

# ALS Processing Tutorial - Part I

Fabian Jörg Fischer

2025-12-16

## Table of contents

<b>1 Overview</b>	<b>2</b>
1.1 Objective . . . . .	2
1.2 General guidance . . . . .	2
1.3 Setup and Loading Packages . . . . .	3
<b>2 Sample Data: ALS Point Clouds from G-LiHT and NEON at Harvard Forest</b>	<b>3</b>
2.1 Study Site: Harvard Forest . . . . .	3
2.2 Data: Airborne Laser Scans . . . . .	5
<b>3 Guiding Question: What is the Typical Forest Structure at Harvard Forest?</b>	<b>5</b>
<b>4 Processing</b>	<b>6</b>
4.1 Reading in Point Cloud Data and Quality Checks . . . . .	6
4.1.1 Pulse Density Analysis . . . . .	15
4.2 Digital Terrain Model / Ground Classification . . . . .	17
4.3 Digital Surface Model and Canopy Height Model . . . . .	18
4.4 Processing of Entire ALS Collection . . . . .	22
4.5 Analysis: Forest Structure at Harvard Forest . . . . .	52
4.6 Tree-Based Analysis (Optional) . . . . .	53
4.6.1 Using an Existing Data Set . . . . .	53
4.6.2 Raster-Based Tree Segmentation . . . . .	54
<b>5 Summary and Key Takeaways</b>	<b>55</b>
<b>6 Data Quality</b>	<b>55</b>
<b>7 Processing Workflow</b>	<b>56</b>
<b>8 Tree Detection</b>	<b>56</b>

<b>9 Best Practices</b>	<b>56</b>
<b>10 Conclusion</b>	<b>56</b>
<b>11 Additional Resources</b>	<b>57</b>

# 1 Overview

This is an Notebook for a tutorial on ALS processing in R.

## 1.1 Objective

**i** What this aims to be

- a quick introduction
- overview over typical workflow
- overview over typical uncertainties and errors

**!** What this does not aim to be

- fully developed and robust processing workflow
- full analysis framework

## 1.2 General guidance

Useful processing maxims:

- prefer upper canopy data over within-canopy data
- prefer simple over complex metrics
- carry out sensitivity analyses (changes in parameters, changes in data)

Useful books/tutorials:

- Interactive introduction by NOAA: <https://coast.noaa.gov/digitalcoast/training/intro-lidar.html>
- Excellent overview by NEON: <https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics>
- Excellent tutorial on open source processing: <https://r-lidar.github.io/lidRbook/>
- Standard software tool / LASTools + documentation: <https://rapidlasso.de/product-overview/> + <https://groups.google.com/g/lastools>

- Surface models guidance: Fischer et al. 2024, Methods in Ecology and Evolution, <https://doi.org/10.1111/2041-210X.14416>
- 3D density processing: <https://amapvox.org/>

### Personal Coding Preferences

I have a few personal preferences for coding in R. These preferences will be visible in the scripts, but you do not have to follow them. In particular:

- no “tidyverse” coding (no pipes, no dplyr, etc.) except ggplot
- using data.table for non-spatial data manipulation
- using terra for spatial data manipulation
- no use of “`<-`” for assigning values, always using “`=`” operator

## 1.3 Setup and Loading Packages

```
# standard libraries for our purposes
library(lidR) # default lidar processing software in R, rasterizes point clouds
library(terra) # for analyzing rasterized products
library(data.table) # fast data manipulation
library(ggplot2) # plotting
library(viridis) # plotting
library(patchwork) # plotting
library(RCSF) # ground classification CSF
# library(RMCC) # ground classification MCC
library(future)
```

## 2 Sample Data: ALS Point Clouds from G-LiHT and NEON at Harvard Forest

Throughout this document, we will focus on 8 square kilometres of forest in the U.S. The data are available [here](#). Unzip the dowloaded folder into your working directory and keep the directory structure as it is.

### 2.1 Study Site: Harvard Forest

Located in the Worcester ecoregion, vegetation is consistent with those of transition hardwoods-white pine-hemlock forests. Due to historical logging and hurricanes, the forest is second-growth

and many of the trees are under 100 years old. The dominant vegetation is regenerating Eastern Deciduous temperate forest comprised of red oak (*Quercus rubra*), red maple (*Acer rubrum*), and white pine (*Pinus strobus*). Understory shrubs, trees, ferns, and flowering herbs are common in areas with higher moisture.



Figure 1: Harvard Forest canopy view

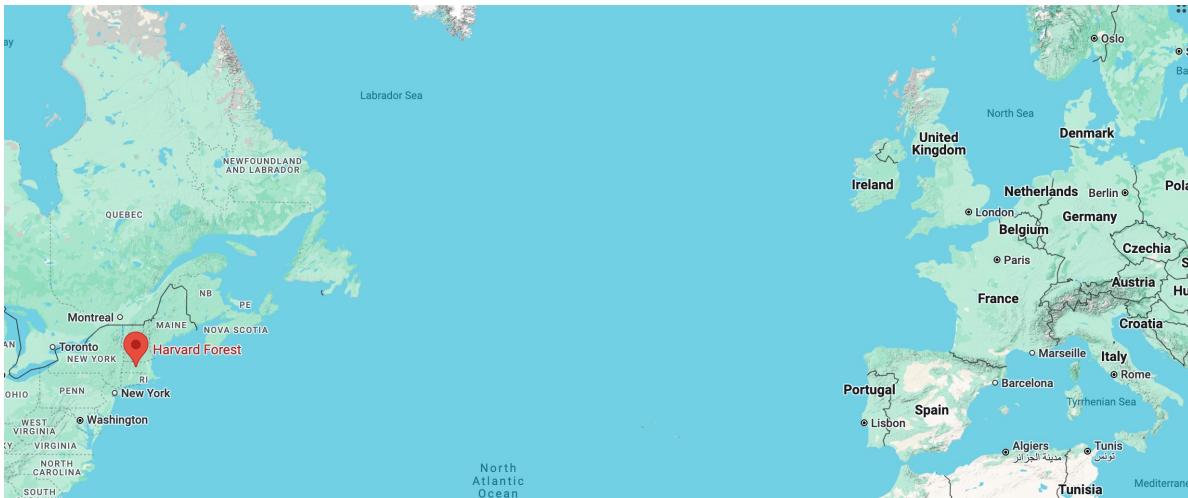


Figure 2: Harvard Forest location

#### Data Source

Directly taken from: <https://www.neonscience.org/field-sites/harv>

**View from above:**



## 2.2 Data: Airborne Laser Scans

Harvard Forest is probably one of the most-scanned sites. Some parts of it have been scanned with airborne lasers 12 times between 2012 and 2024, and these are just the openly available ones.

