



# PCL GROUND FILTER

---

*VOULIVASI TRIANTAFYLLIA*

This report describes the process of ground filtering LIDAR data using the C++ PCL library. This project was assigned to me during my internship in Advanced Technology Development Section from June 2016 to August 2016.

*8/25/2016*

---

# PCL GROUND FILTER

## Contents

LIDAR DATA PROCESSING – GROUND FILTERING.....	2
Automated Ground Filtering .....	2
Ground Filter User .....	2
SYSTEM AND TOOLS DESCRIPTION.....	5
Computer System.....	5
Software Tools .....	5
<i>CloudCompare</i> .....	5
<i>PCL</i> .....	5
<i>PDAL</i> .....	5
GROUND FILTER EXPLANATION .....	6
Technical Details.....	6
Code details .....	7
<i>Input</i> .....	7
<i>Output</i> .....	7
Visual Example .....	7
FILTER IMPLEMENTATION.....	10
UML Diagram.....	10
DEFAULT VALUES.....	11
ESTIMATED VALUES.....	12
BIG DATA FUNCTIONALITY .....	14
UML Diagram.....	14
Explanation.....	15
MANUAL.....	17
CONVERT RAW .LAS TO .PCD .....	17
C++ & PCL Application.....	17
GROUND FILTER EVALUATION .....	18
Visualization Analysis.....	18
Error Analysis .....	18
Evaluation Implementation .....	20
Code .....	20
<i>Input</i> .....	20
<i>Output</i> .....	20
<i>Console screenshot:</i> .....	20
ISSUES .....	21

## LIDAR DATA PROCESSING – GROUND FILTERING

### Automated Ground Filtering

Automated filters are built into lidar production software tools. In the early days of lidar, data acquisition vendors developed proprietary tools as part of their corporate intellectual property.

Automated filters attempt to identify a trend surface; points on that surface are classified as “ground.” The trick to automated filtering is to set appropriate parameters for terrain type (flat, rolling, hilly) and terrain cover (open, light vegetation, medium vegetation, heavy vegetation) to remove as many above-ground features as possible, without chopping off the tops of hills or removing other raised features that are legitimate parts of the natural landscape. Also, it is likely that within a single dataset, the terrain type and terrain cover vary. Adaptive filters can be developed that attempt to identify these areas, so that appropriate parameters are applied locally during the filtering process.

Automated filtering can remove up to about 80-90% of the above-ground features reliably, depending on the complexity of the terrain and vegetation in the project area. Automated filtering usually results in a 90 - 95 percent clean bare-earth surface.

### Ground Filter User

Refining the last 10% of the bare earth terrain model requires human interpretation; manual editing often takes the majority of the project labor budget allotted to post-processing. The final surface editing process is one of the most critical steps influencing vertical accuracy. Vegetation, buildings, vehicles, towers, etc., must be removed for reliable bare-earth-based analyses. It is helpful at this stage to have good assisting datasets, particularly orthophoto imagery. Lidar intensity imagery can be very helpful, especially if no orthophotos are available at an appropriate spatial or temporal resolution. It is also helpful to have a variety of viewing and editing tools, so that the data may be viewed in a variety of ways: as a point cloud, as a hillshade, as contours, or overlaid with transparent imagery. Individual points, or groups of points, can then be manually reclassified based on the judgment of the editor.

In the early days of lidar editing, non-ground points were simply deleted from the data file as unwanted points. But one user's trash is often another user's treasure, and throwing points away makes it difficult for quality control personnel to review editing decisions. The philosophy that points should be flagged with a classification, rather than be deleted from the dataset, emerged during the development of the LAS data format standard. A standardized classification scheme, (see Table 1) was adopted; most data acquisition contracts refer to this standard, and commercial software adheres to this classification scheme as the default during automated filtering and manual editing.

## PCL GROUND FILTER

ASPRS Standard LIDAR Point Classes	
Classification Value	Meaning
0	Created, never classified
1	Unclassified
2	Ground
3	Low Vegetation
4	Medium Vegetation
5	High Vegetation
6	Building
7	Low Point (noise)
8	Model Key-point (mass point)
9	Water
10	<i>Reserved for ASPRS Definition</i>
11	<i>Reserved for ASPRS Definition</i>
12	Overlap Points
13 - 31	<i>Reserved for ASPRS Definition</i>

Table 1: LAS Standard Classification Scheme.

Source: ASPRS

Figure 1 shows a profile of unclassified lidar points, where the color red corresponds to LAS Class 0. All of the lidar points in the profile are red, yet the ground surface and features (likely vegetation) above the ground are easily interpreted visually.

## PCL GROUND FILTER



Figure 1: Profile view of unclassified lidar data.

*Source: Fugro EarthData*

Figure 2 is the same profile after automated filtering, with ground points (Class 2) shown in tan, and above ground points (Class 1) shown in light blue.



Figure 2: Profile view of classified lidar data.

*Source: Fugro EarthData*

## SYSTEM AND TOOLS DESCRIPTION

Here we describe the computer system and libraries used for this project.

### Computer System

Processor: Intel(R) Core(TM)2 Duo CPU T9800 @2.93GHz 2.94 GHz  
RAM: 4 GB  
System type: 64-bit OS

### Software Tools

#### CloudCompare

*CloudCompare is a 3D point cloud (and triangular mesh) processing software. It has been originally designed to perform comparison between two 3D points clouds (such as the ones obtained with a laser scanner) or between a point cloud and a triangular mesh. Afterwards, it has been extended to a more generic point cloud processing software, including many advanced algorithms (registration, resampling, color/normal/scalar fields handling, interactive or automatic segmentation, etc. It relies on a specific octree structure that enables great performances in this particular function<sup>1</sup>.*

#### PCL

*Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. PCL is released under the terms of the BSD license, and thus free for commercial and research use<sup>2</sup>.*

This library contains different tools to read point cloud data from file with different formats, such as PCL, PLY, OBJ, as well as the PCD format: extension created to complement other point cloud file formats. PCL library provides tools to build kd-trees or octrees to simplify searches but also tools to filter data. The **pcl\_segmentation** library in PCL 1.7 version and afterwards contains, among others, a ground filter for 3D point cloud data.

