

R workflow for ALS data pre-processing

Jean-Matthieu Monnet

2021-09-21

The R code below is designed for checking and pre-processing Airborne Laser Scanning (ALS, or lidar remote sensing) data. The workflow is based on functions of the package `lidR`, and includes the following steps:

- check the content of one file,
- create images and statistics from multiple files,
- compute digital surface models.

Licence: GNU GPLv3 / [source page](#)

ALS data

Files required for the tutorial are too large to be hosted on the gitlab repository, they can be downloaded as a zip file from Google drive, and should be extracted in the folder “data/aba.model/ALS/” before proceeding with the processing. Files can be automatically downloaded thanks to the `googledrive` package with the following code, this requires authenticating yourself and authorizing the package to deal on your behalf with Google Drive.

```
# set temporary file
temp <- tempfile(fileext = ".zip")
# download file from google drive
dl <- googledrive::drive_download(googledrive::as_id("1riPo-PLZ8_IjE7rAQ2RECj-fjg1UpC5i"),
  path = temp, overwrite = TRUE)
# unzip to folder
out <- unzip(temp, exdir = "../data/aba.model/ALS/")
# remove temporary file
unlink(temp)
```

Data checking

Single file

Load file

ALS data are usually stored in files respecting the ASPRS LAS specifications. Last version of LAS file format is 1.4. LAS files contain:

- metadata regarding the acquisition,
- basic attributes of each recorded point,
- additional attributes for each point (optional),
- waveforms (optional).

A single LAS file can be loaded in R with the `readLAS` function, which returns an object of class `LAS`. Some checks are automatically performed when the file is read. Function options allow to skip loading certain attributes, which might help saving time and memory. The object contains several slots, including:

- **header**: metadata read from the file,
- **data**: point cloud attributes,

- bbox: the bounding box of the data,
- proj4string: projection information.

It is advisable to fill the projection information if missing, as spatial objects computed from the point cloud will inherit the information.

```
# read file
point_cloud <- lidR::readLAS("../data/aba.model/ALS/tiles.laz/899500_6447500.laz")
```

```
## Warning: Invalid data: ScanAngleRank greater than 90 degrees
```

```
# set projection info (epsg code of Lambert 93)
lidR::projection(point_cloud) <- 2154
# summary of object content
point_cloud
```

```
## class      : LAS (v1.1 format 1)
## memory     : 304.8 Mb
## extent     : 899500, 9e+05, 6447500, 6448000 (xmin, xmax, ymin, ymax)
## coord. ref. : RGF93 / Lambert-93
## area       : 249988.5 m2
## points     : 4 million points
## density    : 15.98 points/m2
```

Metadata

The header slot contains information about the file version, data creation, presence of optional attributes, some point statistics, and offset/scale factors used internally to compute coordinates.

```
point_cloud@header

## File signature:      LASF
## File source ID:     0
## Global encoding:
## - GPS Time Type: GPS Week Time
## - Synthetic Return Numbers: no
## - Well Know Text: CRS is GeoTIFF
## - Aggregate Model: false
## Project ID - GUID:   00000000-0000-0000-0000-000000000000
## Version:             1.1
## System identifier:
## Generating software:  TerraScan
## File creation d/y:   265/2011
## header size:         227
## Offset to point data: 227
## Num. var. length record: 0
## Point data format:   1
## Point data record length: 28
## Num. of point records: 3995413
## Num. of points by return: 2629153 1091181 238951 32893 3235
## Scale factor X Y Z:  0.01 0.01 0.01
## Offset X Y Z:        0 0 0
## min X Y Z:           899500 6447500 1201.08
## max X Y Z:           9e+05 6448000 1322.58
## Variable Length Records (VLR):
##   Variable Length Record 1 of 1
##     Description: Geo Key Directory Tag
##     Tags:
##       Key 3072 value 2154
## Extended Variable Length Records (EVLR): void
```

Basic attributes

The `data` slot contains point attributes

X, Y, Z: coordinates. The point cloud can be displayed with `lidR::plot`

```
lidR::plot(point_cloud)
```

gpstime: time of emission of the pulse associated to the echo. Provided the precision is sufficient, it allows to retrieve echoes originating from the same pulse.

exemple of points with same gps time

```
point_cloud@data[point_cloud$gpstime == point_cloud$gpstime[38], 1:7]
```

```
##           X           Y           Z gpstime Intensity ReturnNumber
## 1: 899997.2 6448000 1255.96 32074.66         20             1
## 2: 899997.3 6448000 1255.34 32074.66         30             2
## 3: 899997.8 6447999 1252.52 32074.66         24             3
##      NumberOfReturns
## 1:                   3
## 2:                   3
## 3:                   3
```

Intensity: amplitude of signal peak associated to the echo. It is usually the raw value recorded by the receiver. In case of a diffuse, circular target entirely covered by the laser beam, the relationship between the transmitted (P_t) and received (P_r) laser power is (Baltsavias 1999):

$$P_r = \rho \frac{M^2 D_r^2 D_{tar}^2}{4R^2(R\gamma + D)^2} P_t$$

with:

