

Point cloud processing I

Lesson 12*

1	Point cloud file formats	1
1.1	ASCII formats	1
1.2	The PLY format	2
1.3	The LAS format	2
2	Thinning	5
3	Outlier detection	6
4	Notes & comments	7


In this lesson we discuss the basics tools and algorithms necessary for point cloud processing: storage in various file formats, thinning, and outlier detection.

1 Point cloud file formats

A point cloud is essentially a array of 3D points, and often that is also how it is stored in a file. Regardless of the format, a point cloud file can often be seen as a array of *point records*, each of which contains the coordinates and attributes of one point. A point record consists of several *fields*, each of which stores a single value, eg an integer, float, or boolean. A field can for instance represent the x , y , or z -coordinate of a point or one of its attributes, eg the lidar return number or color information. The order and meaning of the fields in a record is fixed for all the point records in one file. How exactly the point records are structured and stored in the file, and what additional metadata is available, depends on the specific file format that is used.

1.1 ASCII formats

ASCII formats are plain text files. The point cloud information is thus stored as a sequence of ASCII characters, usually one point record per line. In most cases you can recognise such files by the `.xyz`, `.csv`, or `.txt` extension. A benefit of ASCII files is that you can simply open them in a text editor. The biggest downside is that they are not standardised at all, ie the type, order, and number of attributes varies and also the used CRS is usually not documented in the file.

* Ravi Peters, Hugo Ledoux, Ken Arroyo Ohori. This work is licensed under a Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>) (last update: January 8, 2019)

```

ply
format ascii 1.0 ← encoding and ply version number
comment This is an example file for GEO1015!
element vertex 7 ← number of points, start of point record definition
property float x
property float y
property float z
property int custom_attribute
end_header
91443.89 438385.69 -0.80 11
91443.94 438386.10 -0.78 43
91444.00 438386.51 -0.79 44
91444.06 438386.94 -0.83 31
91444.11 438387.36 -0.86 31
91443.88 438383.50 -0.83 22
91443.93 438383.91 -0.80 65

```

Figure 1: A simple PLY file with 1 additional user defined attribute of type int. It contains 7 points.

1.2 The PLY format

The PLY format can be considered a standardised ASCII format. A PLY file contains a header¹ that specifies the structure of the point records in the file, ie the number of attributes (called *properties*), their order, their names and their data types. This makes it a very flexible standard, since the user can decide on the composition of the point record. Figure 1 shows an example PLY file.

PLY files are readable by many software packages and can also be stored in a binary encoding². Compared to the ASCII encoding, the binary encoding results in a smaller file size and quicker reading and writing from and to the file. There is no standardised way to specify the CRS in a PLY file, although one could add a comment in the header that states the CRS.

1.3 The LAS format

The public LASER (LAS) file format is the most widely used standard for the dissemination of point cloud data. The LAS standard is maintained by the ASPRS organisation and, as the name implies, it was designed for datasets that originate from (airborne) lidar scanners. However, in practice it is also used for other types of point cloud, eg those derived from dense image matching. It is a binary-encoded standard and compared to the PLY format it is rather strict because it prescribes exactly what a point record should look like, ie what attributes are there and how many bits each attribute should use.

Table 1 shows the composition of the simplest record type that is available for LAS files. Other record types are available that also include fields to store for instance RGB color information or the GPS time (the time a point was measured by the scanner), but all records types include at least the fields shown in Table 1. While the LAS standard clearly specifies that all these fields are required, some of the fields are very specific to lidar acquisition and they are

¹A header is supplemental information placed at the beginning of a file, eg to store metadata about the file.