PCL 1.7 is a bit difficult to install in Windows, so please follow the corresponding tutorials in the attached files:

- **HOW TO link PCL 1.7 in VS2010.pdf**
- **HOW TO install PCL 1.7.pdf**

#### PDAL

*PDAL is a C++ BSD library for translating and manipulating point cloud data. It is very much like the GDAL library which handles raster and vector data. In addition to the library code, PDAL provides a suite of command-line applications that users can conveniently use to process, filter, translate, and query point cloud data<sup>3</sup>.*

---

<sup>1</sup> CloudCompare is available at: <http://www.danielgm.net/cc/>

<sup>2</sup> PCL is available at: <http://pointclouds.org/>

<sup>3</sup> PDAL is available at: <http://www.pdal.io/>

## GROUND FILTER EXPLANATION

PCL Implements the Progressive Morphological Filter for segmentation of ground points. This is the ground filter used in this project.

A complete description of the algorithm can be found in the article [“A Progressive Morphological Filter for Removing Nonground Measurements from Airborne LIDAR Data”](#) by K. Zhang, S. Chen, D. Whitman, M. Shyu, J. Yan, and C. Zhang. For more information on how to invoke this PCL-based filter programmatically, see the [ProgressiveMorphologicalFilter](#) tutorial on the PCL website. For more detailed code explanation please find in the attached files:

- FILTER\_EXPLANATION\_PCL.pdf
- FILTER\_EXPLANATION\_ParameterTuning.pdf
- FILTER\_EXPLANATION\_algorithm.xlsx

### Technical Details

The Progressive Morphological Filter works on a minimum surface grid (2D) which is generated by using the minimum elevation values in each grid cell. The morphological operation of opening – erosion followed by dilation – is used to filter the grid surface. This operation is repeated iteratively – each time with a different window size and a different height threshold.

- $w_k$ : window sizes
  - $w_k = c * (2b^k + 1)$   
window size for exponential increase
  - $w_k = c * (2kb + 1)$   
window size for linear increase
- $b$ : the base of the initial window size
- $c$ : the grid cell size
- $k$ : number of the iteration
- $ht_k$ : height threshold values
  - $ht_k = \begin{cases} dh_o & \text{if } w_k \leq 3 (k = 0) \\ s(w_k - w_{k-1})c + dh_o & \text{if } w_k > 3 \\ dh_{max} & \text{if } ht_k > dh_{max} \end{cases}$
- $s$ : the terrain slope[rise/run]
- $dh_o$ : the initial elevation difference threshold
- $dh_{max}$ : the maximum elevation difference threshold



## PCL GROUND FILTER

### Code details

#### Input

- A point cloud
- A list of parameters

	NAME	TYPE	PCL DEFAULT VALUES
1	max_window_size_	Int	max_window_size_ (33)
2	slope_	Float	slope_ (0.7f)
3	max_distance_	Float	max_distance_ (10.of)
4	initial_distance_	Float	initial_distance_ (0.15f)
5	cell_size_	Float	cell_size_ (1.of)
6	base_	Float	base_ (2.of)
7	exponential_	Bool	exponential_ (true)

