameters of the second detected plane - RANSAC iteration number 1 -, row 1 contains the plane parameters of the third detected plane - RANSAC iteration number 2 - and so on);
- *Number_of_RANSAC_applications.txt*: each row contains the number of times that the RANSAC algorithm has been applied for a single building (e.g. row 0 contains the number of RANSAC applications for the building number 0 and so on);
- *Ransac_buildings_results.txt*: the file contains the ID of the inliers and the plane parameters for the first detected plane of all the identified buildings;
- *Ransac_log.txt*: the file is the RANSAC log file;

- *result_watershed.png*: this image shows the results of the watershed algorithm, the buildings should be easily identifiable (see Fig. 5);

Figure 5: Watershed algorithm results for the San Diego sample point cloud

- *result_pca.png*: this image shows the results of the PCA algorithm (see Fig. 6), implemented through the OpenCV library; in fact, during the GSoC, there hasn't been enough time to implement the PCL version of the algorithm. However the results of the PCA aren't used for extracting the roof planes;

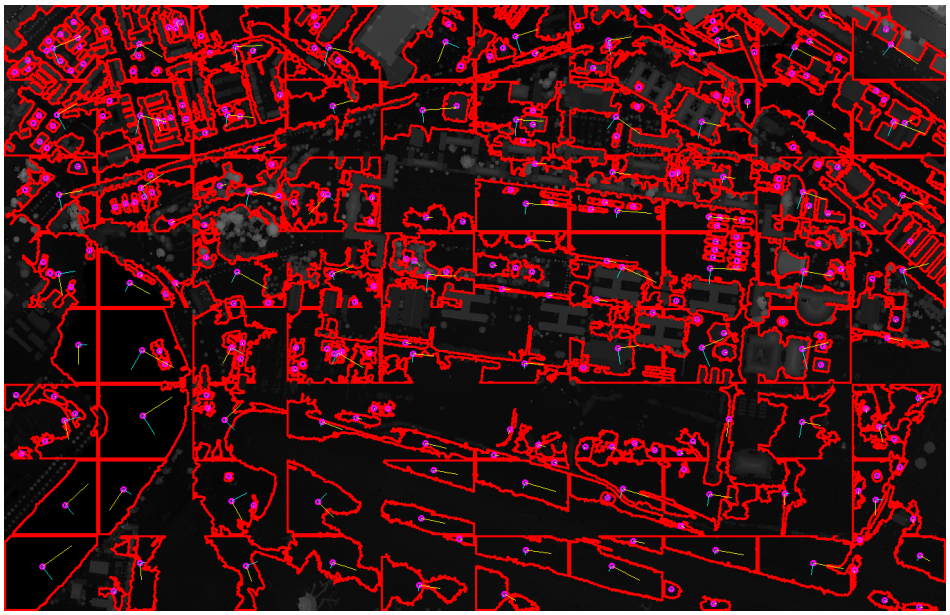

Figure 6: PCA algorithm results for the San Diego sample point cloud

- *building_contours.png*: this image shows the countours of the identified buildings (see Fig. 7); it isn't used for

extracting the roof planes;