- $\frac{\rho}{\pi}$ a bidirectional Lambertian reflection function
- M the atmospheric transmission
- D the aperture diameter of the laser emitter
- D_r diameter of receiving optics
- D_{tar} diameter of target object
- R range (distance between object and sensor)
- γ laser beam divergence

Assuming $D \ll R\gamma$, it can be simplified to:

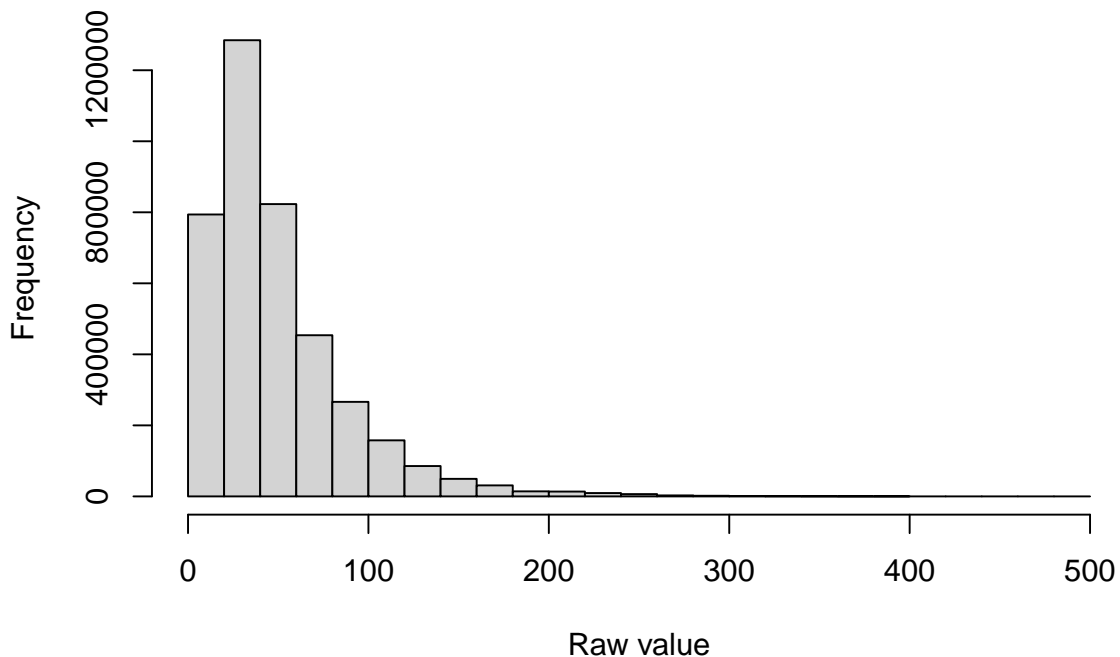
$$P_r = \rho D_{tar}^2 \times \frac{1}{R^4} \times M^2 \times \frac{P_t D_r^2}{\gamma^2} \times \frac{1}{4}$$

If one considers that the power transmitted by the emitter P_t is constant and that the distance to target R does not vary during the acquisition, then the amplitude is proportional to a combination of the geometric and radiometric properties of the target ρD_{tar}^2 . The case of multiple echoes for a single pulse is more complex.

In case the aircraft trajectory is available, it is possible to normalize amplitude values from the range (R) effect. One must however be aware that the resulting distribution of amplitude values might not be homogeneous because in areas with longer range the signal to noise ratio is lower.

```
hist(point_cloud@data$Intensity, xlab = "Raw value", main = "Histogram of Intensity")
```

Histogram of Intensity



Echo position within returns from the same pulse: ReturnNumber is the order of arrival of the echo among the NumberOfReturns echoes associated to a given pulse. Echoes are sometimes referred to as:

- single if NumberOfReturns = ReturnNumber = 1
- first if ReturnNumber = 1
- last if NumberOfReturns = ReturnNumber
- intermediate if $1 < \text{ReturnNumber} < \text{NumberOfReturns}$

The contingency table of ReturnNumber and NumberOfReturns should have no values below the diagonal, and approximately the same values in each column.

```
table(point_cloud@data$ReturnNumber, point_cloud@data$NumberOfReturns)
```

```
##
##      1      2      3      4      5
##  1 1538296 852356 205792 29508 3201
##  2      0 852521 205904 29552 3204
##  3      0      0 206176 29565 3210
##  4      0      0      0 29675 3218
##  5      0      0      0      0 3235
```

ScanDirectionFlag and **EdgeOfFlightline** indicate the scanning direction and if the scanner approached the edge of scan lines.

```
table(point_cloud@data$ScanDirectionFlag, point_cloud@data$EdgeOfFlightline)
```

```
##
##      1
##  0 3995413
```

Classification is usually obtained by an automated analysis of the point cloud geometry (e.g. Terrascan software uses the TIN adaptive algorithm by Axelsson (2000)) followed by manual validation and edition. The categories available in the classification are varying, the minimum being to identify a set of ground points. Classes are defined by numbers which should respect the ASPRS LAS specifications:

- 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.