#### Output

- Indices of ground points

### Visual Example

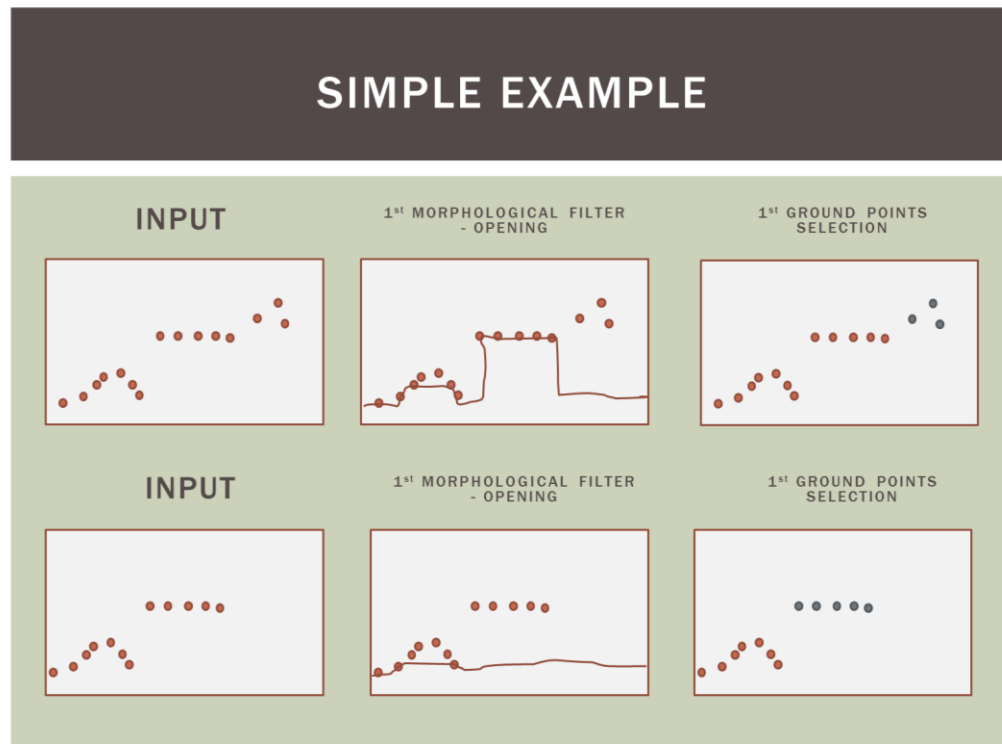


Figure 3



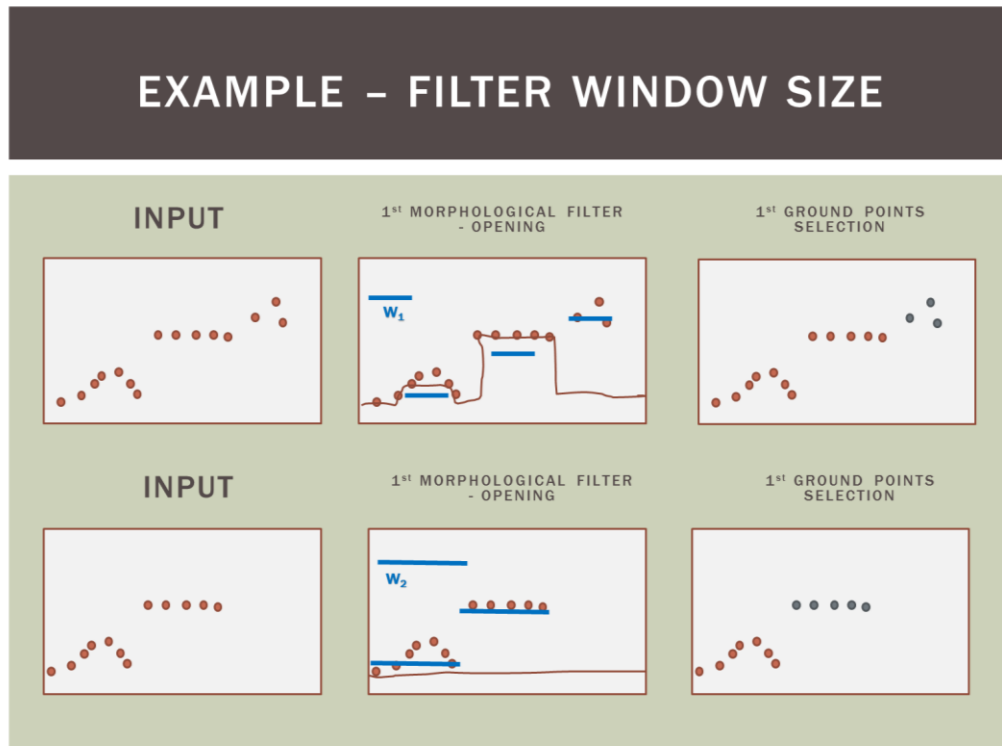


Figure 4

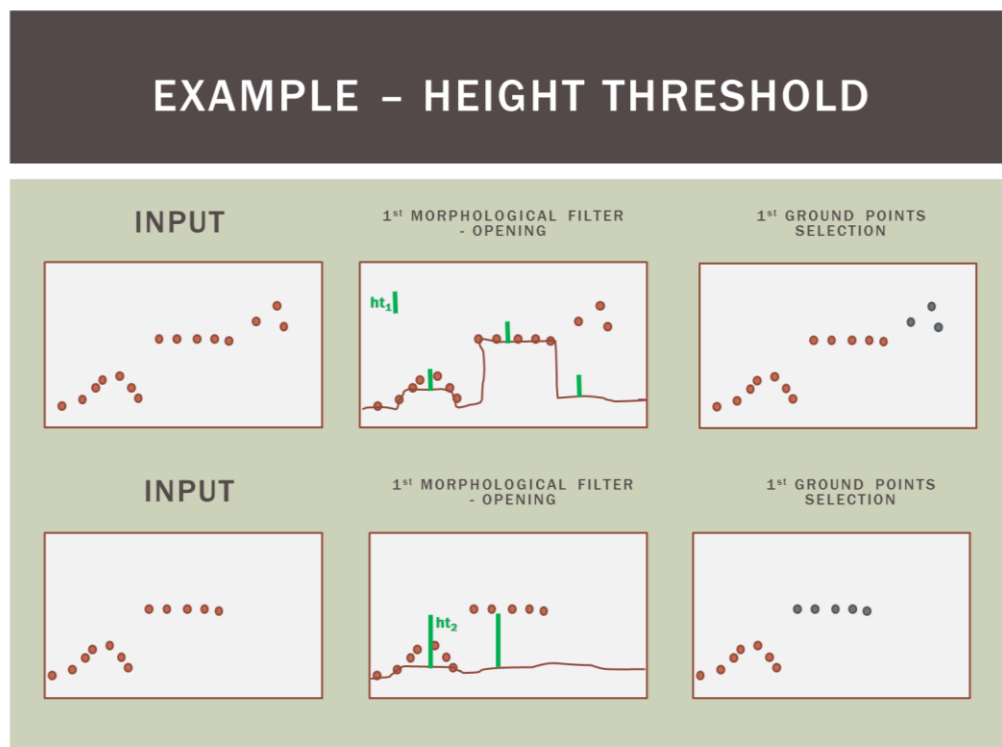


Figure 5

## EXAMPLE - INDICES

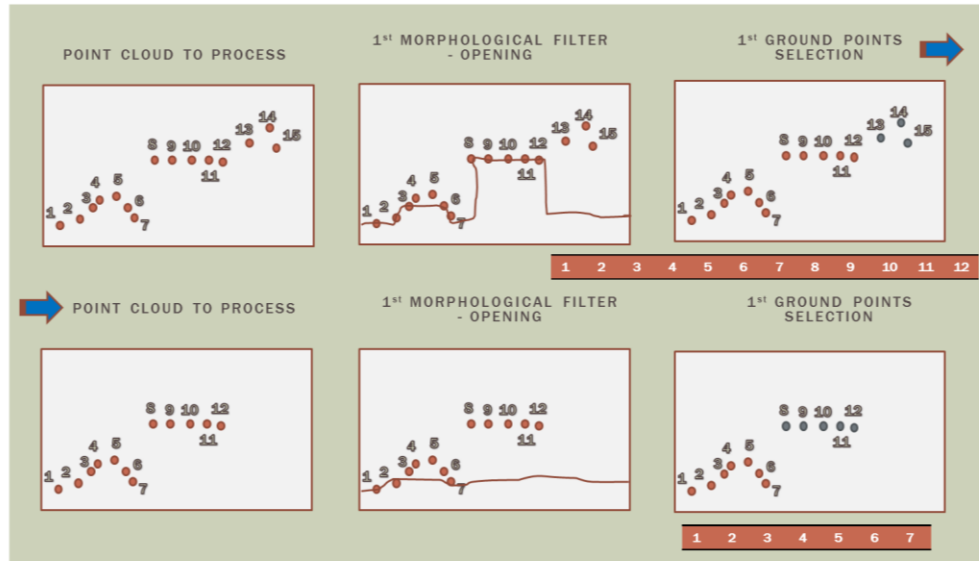


Figure 6