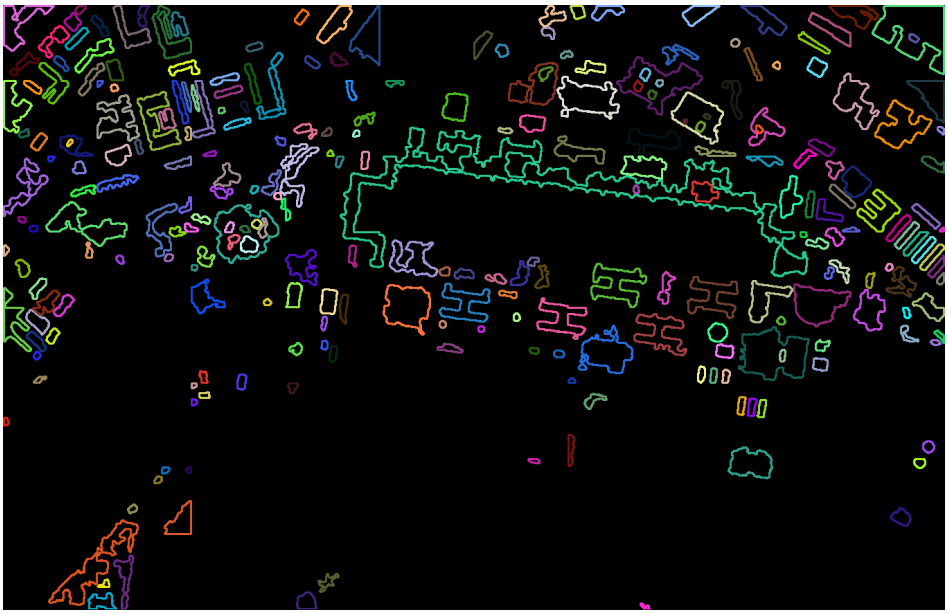

Figure 7: Countours of the identified buildings for the San Diego sample point cloud

- *result_con_comp.png*: this image shows the results of the connected components method, which assigns a different label to every identified building (every building in the image has a different colour in function of its label, see Fig. 8);

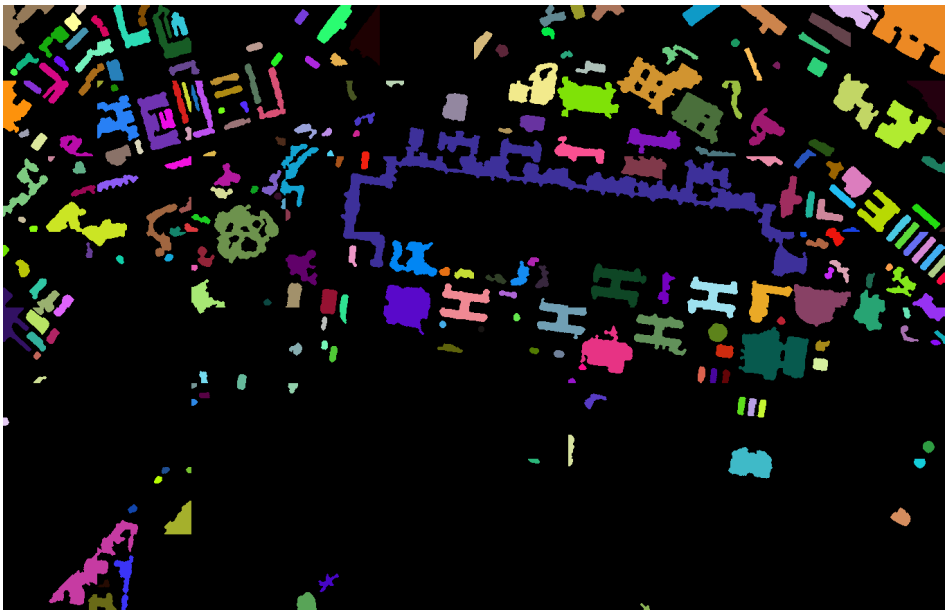

Figure 8: Connected components results for the San Diego sample point cloud

- *building_roofs.png*: this image shows the final results of the *LiDAR Roof Extraction Plug-In*: it will pop up at the end of the processing stage and it must be closed to make the Plug-In finish. For each building the points

that belong to the first four detected planes are represented with different colours (see Fig. 9): the points which belong to the first plane are yellow, those belonging to the second plane are blue, those belonging to the third plane are green and those belonging to the fourth detected plane are brown.