There are three types of scanning agencies:

- **NEON**: National Ecological Observatory Network - continuous surveys of ecologically interesting sites (8 scans, all summer)
- **GLiHT**: NASA laser scanning - mostly satellite validation (3 scans, all summer)
- **3DEP**: General U.S. national survey (1 scan, winter)

## 3 Guiding Question: What is the Typical Forest Structure at Harvard Forest?

### Motivation:

- relevance for biomass
- relevance for disturbance monitoring
- relevance for habitat/biodiversity assessments

## Questions:

- how do we process ALS point clouds to obtain estimates of height?
- what do you expect the typical tree height to be?

## 4 Processing

The different steps here are modelled on the lidR handbook (<https://r-lidar.github.io/lidRbook/>). For this exercise, we rely on an early NEON scan from 2012, as this is a lower quality scan and faster to process. However, for those that would like to compare two scans, there is also a GLiHT scan from 2012 in the folder.

### 4.1 Reading in Point Cloud Data and Quality Checks

When reading in point cloud data, there are several things to consider:

- **Data types:** .las, .laz, .asc, .xyz
- **Coordinate Reference Systems (CRS):** usually UTM coordinates
- **GPS times:** GPS week time vs. adjusted standard time
- **Data quality:** pulse density (more important than point density)

```
# read in a single tile and examine it.  
las = readLAS("data/NEON/2012/731000_4712000.laz")
```

```
[======>] 59% ETA: 1s  
[======>] 59% ETA: 1s  
[======>] 59% ETA: 1s  
[======>] 59% ETA: 1s  
[======>] 60% ETA: 1s  
[======>] 61% ETA: 1s  
[======>] 62% ETA: 1s  
[======>] 62% ETA: 1s  
[======>] 62% ETA: 1s
```

[=====>	] 62% ETA: 1s
[=====>	] 62% ETA: 1s
[=====>	] 63% ETA: 1s
[=====>	] 64% ETA: 1s
[=====>	] 65% ETA: 1s
[=====>	] 66% ETA: 1s
[=====>	] 67% ETA: 1s
[=====>	] 68% ETA: 1s
[=====>	] 69% ETA: 1s
[=====>	] 70% ETA: 0s
[=====>	] 70% ETA: 0s
[=====>	] 71% ETA: 0s

[=====>	] 71% ETA: 0s
[=====>	] 72% ETA: 0s
[=====>	] 73% ETA: 0s
[=====>	] 74% ETA: 0s
[=====>	] 75% ETA: 0s
[=====>	] 76% ETA: 0s
[=====>	] 76% ETA: 0s
[=====>	] 76% ETA: 0s
[=====>	] 77% ETA: 0s
[=====>	] 78% ETA: 0s
[=====>	] 79% ETA: 0s

[=====>	] 79% ETA: 0s
[=====>	] 80% ETA: 0s
[=====>	] 81% ETA: 0s
[=====>	] 82% ETA: 0s
[=====>	] 83% ETA: 0s
[=====>	] 84% ETA: 0s
[=====>	] 85% ETA: 0s
[=====>	] 86% ETA: 0s
[=====>	] 87% ETA: 0s
[=====>	] 88% ETA: 0s
[=====>	] 88% ETA: 0s

```
[======> ] 88% ETA: 0s
[======> ] 88% ETA: 0s
[======> ] 88% ETA: 0s
[======> ] 89% ETA: 0s
[======> ] 90% ETA: 0s
[======> ] 91% ETA: 0s
[======> ] 92% ETA: 0s
[======> ] 93% ETA: 0s
[======> ] 94% ETA: 0s
[======> ] 95% ETA: 0s
[======> ] 96% ETA: 0s
```

```
[=====] 97% ETA: 0s
[=====] 98% ETA: 0s
[=====] 99% ETA: 0s
```

```
# clip the tile to a smaller extent for quicker processing
las = clip_rectangle(las, 731400, 4712175, 731700, 4712475)

# check sumstats
print(las)
```

```
class      : LAS (v1.3 format 1)
memory    : 36.2 Mb
extent    : 731400, 731700, 4712175, 4712475 (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 18N
area       : 90600 m2
points     : 527.4 thousand points
type       : airborne
density    : 5.82 points/m2
density    : 2.74 pulses/m2
```

```
summary(las)
```

```
class      : LAS (v1.3 format 1)
memory    : 36.2 Mb
extent    : 731400, 731700, 4712175, 4712475 (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 18N
area       : 90600 m2
points     : 527.4 thousand points
```

```

type      : airborne
density   : 5.82 points/m2
density   : 2.74 pulses/m2
File signature:      LASF
File source ID:     0
Global encoding:
- GPS Time Type: GPS Week Time
- Synthetic Return Numbers: no
- Well Known Text: CRS is GeoTIFF
- Aggregate Model: false
Project ID - GUID: 00000000-0000-0000-000000000000
Version:            1.3
System identifier: LAStools (c) by rapidlasso GmbH
Generating software: lasmerge (version 160730)
File creation d/y: 46/2018
header size:       235
Offset to point data: 575
Num. var. length record: 2
Point data format: 1
Point data record length: 32
Num. of point records: 527426
Num. of points by return: 248117 174886 80035 24388 0
Scale factor X Y Z: 0.01 0.01 0.01
Offset X Y Z:        7e+05 4700000 0
min X Y Z:          731400 4712175 339.77
max X Y Z:          731700 4712475 383.41
Variable Length Records (VLR):
  Variable Length Record 1 of 2
    Description:
    Tags:
      Key 1024 value 1
      Key 1025 value 2
      Key 3072 value 32618
      Key 4099 value 9001
  Variable Length Record 2 of 2
    Description: by LAStools of rapidlasso GmbH
    Extra Bytes Description:
      reversible index (lastile): original index before tiling
  Extended Variable Length Records (EVLR): void