## OUTPUT

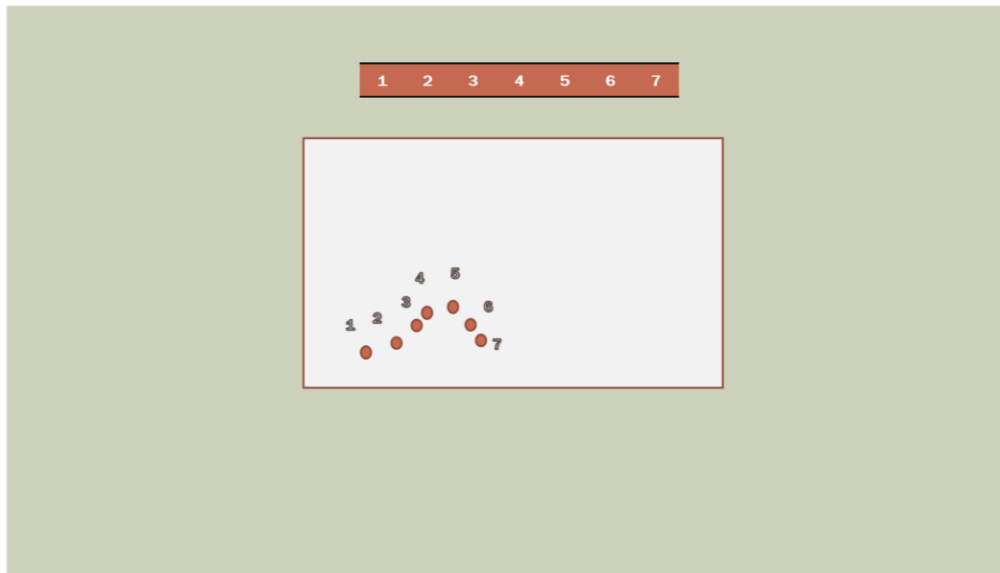
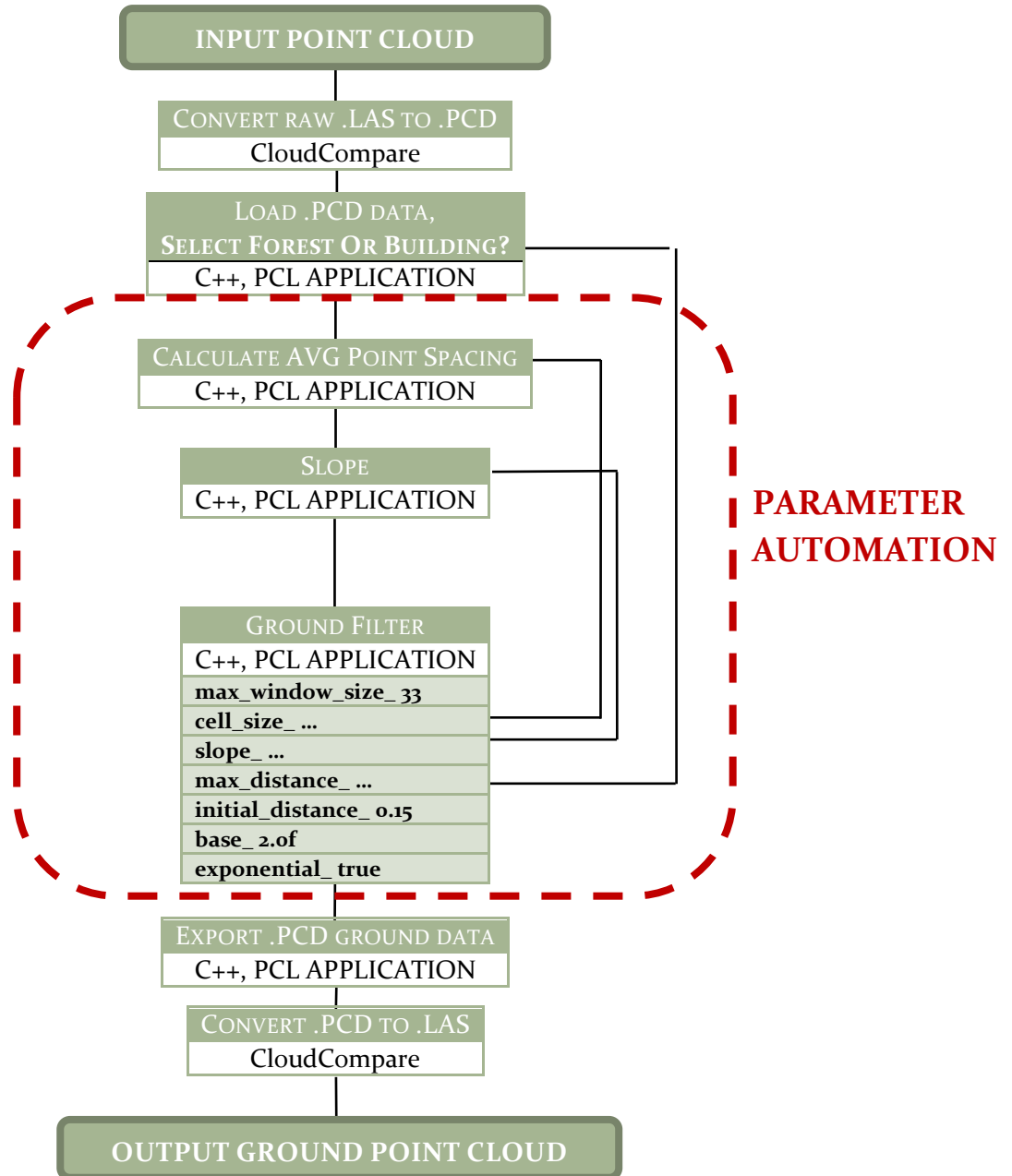


Figure 7

## FILTER IMPLEMENTATION

The Progressive Morphological Filter needs many parameters, so a method to automate the filtering process was implemented. Here are the sub-processes assembled together to create this filter method.