Some specific fields might be added by the data provider during processing or for specific cases. In this file, some points were left with a classification value of 12.

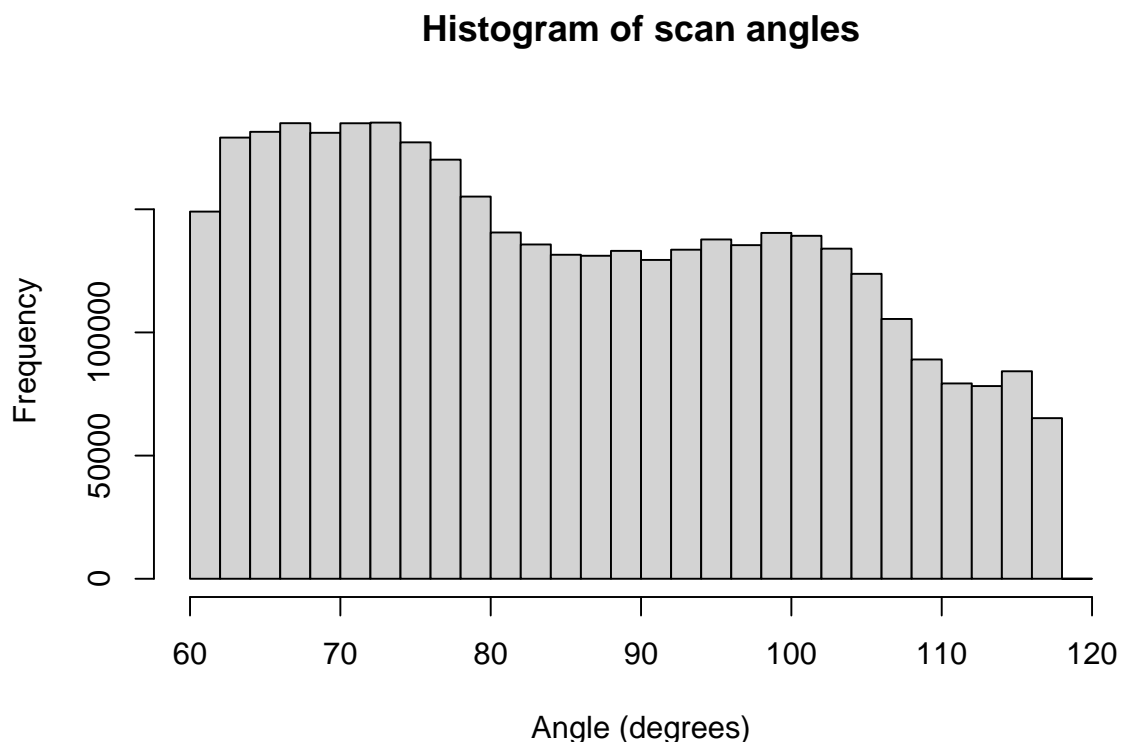
```
table(point_cloud@data$Classification)
```

```
##
##      2      4      12
## 477247 3004331 513835
```

Flags Synthetic, Keypoint, and Withheld indicate points which had a particular role in the classification process.

ScanAngleRank is the scan angle associated to the pulse. Origin of values might correspond to the horizontal or vertical. In most cases it is the scan angle relative to the laser scanner, but sometimes values relative to nadir are indicated. In the case of this file values are from 60 to 120: the scan range is ± 30 degrees on both sides of the scanner, with 90 the value when pulses are emitted downwards.

```
hist(point_cloud@data$ScanAngleRank, main = "Histogram of scan angles", xlab = "Angle (degrees)")
```



UserData is a field to store additional information, but it is limited in capacity. In latest version of LAS files, additional attributes can be added.

PointSourceID is usually a value indicating the flight strip number.

```
table(point_cloud@data$PointSourceID)
```

```
##
##      1710      1712      1713      1715      1716
## 1418803 479175  12460 1504000 580975
# boxplot(ScanAngleRank ~ PointSourceID, data = point_cloud@data)
```

Select and display a specific area

The `lidR` package has functions to extract an area of interest from a LAS object or a set of LAS files. Then the `lidR::plot` function can be used to display the point cloud, colored by different attributes.

```
# extract 15 meter radius disk at center of data
selection <- lidR::clip_circle(point_cloud, 899750, 6447750, 15)

rgl::par3d(mouseMode = "trackball") # parameters for interaction with mouse
# colored by height (default)
lidR::plot(selection)
# lidR::plot(selection, color = 'Intensity') lidR::plot(selection, color =
# 'Classification') lidR::plot(selection, color = 'ReturnNumber')
```