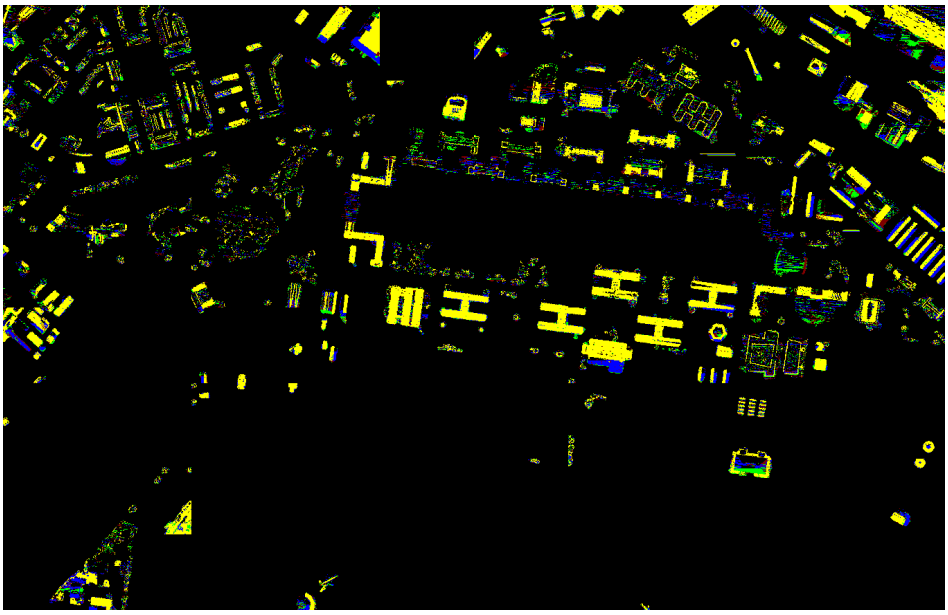

Figure 9: LiDAR Roof Extraction Plug-In results for the San Diego sample point cloud

In the code repository there is also a python script (*buildings.py*) that allows to visualize the Plug-In results in 3D for every identified buildings; to use it the user must copy the script in the *Results* folder.

Fig. 10 shows the results of the segmentation process inside the Opticks environment, while Fig. 11 shows the DEM generated by the Plug-In for the San Diego sample point cloud.

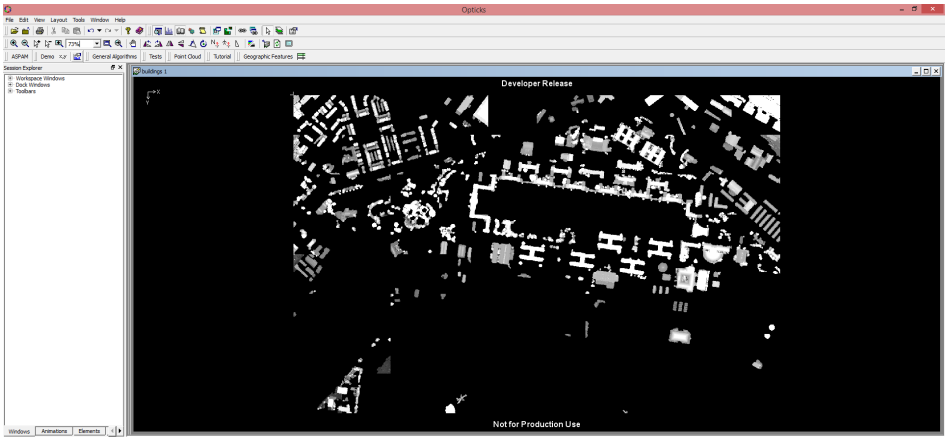

Figure 10: Visualization of the segmentation results inside the Opticks environment

Fig. 12, Fig. 13, Fig. 14 show into details the results for some identified buildings: we can see that *LiDAR Roof Extraction Plug-In* works well on some buildings (for example buildings 128 and 230), but it doesn't work on others (for example building 171). So further developments are needed to improve the results, for example to use different