### UML Diagram



## PARAMETER AUTOMATION

Related code: `ground_filter.cpp`

Some of the parameters were given a default value after evaluation, where others are estimated using the input point cloud.

## DEFAULT VALUES

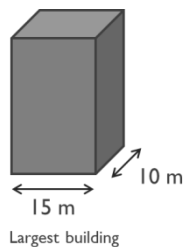
After filtering the sample data and evaluating, the following values performed the best. You can see the evaluation results in the attached file:

- **EVALUATION-SAMPLE DATA-DEFAULT PARAMETERS.xlsx**

# max\_window\_size\_

---

The maximum window size has to be greater than the largest objects (X or Y) in the experimental area.



The maximum window size is set to 33 as the default value.

# base\_ 2.of exponential\_ true

---

After testing of different sets of parameters, we noticed better performance with the exponential increasing method and base equal to 2.

# initial\_distance\_ 0.15f

---

The initial elevation difference threshold

The initial distance is the initial elevation difference threshold and is set fixed to 0.15.

## ESTIMATED VALUES

## cell\_size\_

Related code: avg\_point\_spacing.cpp, main.cpp

To preserve most of the original points, it is suggested to choose the cell size to be two times the average spacing between points. However, evaluation was used to decide the CELL\_FACTOR for which the expression cell size = CELL\_FACTOR \* average point spacing is performing better in the ground filter.

CELL\_FACTOR is defined equal to 2 in main.cpp and can be changed if you noticed any anomalies in further data processing.

## Average Point Spacing Calculation

When converting the data to a Minimum 2D grid from a given Lidar point dataset, the grid cell size must be based on the post-spacing of the Lidar points, and thus the method used to determine the post spacing is critical.

Firstly, to reduce calculations we take a representative sample of our data arbitrarily (approximately 100.000 points).

Then we use only the X and Y values of each point. (We project every point in the XY plane.) To calculate the average point spacing, we calculate the distance from every point to its closest neighbor, and then we take the average.

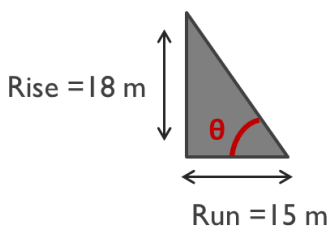
## Average Point Spacing - TESTING

To test the above method, a dataset of line points was created, with mutable distance between each point and its closest neighbor.

## slope\_

Related code: slope.cpp

The terrain slope =  $\tan\theta = \frac{\text{elevation between two points}}{\text{distance between two points}} = \frac{\text{rise}}{\text{run}}$ .



To estimate **slope**, we use the same method described in ESRI's 3D Analyst website:

[http://edndoc.esri.com/arcobjects/9.2/net/shared/geoprocessing/geoprocessing\\_with\\_3d\\_analyst/calculating\\_slope\\_in\\_3d\\_analyst.htm](http://edndoc.esri.com/arcobjects/9.2/net/shared/geoprocessing/geoprocessing_with_3d_analyst/calculating_slope_in_3d_analyst.htm)

To calculate slope another factor needs to be calculated first: **cell\_size**. The **cell\_size** is calculated using the method described above.

After we have calculated the slope in degrees for every point, we take the average value and calculate the tangent. This is our estimation for the **slope** parameter.

### *Slope Estimation – TESTING*

To test the above method, we can use ESRI's 3D Analyst to find the average slope and compare the results for certain datasets.

## max\_distance\_

---

Related code: max_elevation_diff.cpp
--------------------------------------

The maximum distance is depending on whether the area of interest is a building “B” or a forest “F” area.

### Case “B” – Building area:

Max distance has to be less than the lowest building height in a building area, so the max\_distance\_ parameter is set fixed to 3m.

### Case “F” – Forest area:

The max\_distance\_ parameter is set equal to the maximum elevation difference.