/tmp/RtmpXQnIXx/file475c3a99d5ab.png

Multiple files

Build catalog

ALS data from an acquisition are usually delivered in tiles, i.e. files which extents correspond to a division of the acquisition area in non-overlapping rectangles. The `lidR` package contains a catalog engine which enables to consider a set of files simultaneously by referencing them in a catalog object. Functions can then be applied to the whole dataset, or after extraction of subsets based on location or attributes.

```
# build catalog from folder
cata <- lidR::catalog("../data/aba.model/ALS/tiles.laz/")
# set projection
lidR::projection(cata) <- 2154
```

```
## Warning in showSRID(SRS_string, format = "PROJ", multiline
## = "NO", prefer_proj = prefer_proj): Discarded datum Reseau
## Geodesique Francais 1993 in Proj4 definition
```

```
cata
```

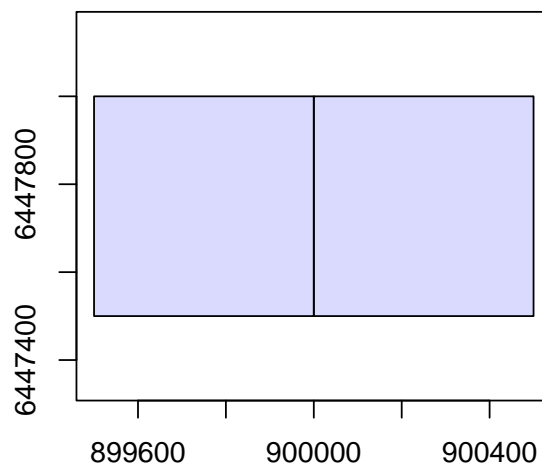
```
## class      : LAScatalog (v1.1 format 1)
## extent     : 899500, 900500, 6447500, 6448000 (xmin, xmax, ymin, ymax)
## coord. ref. : RGF93 / Lambert-93
## area       : 499980 m2
## points     : 8.81million points
## density    : 17.6 points/m2
## num. files : 2
```

```
# information about individual files in the data slot
cata@data
```

```
##   File.Signature File.Source.ID
## 1      LASF             0
## 2      LASF             0
##                               GUID Version.Major
## 1 00000000-0000-0000-0000-000000000000          1
## 2 00000000-0000-0000-0000-000000000000          1
##   Version.Minor System.Identifier Generating.Software
## 1              1                      TerraScan
```

```
## 2          1          TerraScan
## File.Creation.Day.of.Year File.Creation.Year Header.Size
## 1          265          2011          227
## 2          265          2011          227
## Offset.to.point.data Number.of.variable.length.records
## 1          227          0
## 2          227          0
## Point.Data.Format.ID Point.Data.Record.Length
## 1          1          28
## 2          1          28
## Number.of.point.records X.scale.factor Y.scale.factor
## 1          3995413          0.01          0.01
## 2          4814386          0.01          0.01
## Z.scale.factor X.offset Y.offset Z.offset Max.X Min.X
## 1          0.01          0          0          0 900000 899500
## 2          0.01          0          0          0 900500 900000
## Max.Y Min.Y Max.Z Min.Z CRS Number.of.1st.return
## 1 6448000 6447500 1322.58 1201.08 0 2629153
## 2 6448000 6447500 1284.99 1081.97 0 3354582
## Number.of.2nd.return Number.of.3rd.return
## 1          1091181          238951
## 2          1225378          212545
## Number.of.4th.return Number.of.5th.return
## 1          32893          3235
## 2          20851          1030
##                                     filename
## 1 ../data/aba.model/ALS/tiles.laz//899500_6447500.laz
## 2 ../data/aba.model/ALS/tiles.laz//900000_6447500.laz
```

```
# plot extent of files
lidR::plot(cata)
```



Checking specifications

For checking purposes one might be interested to map the following features, in order to estimate the potential of the data or to evaluate the acquisition with respect to the technical specifications:

- pulse density,
- point density,
- ground point density,
- strip overlap.

The spatial resolution of the map should be relevant with respect to the announced specifications, and

reach a trade-off between the amount of spatial detail and the possibility to evaluate the whole dataset at a glance.