```

```

# check the underlying structure (data.table)
las_dt = las@data

```

```

class(las_dt)

[1] "data.table" "data.frame"

str(las_dt)

Classes 'data.table' and 'data.frame': 527426 obs. of 17 variables:
 $ X                  : num 731682 731683 731684 731686 731686 ...
 $ Y                  : num 4712175 4712176 4712176 4712176 4712176 ...
 $ Z                  : num 355 352 352 352 356 ...
 $ gpstime            : num 217776 217776 217776 217776 217776 ...
 $ Intensity           : int 1 1 2 19 18 20 10 17 1 5 ...
 $ ReturnNumber         : int 2 3 2 3 2 3 2 3 2 2 ...
 $ NumberOfReturns      : int 3 3 2 3 3 3 3 3 2 2 ...
 $ ScanDirectionFlag    : int 0 0 0 0 0 0 0 0 0 0 ...
 $ EdgeOfFlightline     : int 0 0 0 0 0 0 0 0 0 0 ...
 $ Classification        : int 5 2 2 2 5 2 5 2 2 5 ...
 $ Synthetic_flag        : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ Keypoint_flag         : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ Withheld_flag          : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ ScanAngleRank         : int -20 -20 -20 -20 -20 -20 -20 -20 -20 -
21 ...
 $ UserData              : int 33 0 0 0 31 0 37 0 0 127 ...
 $ PointSourceID          : int 2 2 2 2 2 2 2 2 2 2 ...
$ reversible index (lastile): int 756865478 756865479 756865481 756865485 756865487 756865487 756865487
- attr(*, ".internal.selfref")=<externalptr>

```

We can visualize the point cloud:

```

# visualize the point cloud
# this may not work, especially for large point clouds
plot(las)

```

We can automatically check the quality of the files:

```

# check quality
las_check(las)

```

Checking the data

- Checking coordinates...
- Checking coordinates type...
- Checking coordinates range...
- Checking coordinates quantization...
- Checking attributes type...
- Checking ReturnNumber validity...
- Checking NumberOfReturns validity...
- Checking ReturnNumber vs. NumberOfReturns...
- Checking RGB validity...
- Checking absence of NAs...
- Checking duplicated points...

14 points are duplicated and share XYZ coordinates with other points

- Checking degenerated ground points...

There were 12 degenerated ground points. Some X Y coordinates were repeated but with di

- Checking attribute population...
- Checking gpstime incoherances
- Checking flag attributes...
- Checking user data attribute...

480222 points have a non 0 UserData attribute. This probably has a meaning

Checking the header

- Checking header completeness...
- Checking scale factor validity...
- Checking point data format ID validity...
- Checking extra bytes attributes validity...
- Checking the bounding box validity...
- Checking coordinate reference system...

Checking header vs data adequacy

- Checking attributes vs. point format...
- Checking header bbox vs. actual content...
- Checking header number of points vs. actual content...
- Checking header return number vs. actual content...

Checking coordinate reference system...

- Checking if the CRS was understood by R...

Checking preprocessing already done

- Checking ground classification... yes
- Checking normalization... no
- Checking negative outliers...
- Checking flightline classification... yes

Checking compression

- Checking attribute compression...
- Synthetic\_flag is compressed
- Keypoint\_flag is compressed

- Withheld\_flag is compressed

### Common Warnings

You will notice a few warnings. Duplicate and degenerate points are normal and usually occur when the laser scanner samples densely, but the resolution of the coordinates is low (rounding of digits). We often remove true duplicates, but it's not super important.

More worrying are:

- outliers
- missing coordinate reference systems (CRS)
- inconsistencies in return numbers (may indicate missing points)

#### 4.1.1 Pulse Density Analysis

We can create simple maps that evaluate the scan quality. For example, we can get an estimate of the pulse density of scans. This is the shots per unit area (usually reported per sqm), and indicates how well an area has been sampled. This is usually calculated as the density of first or last returns per square metre, because there is only one first and one last return for each pulse.

```
# evaluate pulse density
# set desired resolution of rasterization
res_pd = 5
pd = grid_metrics(las, ~sum(ReturnNumber == 1), res = res_pd) # calculate the sum of first returns
pd = pd / (res_pd * res_pd) # normalize by area (pd per sqm)
plot(pd, col = viridis(100, option = "turbo"), main = "Pulse density")
```

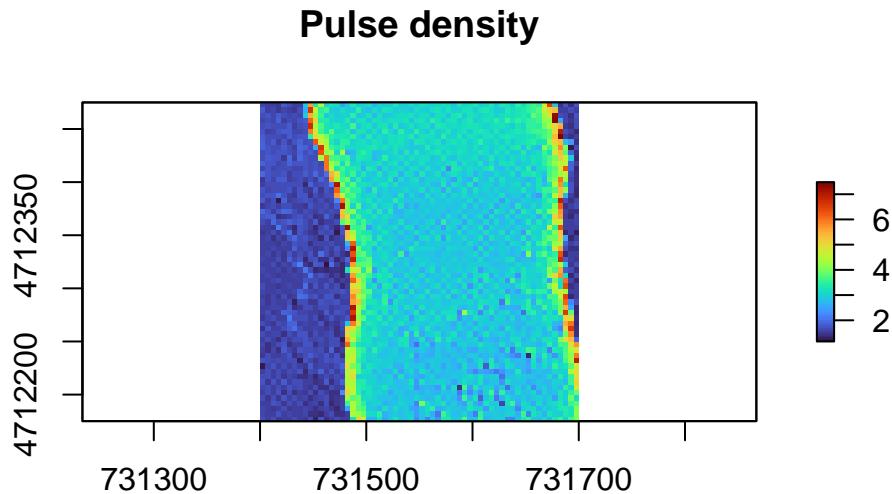


Figure 3: Pulse density map showing variation across the scan area

#### *i* Pulse Density Patterns

Notice how pulse density can vary strongly within scans - airplanes fly back and forth and some areas are sampled twice (much higher pulse density in the centre of the map), while areas underneath the plane may only be sampled once (left and right side).