²[https://en.wikipedia.org/wiki/PLY_\(file_format\)#ASCII_or_binary_format](https://en.wikipedia.org/wiki/PLY_(file_format)#ASCII_or_binary_format)

Field	Format	Length (bits)	Description
X	int	32	X coordinate.
Y	int	32	Y coordinate.
Z	int	32	Z coordinate.
Intensity	unsigned int	16	The pulse return amplitude.
Return number	unsigned int	3	The total pulse return number for a given output pulse.
Number of returns	unsigned int	3	Total number of returns for a given pulse
Scan Direction Flag	boolean	1	Denotes the direction at which the scanner mirror was traveling at the time of the output pulse. A bit value of 1 is a positive scan direction, and a bit value of 0 is a negative scan direction (where positive scan direction is a scan moving from the left side of the in-track direction to the right side and negative the opposite).
Edge of Flight Line	boolean	1	Has a value of 1 only when the point is at the end of a scan. It is the last point on a given scan line before it changes direction.
Classification	unsigned int	5	Classification code
Scan Angle Rank	int	4	The angle at which the laser pulse was output from the scanner including the roll of the aircraft.
User Data	unsigned int	4	May be used at the user's discretion.
Point Source ID	unsigned int	8	Indicates the file from which this point originated. Non-zero if this point was copied from another file.

Table 1: LAS Point Data Record Format 0

Code	Meaning
0	never classified
1	unclassified
2	ground
3	low vegetation
4	medium vegetation
5	high vegetation
6	building
7	low point (noise)
8	<i>reserved</i>
9	water

Table 2: The first 10 LAS classification code numbers. More codes exist, but they are not listed here.

sometimes ignored in practice, eg if a point cloud originating from dense matching is stored in the LAS format. Notice that unused fields will still take up storage space in each record.

The CRS of the point cloud can be stored in the header of a LAS file, together with some other general information such as the total number of points and the bounding box of the point cloud. The X, Y, and Z fields are stored as 32-bit integers. To convert these values to the actual coordinates, they need to be multiplied by a scaling factor and added to an offset value, ie:

$$X_{coordinate} = (X_{record} * X_{scale}) + X_{offset}$$

$$Y_{coordinate} = (Y_{record} * Y_{scale}) + Y_{offset}$$

$$Z_{coordinate} = (Z_{record} * Z_{scale}) + Z_{offset}$$

The scaling factors X_{scale} , Y_{scale} , Z_{scale} and the offsets X_{offset} , Y_{offset} , Z_{offset} are also given in the header. Notice that the scaling factor determines the number of decimals that can be stored, eg the factors 10, 100, and 1000 would give us 1, 2, and 3 decimals respectively.

The LAS standard defines several classification codes, as listed in Table 2. These codes are to be used as values for the classification field of a point record, and are intended to indicate the type of object a point belongs to. Which classes are used strongly depends on the dataset at hand. The codes 0 and 1 may appear ambiguous, but there is a clear distinction. To be exact, the code 0 is used for points that were never subject to a classification algorithm, whereas the code 1 is used for points that were processed by a classification algorithm, but could not successfully be assigned to a class. It is possible to define your own classes using code ranges that are reserved for that purpose in the standard. For example, the Dutch AHN3 dataset, which is disseminated in the LAS format, uses the code 26 for an 'artefact' class that includes infrastructural works such as bridges and viaducts (see Figure 2).

Finally, a compressed variant of the LAS format, dubbed the LAZ format, exists. While it is not maintained by an 'official' organisation like the LAS standard, it is an open standard and it is widely used, especially for very big dataset. Through the use of lossless compression algorithms, a LAZ file can be packed into a fraction of the storage space required for the equivalent LAS file without any loss of information. The LAZ format closely resembles the LAS format, ie the header and the structure of the point records are virtually identical. In a LAZ file the point records are grouped in blocks of 50,000 records each. Each block is individually compressed, which makes it possible to partially decompress only the needed blocks from a file (instead of always needing to decompress the whole file). This can save a lot of time if only a few point from a huge point cloud are needed. Notice that the effectiveness of the



Figure 2: Classification codes used in the AHN3 dataset.

compression algorithms depends on the similarity in information between subsequent point records. Typically information is quite similar for points that are close to each other in space. Therefore, a greater compression factor can often be achieved after spatially sorting the points.