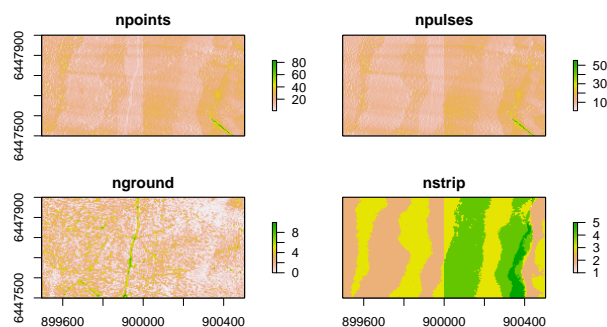
The `lidR::grid_metrics` function is very efficient to derive statistics maps from the point cloud. Point statistics are computed for all pixels at the given resolution, based on points located in the cells. Point density patterns are usually linked to variations in aircraft speed or strip overlap. In this case the density variation observed between the tiles is probably due to errors in processing.

```
# resolution
res <- 5
# build function to compute desired statistics
f <-
  function(rn, c, pid) { # function takes 3 inputs (ReturnNumber, Classification and PointId attributes)
    list(
      # number of points
      npoints = length(rn),
      # number of first points (proxy for pulse number)
      npulses = sum(rn == 1),
      # number of points of class ground
      nground = sum(c == 2),
      # number of unique values of strips
      nstrip = length(unique(pid))
    )
  }
# apply the function to the catalog, indicating which attributes to use, and output resolution
metrics <- lidR::grid_metrics(cata, ~f(ReturnNumber, Classification, PointSourceID), res = res)
```

```
## Warning: Invalid data: ScanAngleRank greater than 90 degrees
```

```
## Warning: Invalid data: ScanAngleRank greater than 90 degrees
```

```
# convert to density
for (i in c("npoints", "npulses", "nground")) {
  metrics[[i]] <- metrics[[i]] / res ^ 2
}
raster::plot(metrics)
```

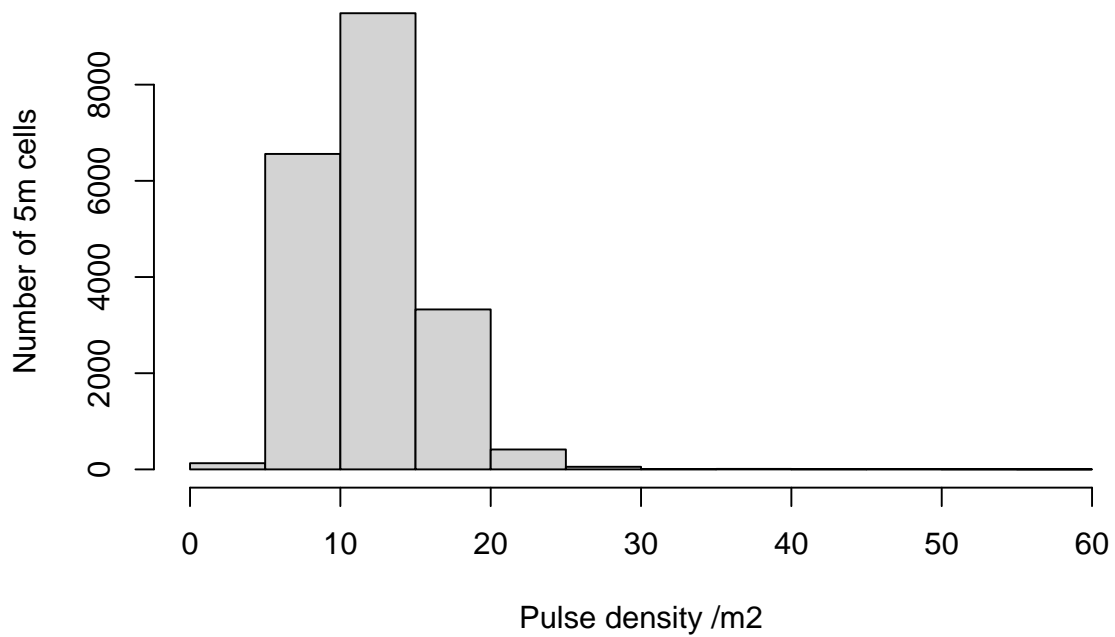


```
# percentage of 5 m cells with no ground points
p_no_ground <- round(sum(raster::values(metrics$nground)==0) / length(raster::values(metrics$nground)))
# percentage of 5 m cells with less than 5 pulses / m2
p_pulse_5 <- round(sum(raster::values(metrics$npulses)<10) / length(raster::values(metrics$npulses)))
```

From those maps summary statistics can be derived, e.g. the percentage of cells with no ground points (0.9). Checking that the spatial distribution of pulses is homogeneous can be informative.