Based on our own research:

- Areas with pulse densities  $< 2$  are not sufficiently sampled for high resolution analysis ( $\sim 1\text{m}$ )
- Pulse densities  $> 5$  are generally recommended
- $> 10\text{-}15$  in very dense forests
- If scanners have a low ground penetration rate (most pulses only generate one return), higher pulse densities may be necessary

You can also check out the scan angle. We plot the absolute scan angle to identify areas where points are mostly scanned from oblique angles.

```
# evaluate scan angle
# set desired resolution of rasterization
scanangle = grid_metrics(las, ~abs(mean(ScanAngleRank)), res = 5)
plot(scanangle, col = viridis(100, option = "turbo"), main = "Absolute scan angle")
```

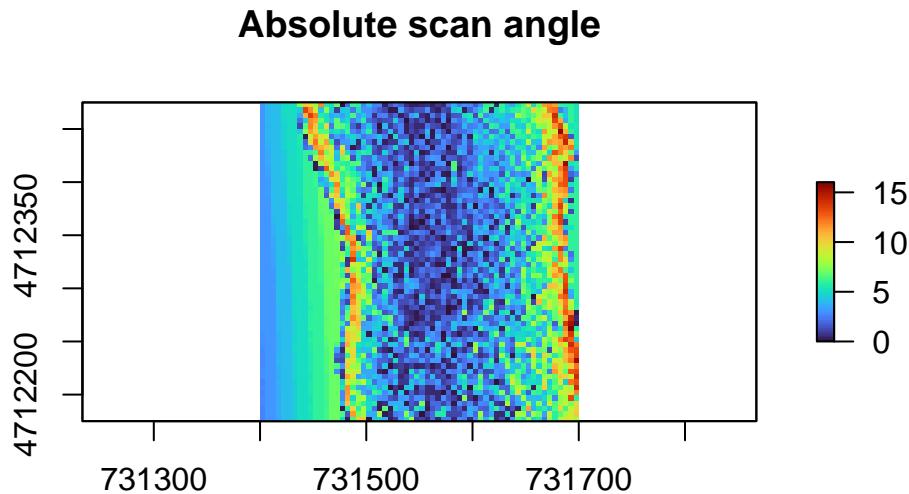


Figure 4: Absolute scan angle map

## 4.2 Digital Terrain Model / Ground Classification

A key requirement of most ALS-based analysis is to derive a high-resolution model of the terrain, a so-called digital terrain model (DTM).

### DTM Recommendations

A few recommendations:

- use default classification provided by companies / agencies that acquired data
- if you need to classify yourself, use commercial software
- if no access to commercial software, use Cloth Simulation Filter (CSF)
- evaluate uncertainties

First, we check ground classification for a single tile:

```
# check classifications
las@data[,.N,Classification] # ground classification is 2, noise is 7
```

Classification	N
2	1000000
7	1000000

```

<int> <int>
1:      5 457150
2:      2 46679
3:      1 23590
4:      7     7

```

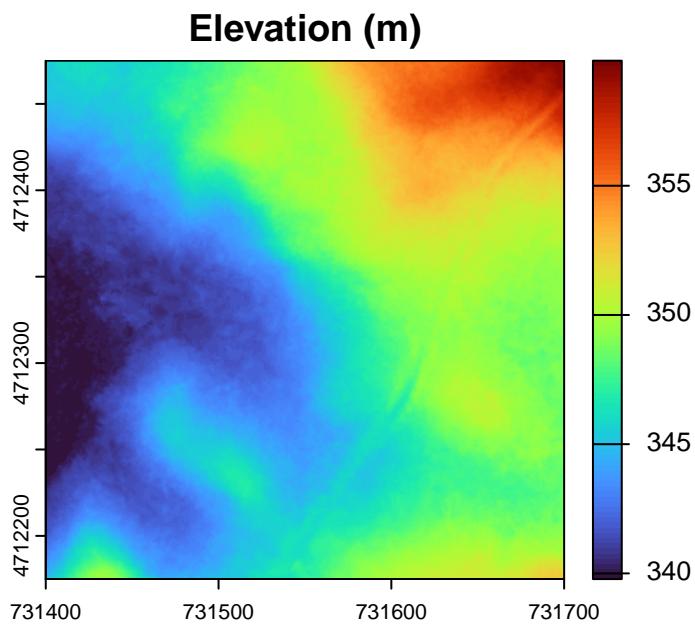
```
plot(las, color = "Classification", size = 3, bg = "white")
```

Then, we derive a digital terrain model for the tile:

```

# derive the DTM
dtm = rasterize_terrain(las, res = 1, algorithm = tin(), use_class = 2L)
plot(dtm, col = viridis(100, option = "turbo"), main = "Elevation (m)")

```



### 4.3 Digital Surface Model and Canopy Height Model

A canopy height model (CHM) shows the height of the vegetation above the ground. It is calculated as the difference between a digital surface model (DSM, height of the canopy) and the digital terrain model (DTM, height of the ground).

```

# remove noise (class 7 and 18)
las_nonoise = filter_poi(las, Classification != 7L & Classification != 18L)

# create CHM (use highest point per grid cell)
dsm_highest = rasterize_canopy(las_nonoise, res = 1, algorithm = p2r())
chm_highest = dsm_highest - dtm
plot(chm_highest, main = "CHM (highest point)")

```

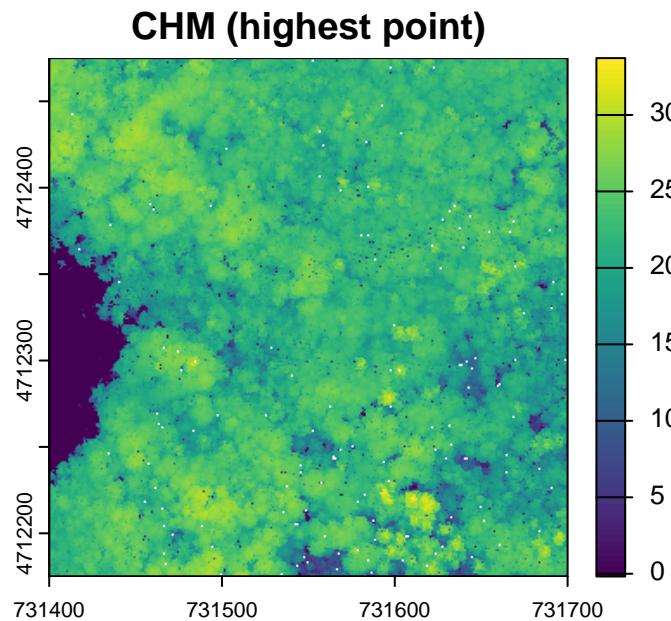


Figure 5: Canopy Height Model using highest point method

### ⚠ CHM Limitations

Notice that there are a few NA values and a few small “holes” in the CHM. This happens often with “highest point” CHMs when the scan quality is low. They are generally susceptible to differences in pulse density and should be avoided for comparisons across scans.