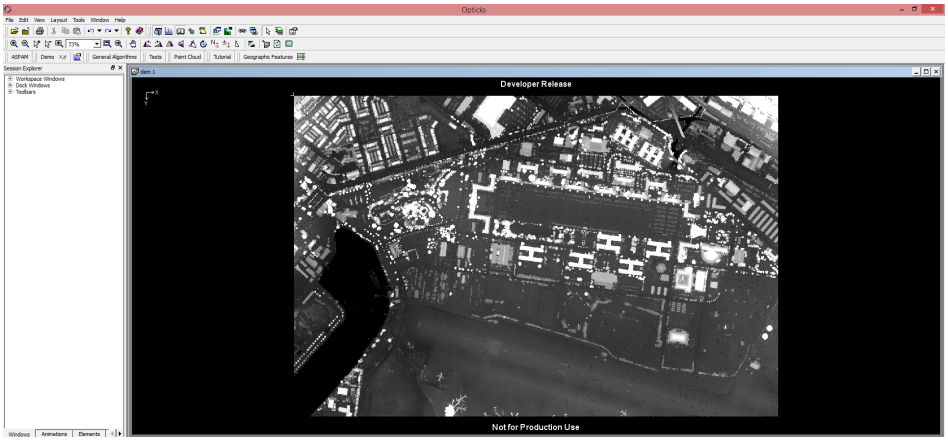

Figure 11: Dem of the San Diego sample point cloud generated by the LiDAR Roof Extraction Plug-In

RANSAC thresholds for every building, to improve the segmentation results (trees and also airplanes are classified as buildings). Moreover, the Plug-In works enough well on this sample point cloud, but it doesn't work so well on others: a generalization of the threshold selection in the watershed algorithm must be also done. Then a re-organization of the results (especially for the text files) is also needed.

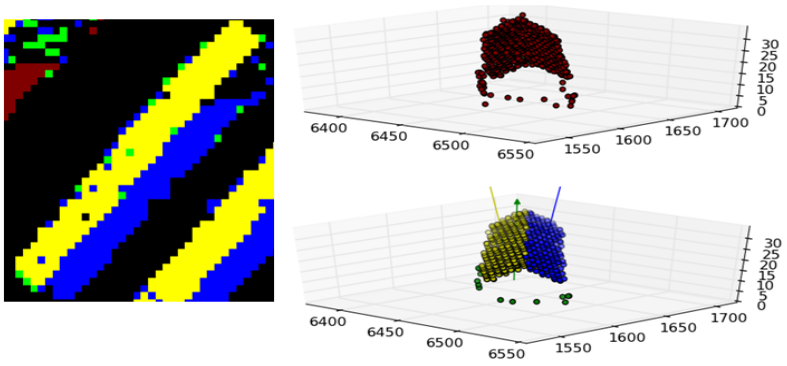

Figure 12: LiDAR Roof Extraction Plug-In results for the building 128

2.2 Code architecture

The Plug-In code is based on five main classes: *Interpolation*, *Segmentation*, *Ransac*, *Gui* and *LiDAR_roof_segmentation*.

The *Interpolation* class is used to generate the DEM from the original .las file; for now it supports only the nearest neighbor interpolation method, but in future other interpolation methods could be added.

The *Segmentation* class implements all the segmentation methods described earlier (watershed segmentation, PCA segmentation, connected components and RANSAC segmentation) is heavily based on the OpenCV library and also on the *Ransac* class.

The *Ransac* class implements the RANSAC algorithm; it is heavily based on the PCL library implementation, but there are some additions in order to use the algorithm in a recursive manner.