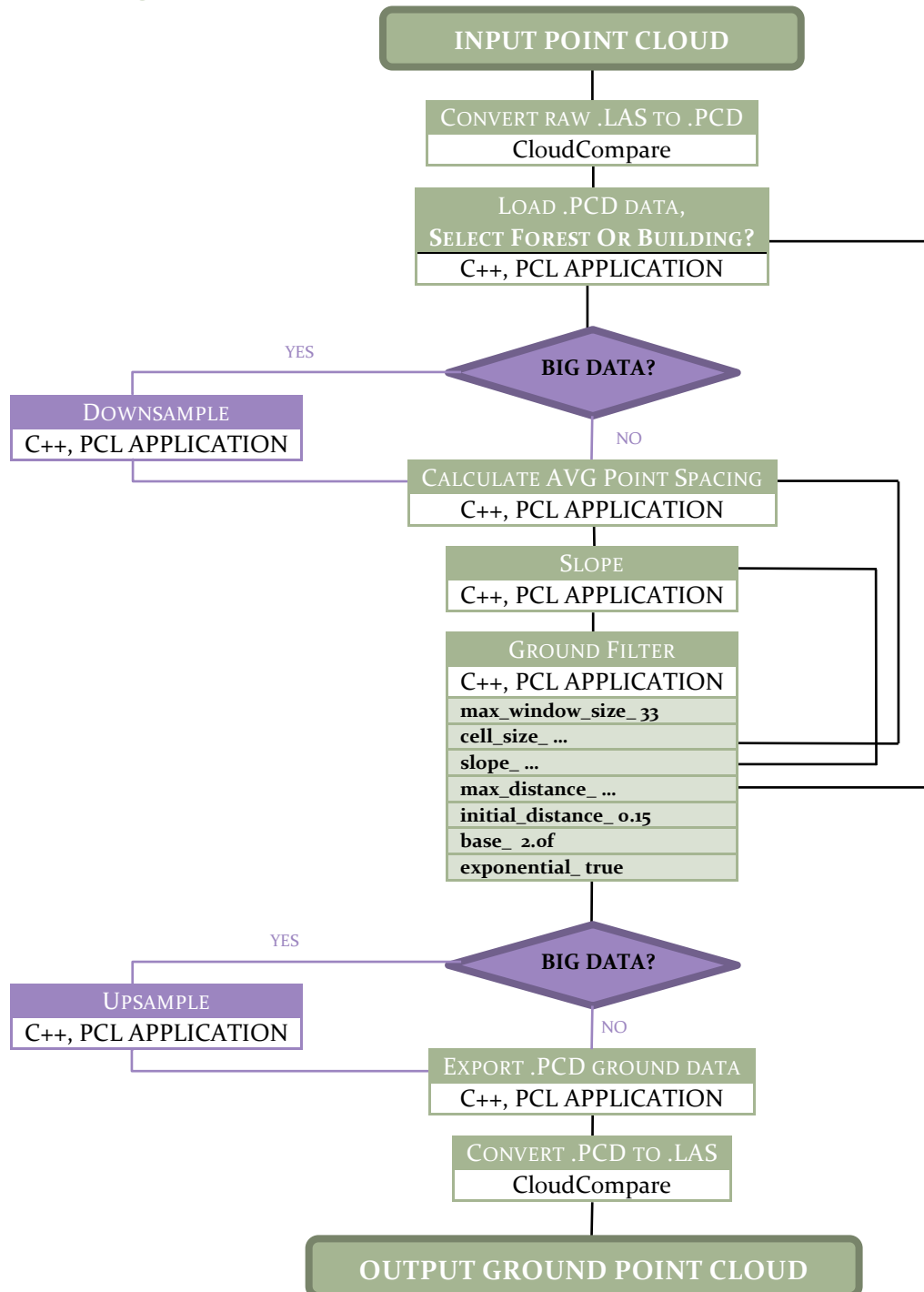
This is the only parameter the user has to enter.

## BIG DATA FUNCTIONALITY

Related code: `downsample.cpp`, `upsample.cpp`, `main.cpp`

Processing point data is becoming more and more challenging due to the growing data volume. An idea to process big point cloud data follows:

### UML Diagram





## PCL GROUND FILTER

### Explanation

#### *BIG DATA*

We set a `point_limit` variable to define when we think we have Big Data.

i.e : `point_limit = 35000`; → If the input Point Cloud is more than 35000 points, then we have Big Data.

#### *OCTREE*

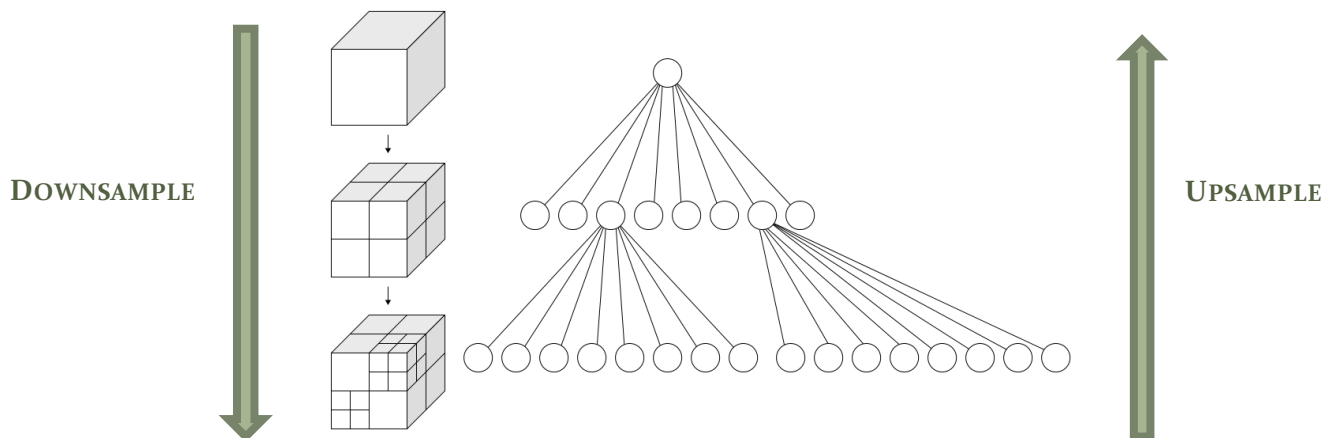
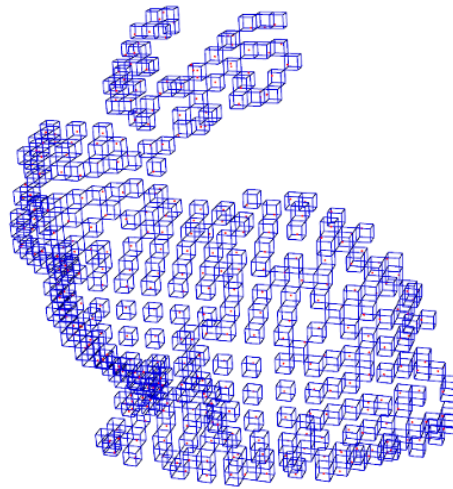


Figure 8: Octree

## PCL GROUND FILTER

After testing several values of octree resolution to achieve number of downsampled points equal to the point limit, I found that the average point spacing is somehow related with the average point spacing.