Now we create another CHM, but interpolate the canopy with a so-called Delaunay-triangulation (create a mesh of triangles between the nearest points in xy space). When applied only to the first returns, this is a robust (albeit not always aesthetically pleasing) way of inferring canopy height.

### TIN Preprocessing

Note that lidR by default carries out a pre-selection of “highest” points before the interpolation. In our experience, this makes the CHM less robust, so it needs to be specifically deactivated.

```
# create a more robust CHM by using only first returns
las_nonoise_first = filter_first(las_nonoise)
dsm_tin = rasterize_canopy(las_nonoise_first, res = 1, algorithm = dsmtin(highest = FALSE))
chm_tin = dsm_tin - dtm
plot(chm_tin, main = "CHM (tin)")
```

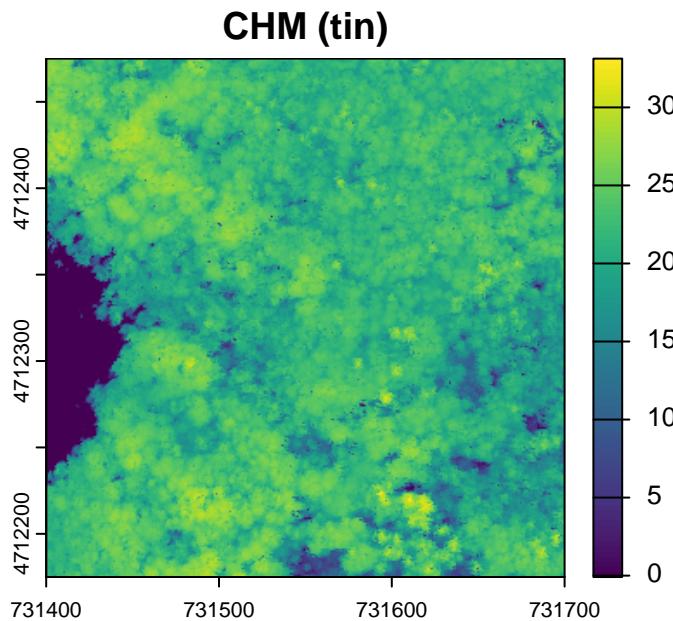


Figure 6: Canopy Height Model using TIN method

Now plot the difference between the two CHMs:

```
# compare the differences between the two CHMs
chm_diff = chm_highest - chm_tin
(mindiff = as.numeric(global(chm_diff, "min", na.rm = T)))
```

[1] -27.055

```
(maxdiff = as.numeric(global(chm_diff, "max", na.rm = T)))
```

```
[1] 22.58
```

```
(meandiff = as.numeric(global(chm_diff, "mean", na.rm = T)))
```

```
[1] 0.4274639
```

```
plot(chm_diff, col = viridis(100, option = "turbo"), main = "Canopy height difference (m)")
```

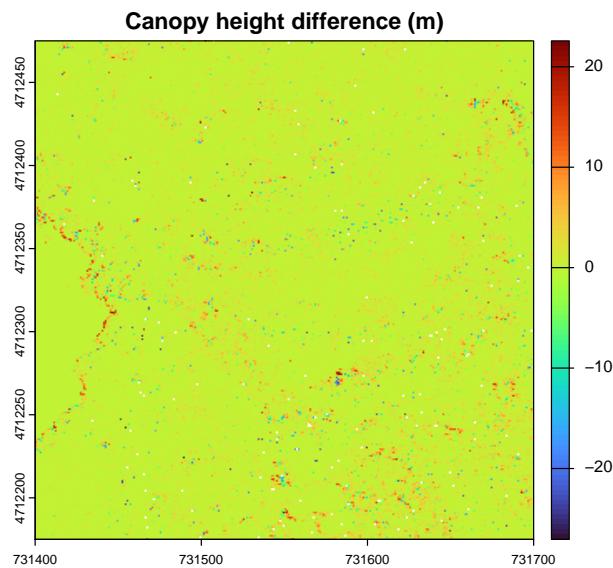


Figure 7: Comparison of CHM methods

```
hist(chm_diff, main = "Canopy height difference (m)", breaks = seq(floor(mindiff), ceiling(ma
```