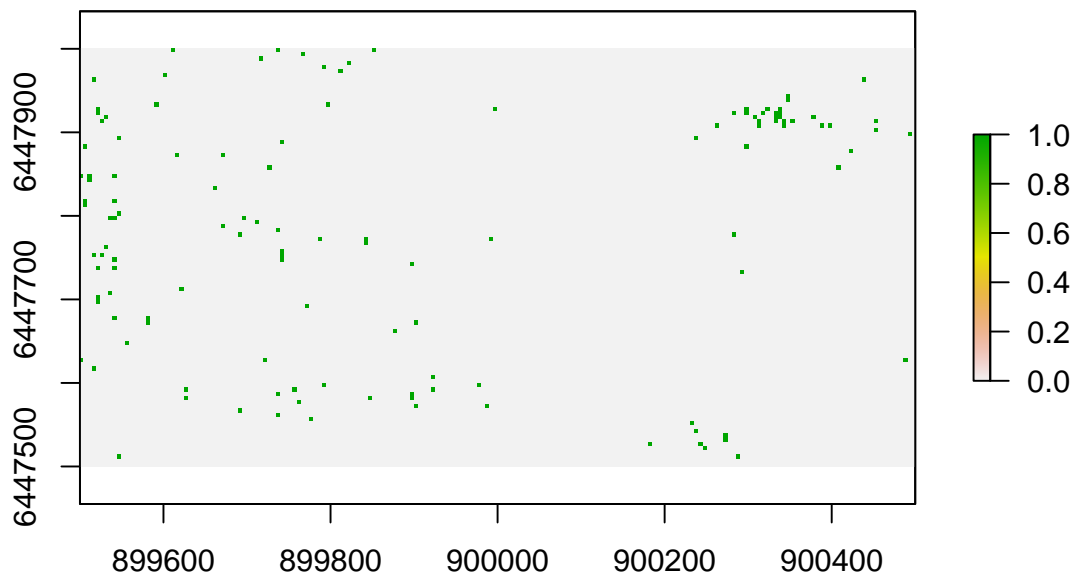
```
hist(raster::values(metrics$npulses), main = "Histogram of pulse density", xlab = "Pulse density /m2",
     ylab = "Number of 5m cells")
```


Histogram of pulse density



```
raster::plot(metrics$npulses < 5, main = "Areas with pulse density < 5 /m2")
```

Areas with pulse density < 5 /m2



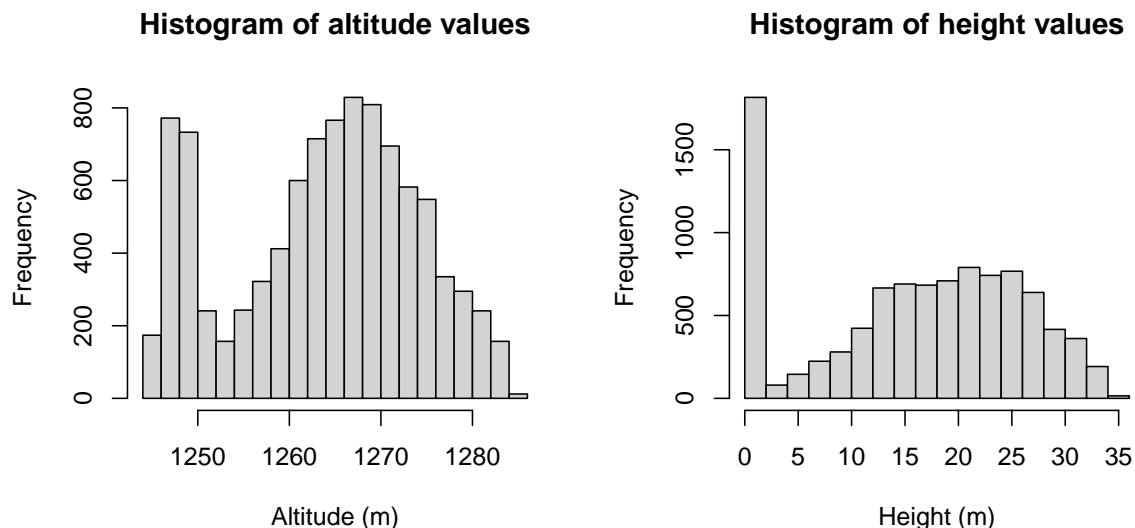
Data pre-processing

Point cloud normalization

Normalization of a point cloud refers to the operation that consists in replacing the altitude coordinate by the height to the ground in the Z attribute. This is an important pre-requisite for the analysis of the vegetation 3D structure from the point cloud. A pre-existing digital terrain model (DTM) can be used for normalizing, or it can be computed on-the-fly for this purpose. Obtaining relevant height values requires a DTM with sufficient resolution or the presence of enough ground points inside the data. To make sure that correct values are obtained at the border of the region of interest, it is advisable to add a buffer area before normalization.

The next line normalizes the point cloud using a TIN constructed from the points classified as ground (class 2). The height values are stored in the Z attribute. The altitude value is copied into a new attribute called **Zref**. When the LAS object is written to a LAS file, this field is lost, unless the file version allows the writing of additional fields.

```
# normalize with tin algorithm for computed of surface from ground points
selection_n <- lidR::normalize_height(selection, lidR::tin())
# lidR::plot(selection_n)
par(mfrow = c(1, 2))
hist(selection$Z, main = "Histogram of altitude values", xlab = "Altitude (m)")
hist(selection_n$Z, main = "Histogram of height values", xlab = "Height (m)")
```



Computation of digital elevation models

Digital elevation models (DEMs) are raster images where each cell value corresponds to the height or altitude of the earth surface. They are easier to display and use than point clouds but the information of the 3D structure of vegetation is mostly lost during computation. Three main types of DEMs can be computed from ALS point clouds.

Digital terrain model (DTM)

It represents the altitude of the bare Earth surface. It is computed using points classified as ground. For better representation of certain topographical features, some additional points or lines might be added. The function `lidR::grid_terrain` proposes different algorithms for computation, and works either with catalogs or LAS objects.