2 Thinning

A point cloud with fewer points is easier to manage and quicker to process. Therefore a point cloud is sometimes *thinned*, which simply means that a portion of the points is discarded and not used for processing. Commonly encountered thinning methods in practice are

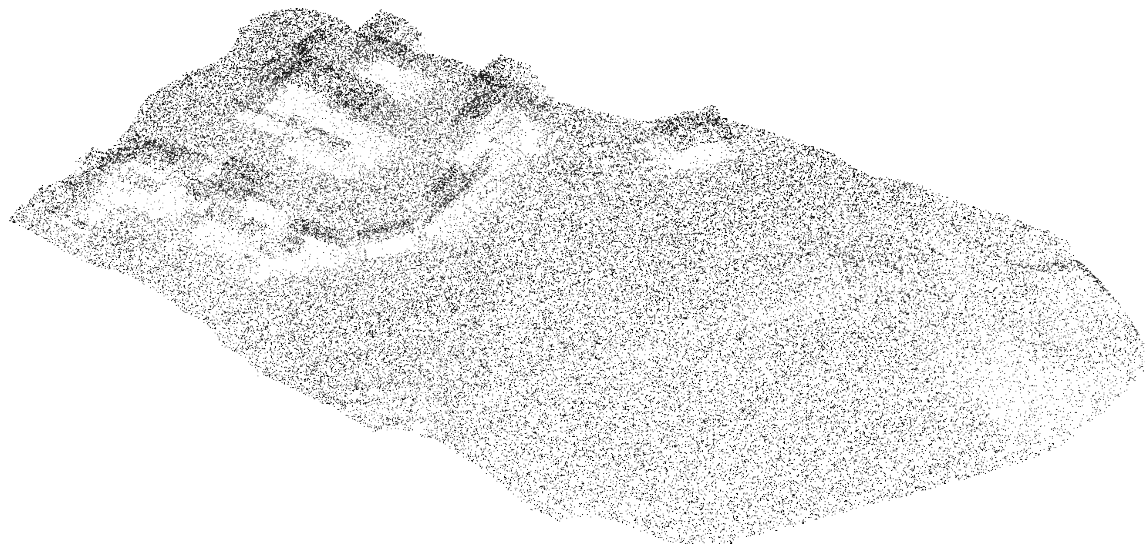
random randomly remove a given percentage of the points,

nth-point iterate through the points and keep only the first point for every n points. For example a dataset with 1000 points is reduced to 100 points if $n = 10$. This is the quickest thinning method.

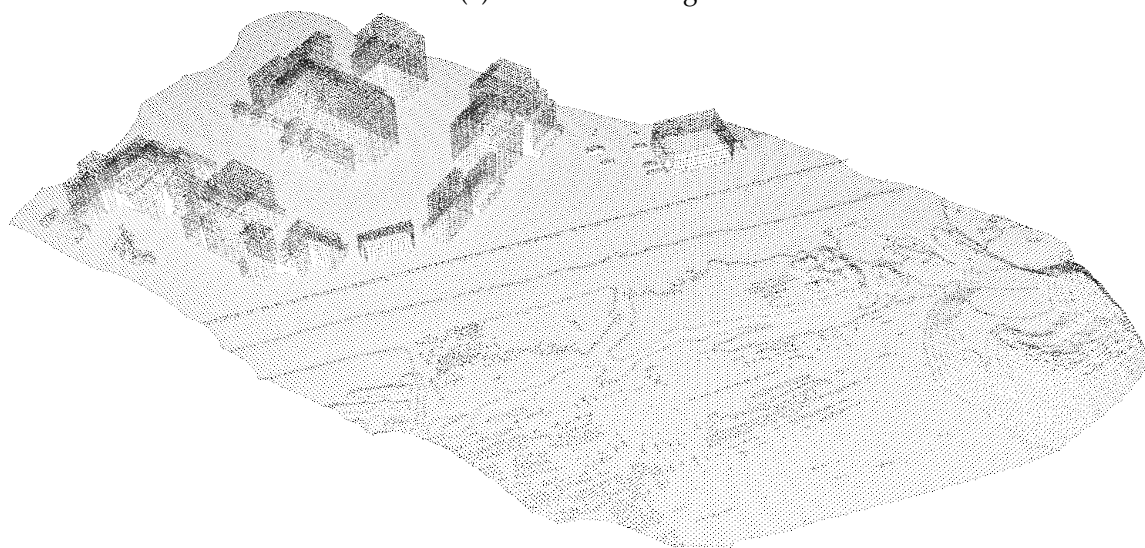
grid Overlay a 2D or 3D regular grid over the point cloud and keep one point per grid cell. That can be one of the original points, an average of those, or the exact center of the cell. The thinning factor depends on the chosen cell-size. Notice that the result is often a point cloud with a homogeneous point density on all surfaces (only on the horizontal surfaces if a 2D grid is used).