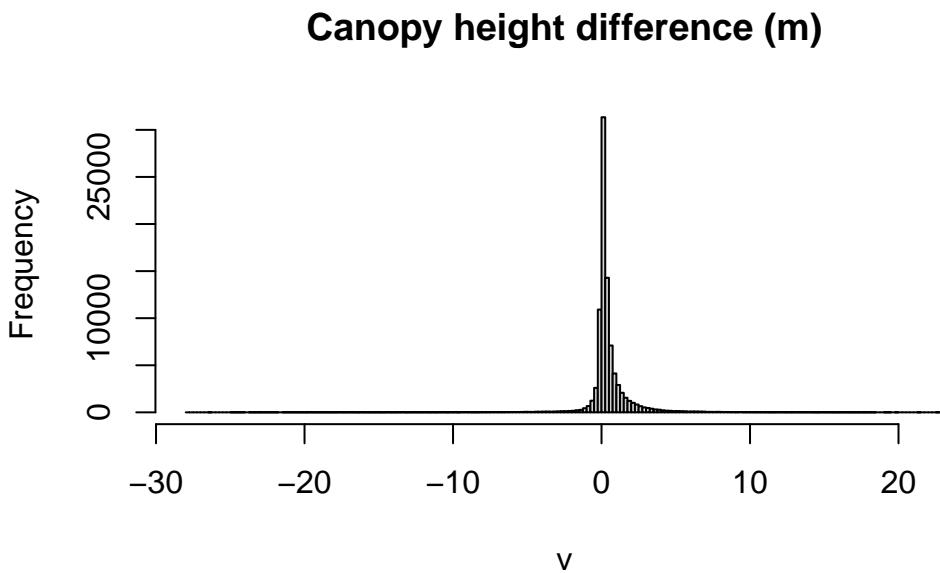


Figure 8: Distribution of CHM differences

#### 4.4 Processing of Entire ALS Collection

Usually, we do not just want to process data with a single small point cloud, but large areas (e.g., many square kilometres). In this case, we can read the tiles in all together as LAScatalog:

```
# read in a full catalog of LAS files
las_ctg = readLAScatalog("data/NEON/2012")
```

```
# evaluate summary statistics
print(las_ctg)
```

```
class      : LAScatalog (v1.3 format 1)
extent     : 731000, 733000, 4712000, 4716000 (xmin, xmax, ymin, ymax)
coord. ref.: WGS 84 / UTM zone 18N
area       : 8 km2
points     : 37.88 million points
type       : airborne
density    : 4.7 points/m2
density    : 2.4 pulses/m2
num. files : 8
```

```
summary(las_ctg)
```

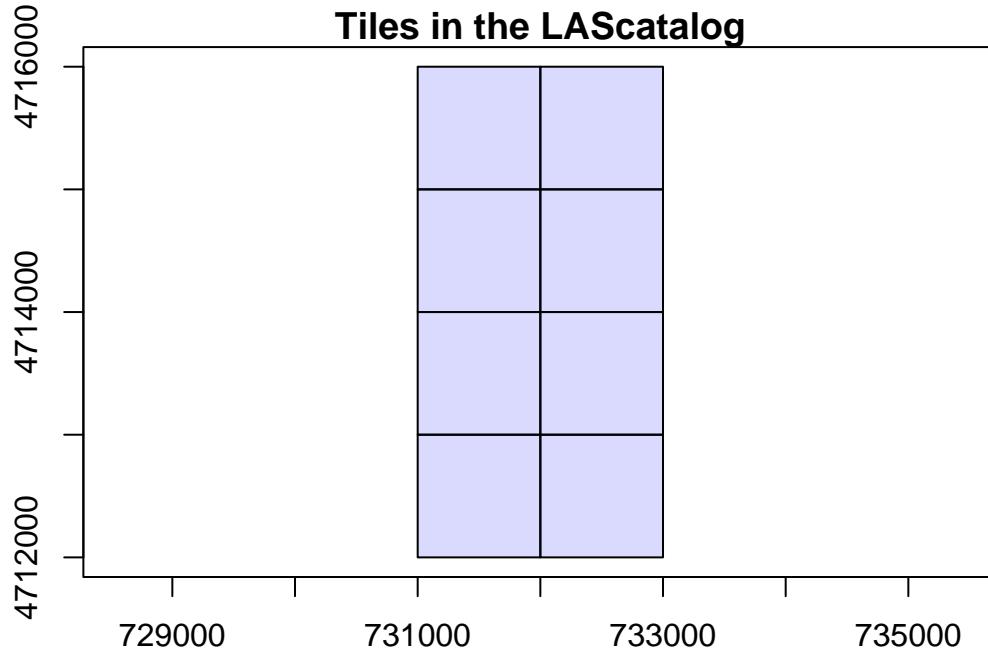
```
class      : LAScatalog (v1.3 format 1)
extent     : 731000, 733000, 4712000, 4716000 (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 18N
area       : 8 km2
points     : 37.88 million points
type       : airborne
density    : 4.7 points/m2
density    : 2.4 pulses/m2
num. files : 8
proc. opt.  : buffer: 30 | chunk: 0
input opt.  : select: * | filter:
output opt. : in memory | w2w guaranteed | merging enabled
drivers    :
- Raster : format = GTiff NAflag = -999999
- stars : NA_value = -999999
- Spatial : overwrite = FALSE
- SpatRaster : overwrite = FALSE NAflag = -999999
- SpatVector : overwrite = FALSE
- LAS : no parameter
- sf : quiet = TRUE
- data.frame : no parameter
```

```
las_check(las_ctg)
```

```
Checking headers consistency
- Checking file version consistency...
- Checking scale consistency...
- Checking offset consistency...
- Checking point type consistency...
- Checking VLR consistency...
- Checking CRS consistency...
Checking the headers
- Checking scale factor validity...
- Checking Point Data Format ID validity...
Checking preprocessing already done
- Checking negative outliers...
  8 file(s) with points below 0
- Checking normalization... no
Checking the geometry
```

- Checking overlapping tiles...
- Checking point indexation... no

```
plot(las_ctg, main = "Tiles in the LAScatalog")
```