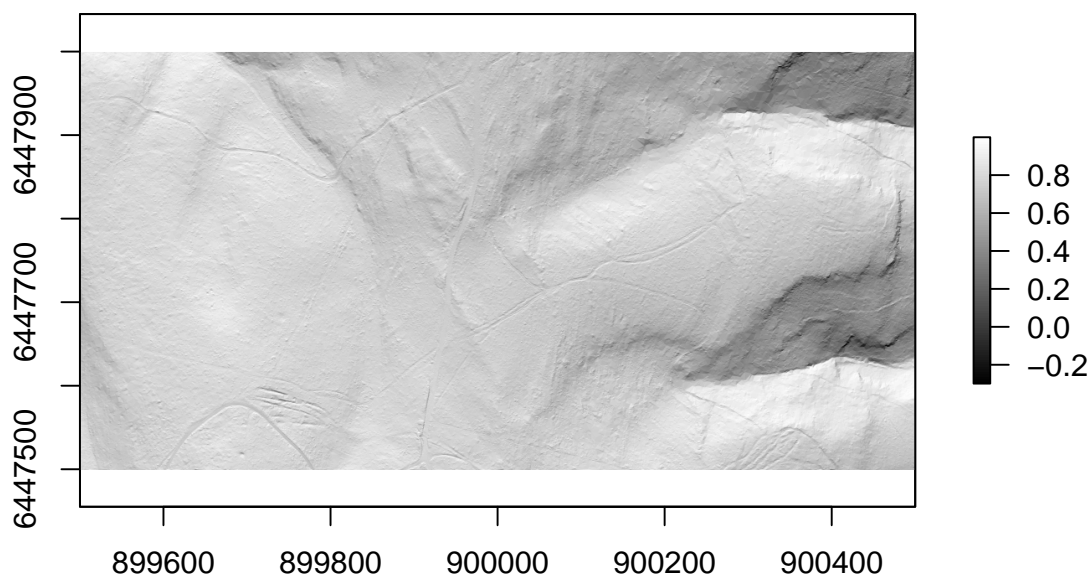
```
# create dtm at 0.5 m resolution with TIN algorithm
dtm <- lidR::grid_terrain(cata, res = 0.5, lidR::tin())
dtm
```

```
## class      : RasterLayer
## dimensions : 1000, 2000, 2e+06 (nrow, ncol, ncell)
## resolution : 0.5, 0.5 (x, y)
## extent     : 899500, 900500, 6447500, 6448000 (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source     : memory
## names      : Z
## values     : 1081.98, 1297.58 (min, max)
```

Functions from the **raster** package are available to derive topographical rasters (slope, aspect, hillshade, contour lines) from the DTM.

```
dtm_slope <- raster::terrain(dtm)
dtm_aspect <- raster::terrain(dtm, opt = "aspect")
dtm_hillshade <- raster::hillShade(dtm_slope, dtm_aspect)
raster::plot(dtm_hillshade, col = gray(seq(from = 0, to = 1, by = 0.01)), main = "Hillshade")
```

Hillshade



Digital surface model (DSM)

It represents the altitude of the Earth surface, including natural and man-made objects. It is computed using all points. The function `lidR::grid_canopy` proposes different algorithms for that purpose, and works either with catalogs or LAS objects. A straightforward way to compute the DSM is to retain the value of the highest point present in each pixel. In this case, choosing an adequate resolution with respect to the point density should limit the proportion of empty cells.

```
# create dsm at 0.5 m resolution with highest point per pixel algorithm
dsm <- lidR::grid_canopy(cata, res = 0.5, lidR::p2r())
dsm
```

```
## class      : RasterLayer
## dimensions : 1000, 2000, 2e+06 (nrow, ncol, ncell)
## resolution : 0.5, 0.5 (x, y)
## extent     : 899500, 900500, 6447500, 6448000 (xmin, xmax, ymin, ymax)
```

```
## crs      : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source   : memory
## names     : Z
## values    : 1084.83, 1322.58 (min, max)
```

Canopy Height Model (CHM)

It represents the height of objects of the Earth surface. It can be computed either by subtracting the DTM to the DSM, or by using the normalized point cloud as input in the DSM computation workflow (slightly more precise than previous option).

```
# create chm from dsm and dtm
```

```
chm <- dsm - dtm
chm
```

```
## class      : RasterLayer
## dimensions  : 1000, 2000, 2e+06 (nrow, ncol, ncell)
## resolution  : 0.5, 0.5 (x, y)
## extent     : 899500, 900500, 6447500, 6448000 (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source     : memory
## names      : layer
## values     : -0.82, 104.26 (min, max)
```

```
# create chm from normalized point cloud
```

```
cata_norm <- lidR::catalog("../data/aba.model/ALS/tiles.norm.laz/")
# set projection
chm_norm <- lidR::grid_canopy(cata_norm, res = 0.5, lidR::p2r())
chm_norm
```