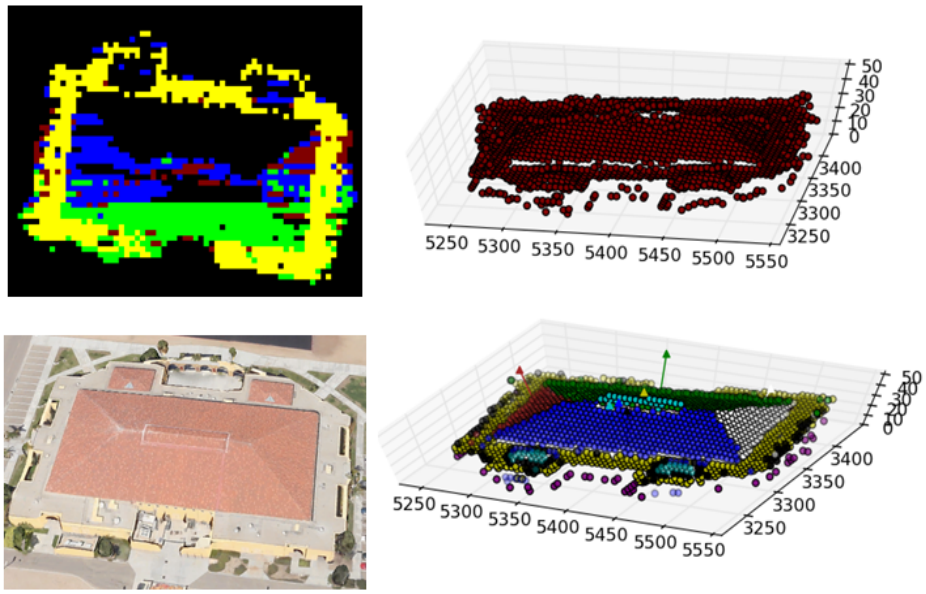

Figure 13: LiDAR Roof Extraction Plug-In results for the building 230

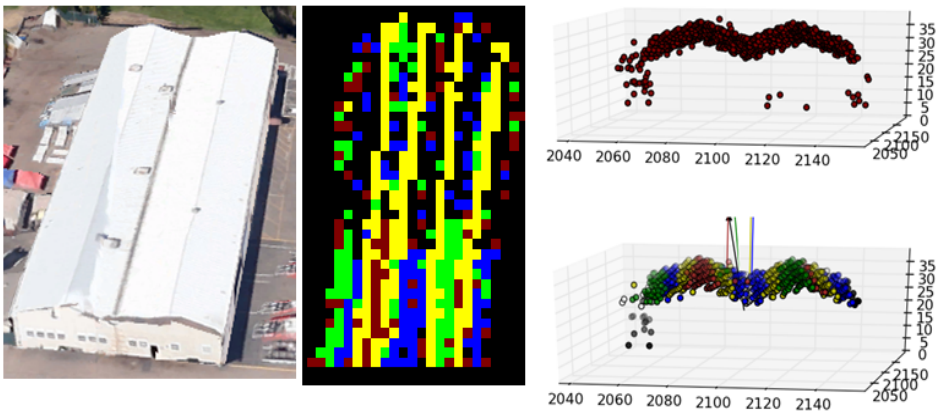

Figure 14: LiDAR Roof Extraction Plug-In results for the building 171

The *Gui* class is used to create the Graphical User Interface of the Plug-In and is based on the Qt framework (moc_Gui.cpp file must be generated).

The *LiDAR_roof_segmentation* is the class that interfaces with Opticks (it calls the *execute()* method).

The Plug-In has also other two classes: *RansacOld* and *Tutorial1*, that belong to the old version of the Plug-In; there isn't the GUI (the values of parameters are hard-coded) and all the processing stages are implemented inside the *RansacOld* class, while *Tutorial1* is the class that interfaces with Opticks.

2.2.1 Dependencies and compiling instructions

The code is hosted in the project repository (link), where, inside the LiDAR folder, there is the Visual Studio project file (LiDAR.vcxproj). To compile it, the user can add it to the Opticks SDK solution (the version 20140410 of the Opticks SDK was used), but first some steps are needed.

OPTICKS_CODE_DIR environment variable must be set: it is the path to the location where the Opticks SDK is installed.

All the Plug-In classes depend on the Eigen library (link), that it is not included in the Opticks dependencies: instructions on how to use it inside Visual Studio can be found at following link.

Since the Plug-In uses the OpenCV highgui methods, the default OpenCV property sheets (*SDK_HOME/application/CompileSettings*) must be changed (both the debug and release version), adding the *opencv_highgui220.lib* to the additional dependencies (line 11).

In addition, the *cxcore.h* header file (needed for the conversion from/to OpenCV - Eigen matrices) must be copied in the OpenCV2 core folder (*SDK_HOME/Dependencies/64/include/opencv2/core*).

Then, to run the Plug-In (Opticks 4.12 Nightly Build 20140410 was used), the *open_cv_highgui220.dll* and *open_cv_highgui220d.dll* must be copied from the SDK dependencies folder (*SDK_HOME/Dependencies/64/bin*) to the Opticks installation bin folder (*C:/Program Files/Opticks/4.12Nightly20140410.18697/Bin*).