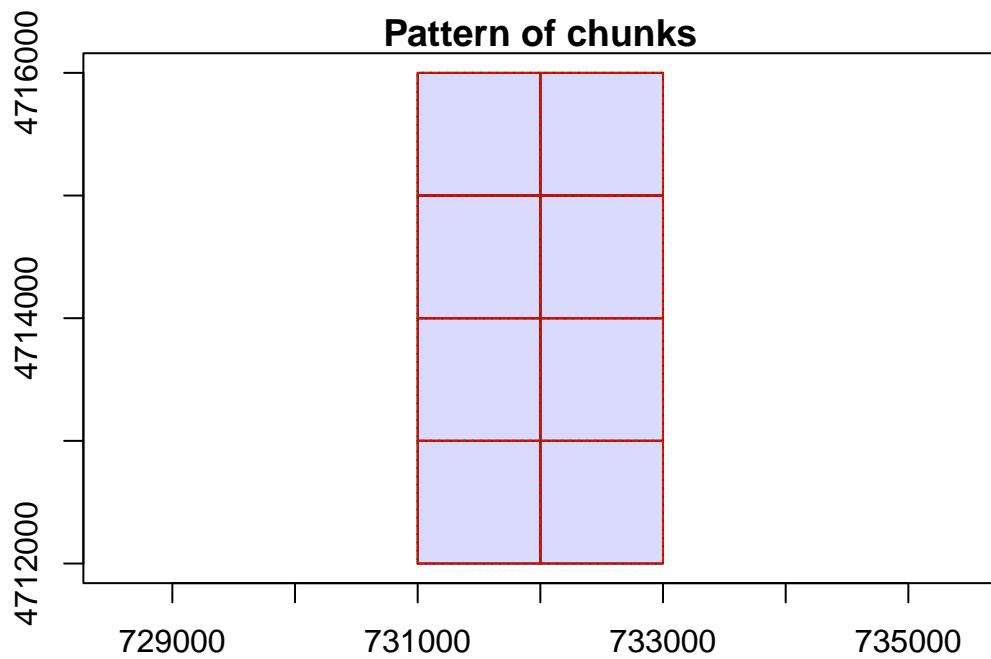


We can plot pulse density for the entire scan area. Note that this will take a bit of time on your computer. To speed it up, we use parallel processing.

```
# rasterize pulse density
# we pick a coarser resolution than before
res_pd = 20

# we process on 4 cores
# Set up a parallel plan (adjust number of workers as needed)
plan(multisession, workers = 4) # if it creates problems, deactivate

pd = grid_metrics(las_ctg, ~sum(ReturnNumber == 1), res = res_pd) # calculate the sum of first
```



Chunk 1 of 8 (12.5%): state

[=====>	] 25% ETA: 6s
[=====>	] 26% ETA: 8s

```
[======>] 26% ETA: 11s  
Chunk 2 of 8 (25%): state  
[======>] 15% ETA: 10s  
[======>] 15% ETA: 10s  
[======>] 15% ETA: 10s  
[======>] 15% ETA: 10s  
[======>] 16% ETA: 10s  
[======>] 17% ETA: 10s
```

```
[=====]> ] 17% ETA: 10s
[=====]> ] 17% ETA: 18s
[=====]> ] 17% ETA: 24s
[=====]> ] 17% ETA: 38s
```

Chunk 3 of 8 (37.5%): state

```
[=====]> ] 15% ETA: 11s
[=====]> ] 15% ETA: 10s
[=====]> ] 15% ETA: 10s
[=====]> ] 15% ETA: 10s
[=====]> ] 16% ETA: 10s
```



[=====>	] 17% ETA: 10s
[=====>	] 18% ETA: 15s
[=====>	] 18% ETA: 22s
[=====>	] 18% ETA: 35s
[=====>	] 18% ETA: 38s

Chunk 4 of 8 (50%): state

[=====>	] 22% ETA: 6s
[=====>	] 23% ETA: 6s
[=====>	] 24% ETA: 6s

[=====>	] 24% ETA: 6s
[=====>	] 25% ETA: 6s
[=====>	] 26% ETA: 11s
[=====>	] 26% ETA: 18s

Chunk 5 of 8 (62.5%): state

[=====>	] 22% ETA: 6s
---------	---------------



```
[======> ] 24% ETA: 6s  
[======> ] 24% ETA: 10s  
[======> ] 24% ETA: 18s
```

Chunk 6 of 8 (75%): state

```
[======> ] 14% ETA: 11s  
[======> ] 15% ETA: 30s  
[======> ] 15% ETA: 46s  
[======> ] 15% ETA: 52s
```

Chunk 8 of 8 (87.5%): state

```
[======> ] 22% ETA: 18s  
[======> ] 22% ETA: 24s
```

Chunk 7 of 8 (100%): state

```
[======> ] 15% ETA: 11s  
[======> ] 15% ETA: 11s
```

```
[=====]> ] 15% ETA: 11s
[=====]> ] 16% ETA: 11s
[=====]> ] 16% ETA: 28s
[=====]> ] 16% ETA: 46s
[=====]> ] 16% ETA: 49s
```

```
pd = pd / (res_pd * res_pd) # normalize by area (pd per sqm)
```

Plot the pulse density:

```
# plot it again
plot(pd, col = viridis(100, option = "turbo"), main = "Pulse density")
```

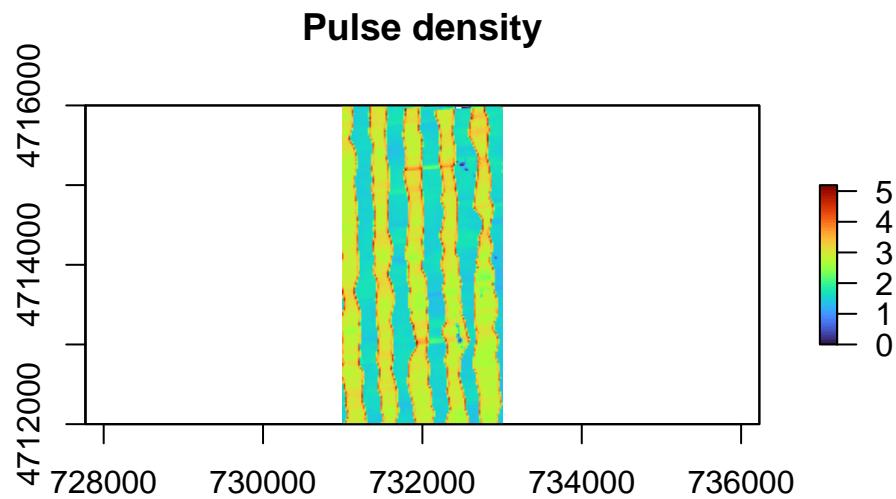
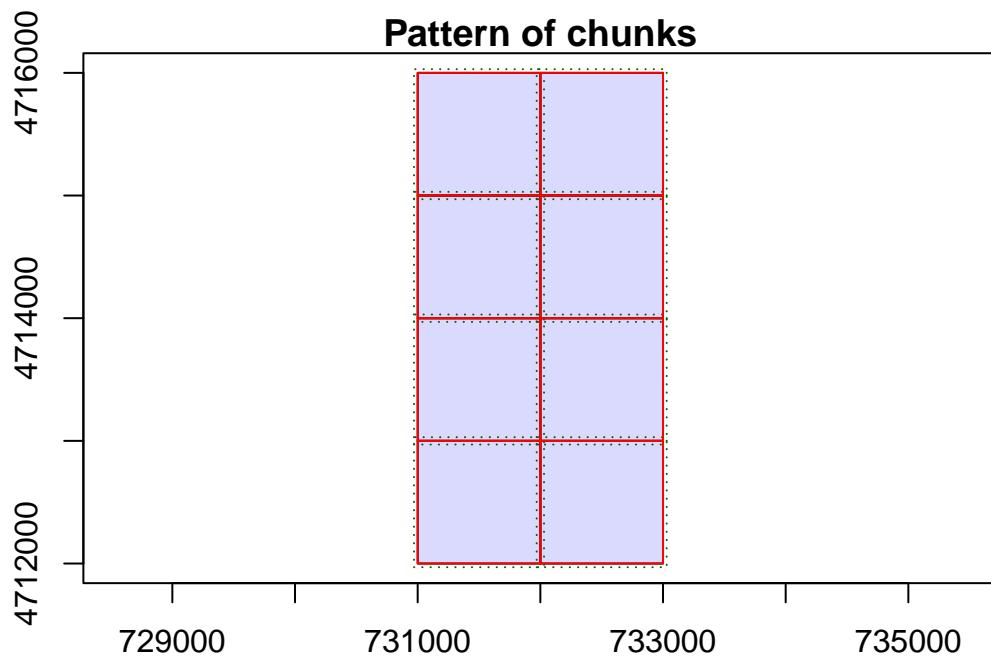