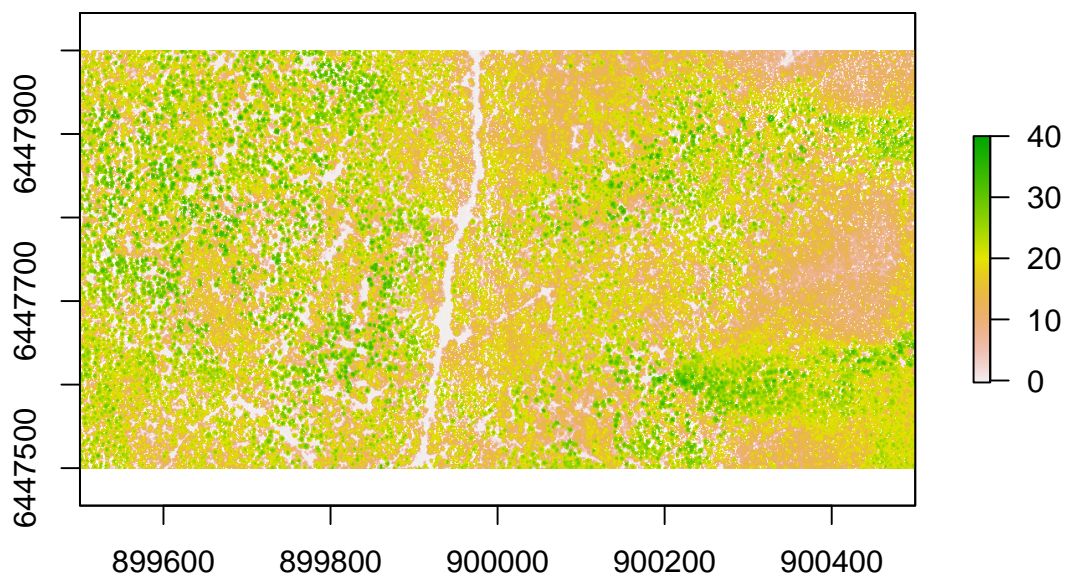
```
## class      : RasterLayer
## dimensions  : 1000, 2000, 2e+06 (nrow, ncol, ncell)
## resolution  : 0.5, 0.5 (x, y)
## extent     : 899500, 900500, 6447500, 6448000 (xmin, xmax, ymin, ymax)
## crs        : NA
## source     : memory
## names      : Z
## values     : -0.85, 104.22 (min, max)
```

```
# boxplot(raster::values(chm - chm_norm))
```

If high points are present in the dataset, it is sometimes useful to apply a threshold to the CHM values.

```
threshold <- 40
chm[chm > threshold] <- threshold
raster::plot(chm, main = "Canopy Height Model (m)")
```

Canopy Height Model (m)



Batch processing of files

To process multiple files and save outputs, the catalog engine can be used by specifying the output options.

```
# use parallelisation for faster processing
library(foreach)
# create parallel frontend, specify to use two parallel sessions
doFuture::registerDoFuture()
future::plan("multisession", workers = 2L)
# output resolution
res <- 0.5
# process by file
lidR::opt_chunk_size(cata) <- 0
# buffer size for DTM computation and normalization
lidR::opt_chunk_buffer(cata) <- 10
# DTM output file template
lidR::opt_output_files(cata) <- "dtm_{ORIGINALFILENAME}"
# create DTM
dtm <- lidR::grid_terrain(cata, 0.5, lidR::tin())
# laz compression
lidR::opt_laz_compression(cata) <- TRUE
# LAZ output file template
lidR::opt_output_files(cata) <- "norm_{ORIGINALFILENAME}"
# create normalized files
cata.norm <- lidR::normalize_height(cata, lidR::tin())
# buffer size for dsm
lidR::opt_chunk_buffer(cata) <- 0
# DSM output file template
lidR::opt_output_files(cata) <- "dsm_{ORIGINALFILENAME}"
# create DSM
```

```

dsm <- lidR::grid_canopy(cata, 0.5, lidR::p2r())
# create CHM from normalized files process by file
lidR::opt_chunk_size(cata_norm) <- 0
# buffer size
lidR::opt_chunk_buffer(cata_norm) <- 0
# CHM output file template
lidR::opt_output_files(cata_norm) <- "chm_{ORIGINALFILENAME}"
# create CHM
chm <- lidR::grid_canopy(cata_norm, 0.5, lidR::p2r())

```

References

- Axelsson, P. 2000. "DEM Generation from Laser Scanner Data Using Adaptive TIN Models." In *XIXth ISPRS Congress, IAPRS*, XXXIII:110–17.
- Baltsavias, E. P. 1999. "Airborne Laser Scanning: Basic Relations and Formulas." *ISPRS Journal of Photogrammetry and Remote Sensing* 54 (2): 199–214. [https://doi.org/10.1016/S0924-2716\(99\)00015-5](https://doi.org/10.1016/S0924-2716(99)00015-5).