The following table (Table 2) describes this relation after testing:

POINT LIMIT	DOWNSAMPLE FACTOR
50000 points	40
35000 points	45
20000 points	60
3000 points	130

Table 2

So a DOWNSAMPLE\_FACTOR and a BIG\_DATA\_POINT\_LIMIT are set as default at the beginning of the main.cpp function. Then we choose a resolution for the octree such as:

resolution / average point spacing = DOWNSAMPLE\_FACTOR

## MANUAL

### CONVERT RAW .LAS TO .PCD

The conversion is very simple using CloudCompare:

1. Open CloudCompare.
2. Drag and Drop the .las file
3. Check only R, G, B and Apply
4. Click No
5. Select the .las file and Save as .pcd

### C++ & PCL Application

A C++ console application was created with:

INPUT:

`my_cloud.pcd`      `f`

*my\_cloud.pcd*: the original point cloud in PCD format

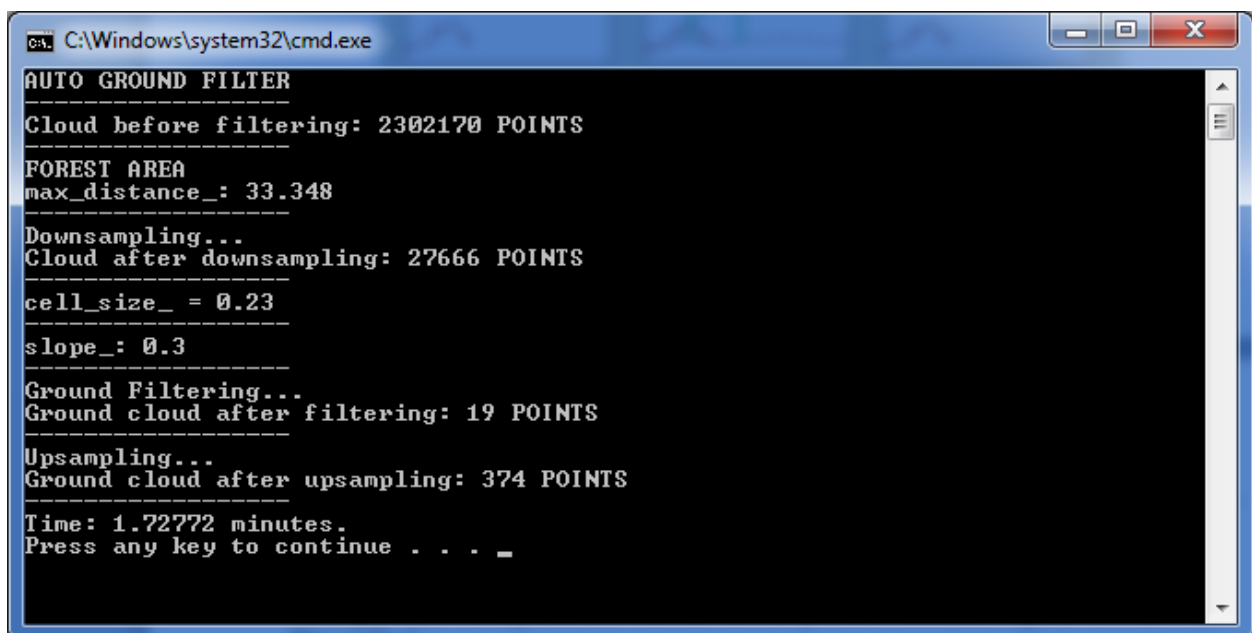
*f*: for Forest area. Please choose “f” or “b” depending on your area. “f” refers to Forest and “b” refers to Building area.

OUTPUT:

`my_cloud_ground.pcd`

*my\_cloud\_ground.pcd*: the filtered ground data in PCD format

Below (Figure 9) is a console screenshot.



```

C:\Windows\system32\cmd.exe
AUTO GROUND FILTER
Cloud before filtering: 2302170 POINTS
FOREST AREA
max_distance_: 33.348
Downsampling...
Cloud after downsampling: 27666 POINTS
cell_size_ = 0.23
slope_: 0.3
Ground Filtering...
Ground cloud after filtering: 19 POINTS
Upsampling...
Ground cloud after upsampling: 374 POINTS
Time: 1.72772 minutes.
Press any key to continue . . . _
  
```

Figure 9

## GROUND FILTER EVALUATION

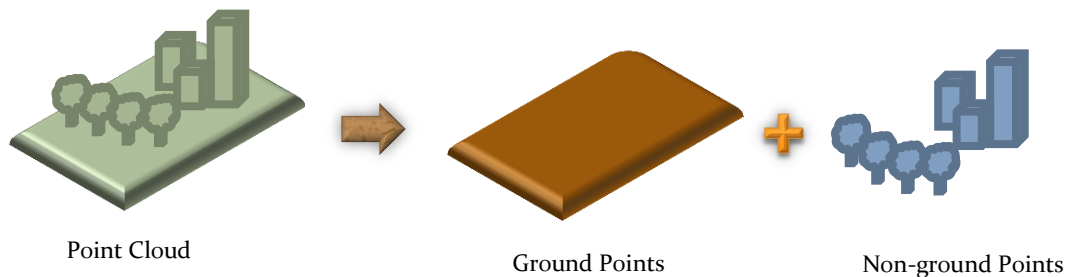
Only visualization analysis has been conducted using the finished process in the sample datasets due to lack of time. Error analysis has been conducted in every sub-process and should be also conducted for the final process presented in this report.

### Visualization Analysis

Visualization is another effective way of evaluating the ground points. The human eye can easily distinguish between ground and non-ground points and evaluate the quality of the filter.

### Error Analysis

We need 1 or more 3D point clouds with ground and non-ground points already classified.



After that we can evaluate the ground filter by calculating the following errors:

- **omission error** = how many ground points have been mistakenly removed?
- **commision error** = how many non-ground points have been classified as ground measurements?

These errors can be calculated by comparing the already classified points and those after filtering.