See Figure 3 for a comparison between random thinning and grid thinning. Observe the

From Lesson 08 you undoubtedly remember that TIN simplification has a somewhat similar objective: data reduction. However, for a given number of resulting points TIN simplification yields a higher quality end result because it only removes points that are deemed unimportant. Thinning methods on the other hand do not consider the 'importance' of a point in any way, and might discard a lot of potentially meaningful details. So why bother with thinning? The answer is that thinning methods are a lot faster since they do not require something like a computationally expensive triangulation. Especially in scenarios where the point density is very high and the available time is limited, thinning can be useful.



(a) random thinning



(b) 3D grid thinning

Figure 3: Comparison of two thinning methods. The thresholds were chosen such that the number of remaining points is approximately the same.

3 Outlier detection

Recall from Lesson 02 that outliers are points that have a large error in their coordinates. Outliers are typically located far away from the terrain surface and often occur in relatively low densities. Outlier detection aims to detect and remove outliers and is a common processing step for point clouds.

Most outlier detection methods revolve around analysing the local neighbourhood of a point. The neighbourhood can be defined using a k -nearest neighbour (knn) search, a fixed radius search, or by superimposing a regular grid on the point cloud and finding the points that are in the same grid-cell. The points that are determined to be in the neighbourhood of a point of interest \mathbf{p} are used to determine whether \mathbf{p} is an outlier or not.

The underlying assumption of most outlier detection methods is that an outlier is often an isolated point, ie there are not many points in its neighbourhood. We distinguish the following outlier detection methods (see also Figure 4):

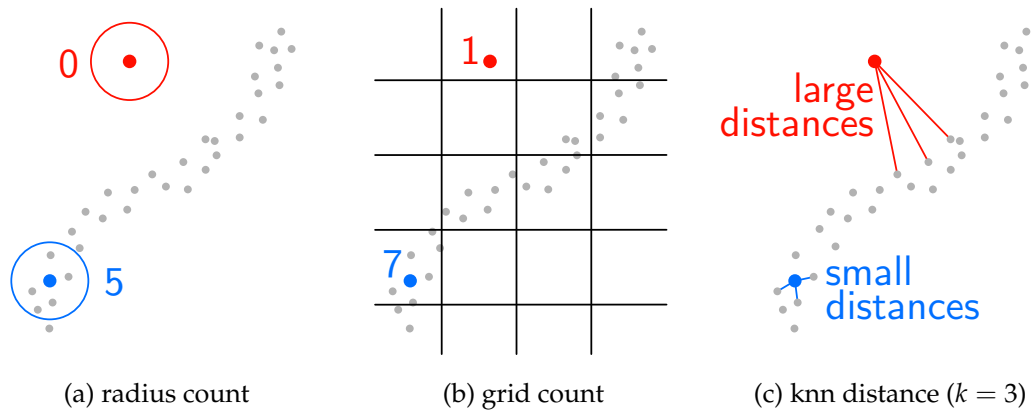


Figure 4: Three outlier detection methods based on local point density. The red point is an outlier, whereas the blue point is an inlier.