EXAMPLE(Point indices):

ALREADY CLASSIFIED		AFTER FILTER	
GROUND	NON-GROUND	GROUND	NON-GROUND
1	2	1	4
3	7	2	5
4	8	3	7
5	9	6	8
6	10		9
	11		10
			11

Omission Errors	Commision Errors
4	2
5	

We have 2 omission error points and 1 commission error point

## PCL GROUND FILTER

The following errors are used for evaluation:

Type I error                      Omission Error / Classified Ground Points

Type II error                      Commission Error / Classified Non-Ground Points

**Total error**                      **(Omission Error + Commission Error) / Total Points**

The following equations can be used:

TP: Total Points

CG: Classified Ground Points

CNG: Classified Non Ground Points

FG: Filter Ground Points

CE: Commission Error Points

OE: Omission Error Points



TP: 11

CG: 5

CNG: 6

FG: 4

CE: 2

OE: 1

---

$$\text{CG} = \text{FG} - \text{CE} + \text{OE} \quad \rightarrow \quad 5 = 4 - 2 + 1$$

$$\text{TP} = \text{CG} + \text{CNG} \quad \rightarrow \quad 11 = 5 + 6$$

$$\text{Type I error} = \text{OE} / \text{CG} \quad \rightarrow \quad \text{Type I error} = 1 / 5 = 20\%$$

$$\text{Type 2 error} = \text{CE} / \text{CNG} \quad \rightarrow \quad \text{Type 2 error} = 2 / 6 = 33,33\%$$

$$\text{Total Error} = \frac{\text{OE} + \text{CE}}{\text{TP}} \quad \rightarrow \quad \text{Total Error} = \frac{1 + 2}{11} = 27,27\%$$

---

## PCL GROUND FILTER

### Evaluation Implementation

To evaluate the performance of this filter, a Forest area and a Building area LAS data with already classified ground attributes were used.

The evaluation point cloud files and some of the evaluation results are shown in the attached excel files.

### Code

An evaluation application was created with PCL.

### Input

`my_cloud.pcd`      `my_cloud_ground_classified.pcd`      `my_cloud_ground.pcd`

*my\_cloud.pcd*: the original point cloud in PCD format

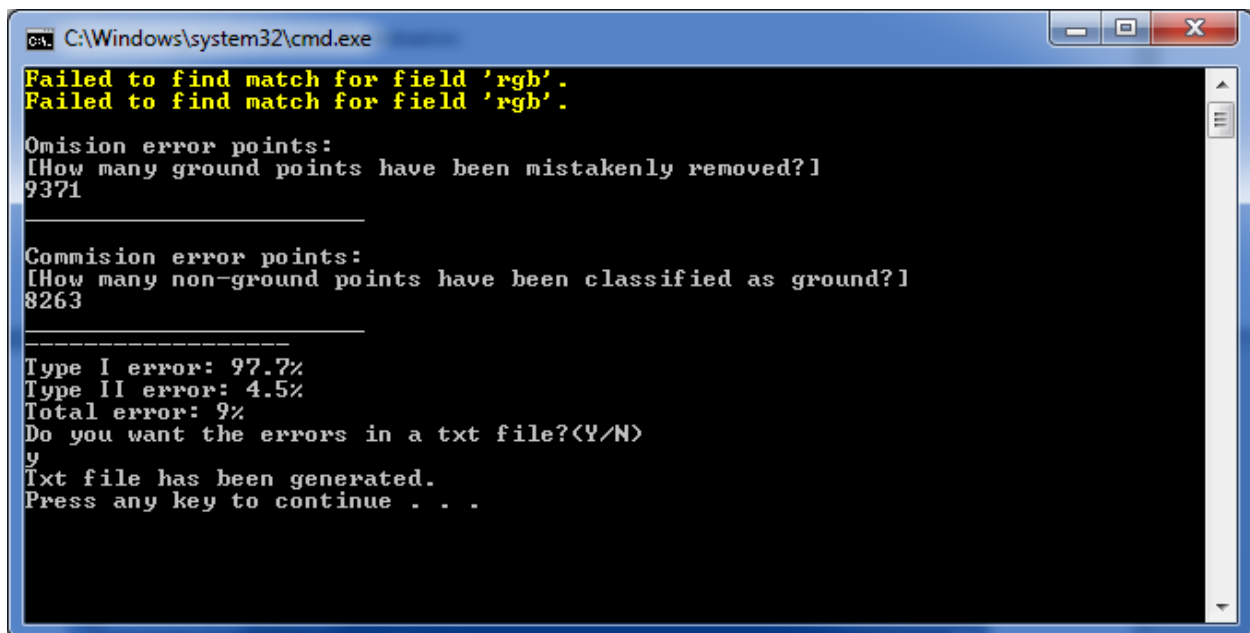
*my\_cloud\_ground\_classified.pcd*: the original point cloud classified data in PCD format

*my\_cloud\_ground.pcd*: the filtered ground data in PCD format

### Output

Evaluation errors in the console or in a text file

### Console screenshot:



```
C:\Windows\system32\cmd.exe
Failed to find match for field 'rgb'.
Failed to find match for field 'rgb'.

Omission error points:
[How many ground points have been mistakenly removed?]
9371

Commision error points:
[How many non-ground points have been classified as ground?]
8263

-----
Type I error: 97.7%
Type II error: 4.5%
Total error: 9%
Do you want the errors in a txt file?(Y/N)
y
Txt file has been generated.
Press any key to continue . . .
```

Figure 10

### ISSUES

- This process has not been tested in point clouds that contain buildings.
- Better Big Data functionality could be achieved with further study of Octrees and lead to less ground filter errors.
- Conversion from .las to .pcd and backwards could be done automatically.
- Classified Point Clouds are needed in order to evaluate properly.
- Only visualization analysis has been conducted using the finished process in the sample datasets due to lack of time. Error analysis should be also conducted.
- Minor code implementation error → Related to C++ Boost library → `boost::checked_delete`