radius count count the number of points that are within a fixed radius from \mathbf{p} . If the count is lower than a given threshold, \mathbf{p} is marked as an outlier.

grid count Superimpose a grid on the point cloud and count for each grid-cell the number of points. If the count is lower than a given threshold, the points inside the corresponding grid cell are marked as outliers. Sometimes the neighbourhood is extended with adjacent grid cells. The grid method has the advantage that it can be used with the spatial streaming paradigm, that was explained in Lesson 10.

knn distance Find the k nearest neighbours of \mathbf{p} , eg using a kd-tree (see Lesson 10), and compute the mean or median of the distances between \mathbf{p} and its neighbours. If this value is above a given threshold, \mathbf{p} is marked as an outlier.

These methods generally work well if the outliers are isolated. However, in some cases this assumption does not hold. For example in case of a point cloud derived from multi-beam echo sounding³ a common issue is the occurrence of (shoals of) fish. These fish cause large groups of points that are clustered closely together above the seafloor. These are not isolated points since each outlier will have plenty of other points nearby. A possible solution is to construct a TIN of all points and to 'cut' the relatively long edges that connect the outlier clusters to the seafloor. This splits the TIN into several smaller TINs, and the largest of those should then be the seafloor surface without the outliers. Figure 5 gives an example.

4 Notes & comments

More information on the PLY format can be found online⁴. Notice that the PLY format can also be used to store a 3D mesh.

The full LAS specification is described in ASPRS (2013), and Isenburg (2013) describes the details of the compressed LAZ format.

³See Lesson 02.

⁴<http://paulbourke.net/dataformats/ply/>

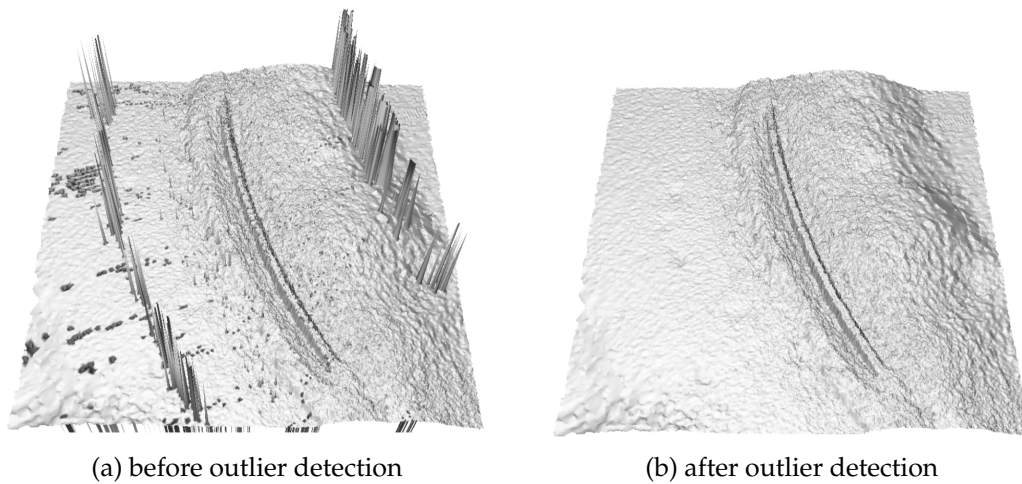


Figure 5: Outlier detection in a multi-beam echo sounding dataset using a TIN (Arge et al., 2010).

References & further reading

Arge L, Larsen KG, Mølhave T, and van Walderveen F (2010). Cleaning massive sonar point clouds. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10*, pages 152–161. ACM, New York, NY, USA.

ASPRS (2013). *LAS specification version 1.4 – R13*. The American Society for Photogrammetry & Remote Sensing.

Isenburg M (2013). LASzip: lossless compression of LiDAR data. *Photogrammetric Engineering and Remote Sensing*, 79(2):209–217.