Figure 9: Pulse density across the entire study area

Now we rasterize a DTM for the entire area:

```
# derive digital terrain model
dtm_ctg = rasterize_terrain(las = las_ctg, res = 1, algorithm = tin(), use_class = 2L)
```



Chunk 1 of 8 (12.5%): state

[=> ] 3% ETA: 114s

Chunk 4 of 8 (25%): state

[=> ] 2% ETA: 108s  
[=> ] 2% ETA: 156s  
[=> ] 2% ETA: 236s

Chunk 2 of 8 (37.5%): state

[> ] 1% ETA: 222s  
[> ] 1% ETA: 288s  
[> ] 1% ETA: 455s

Chunk 3 of 8 (50%): state

[> ] 1% ETA: 130s  
[> ] 1% ETA: 240s  
[> ] 1% ETA: 301s  
[> ] 1% ETA: 463s  
[> ] 1% ETA: 460s

```
[> ] 1% ETA: 467s
```

Chunk 5 of 8 (62.5%): state

```
[=> ] 2% ETA: 252s  
[=> ] 2% ETA: 437s
```

Chunk 8 of 8 (75%): state

```
[=> ] 2% ETA: 74s  
[=> ] 2% ETA: 73s  
[=> ] 2% ETA: 73s  
[=> ] 2% ETA: 73s  
[=> ] 2% ETA: 72s  
[=> ] 2% ETA: 72s  
[=> ] 2% ETA: 72s  
[=> ] 3% ETA: 71s  
[=> ] 3% ETA: 70s  
[=> ] 3% ETA: 213s  
[=> ] 3% ETA: 233s  
[=> ] 3% ETA: 259s  
[=> ] 3% ETA: 289s
```

Chunk 6 of 8 (87.5%): state

```
[> ] 0% ETA: 238s  
[> ] 0% ETA: 237s  
[> ] 0% ETA: 233s  
[> ] 0% ETA: 229s  
[> ] 1% ETA: 225s  
[> ] 1% ETA: 221s  
[> ] 1% ETA: 217s  
[> ] 1% ETA: 213s  
[> ] 1% ETA: 209s  
[> ] 1% ETA: 205s  
[> ] 1% ETA: 202s
```

```
[> ] 1% ETA: 200s  
[> ] 1% ETA: 199s  
[> ] 1% ETA: 202s  
[> ] 1% ETA: 203s  
[> ] 1% ETA: 202s  
[> ] 1% ETA: 203s  
[> ] 1% ETA: 201s  
[> ] 1% ETA: 198s  
[> ] 1% ETA: 194s  
[> ] 1% ETA: 191s  
[> ] 1% ETA: 187s  
[> ] 1% ETA: 184s  
[> ] 1% ETA: 181s  
[> ] 1% ETA: 178s  
[> ] 1% ETA: 175s  
[> ] 1% ETA: 172s  
[> ] 1% ETA: 459s  
[> ] 1% ETA: 493s  
[> ] 1% ETA: 780s  
[> ] 1% ETA: 839s  
[> ] 1% ETA: 888s
```

Chunk 7 of 8 (100%): state

```
[> ] 1% ETA: 135s  
[> ] 1% ETA: 136s  
[> ] 1% ETA: 136s  
[> ] 1% ETA: 134s  
[> ] 1% ETA: 132s  
[> ] 1% ETA: 130s  
[> ] 1% ETA: 127s  
[> ] 1% ETA: 125s  
[> ] 1% ETA: 124s  
[> ] 1% ETA: 125s  
[=> ] 2% ETA: 125s  
[=> ] 2% ETA: 125s  
[=> ] 2% ETA: 126s  
[=> ] 2% ETA: 357s
```

```
[=> ] 2% ETA: 368s
[=> ] 2% ETA: 396s
[=> ] 2% ETA: 460s
[=> ] 2% ETA: 616s
[=> ] 2% ETA: 627s
[=> ] 2% ETA: 644s
```

Plot the DTM:

```
# plot DTM
plot(dtm_ctg, col = viridis(100, option = "turbo"), main = "Elevation (m, full area)")
```

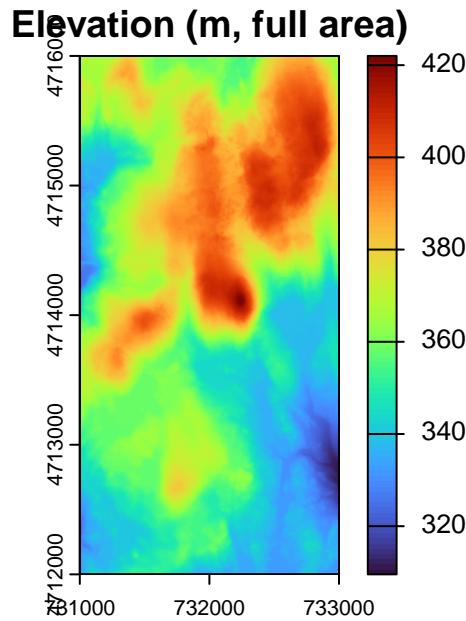
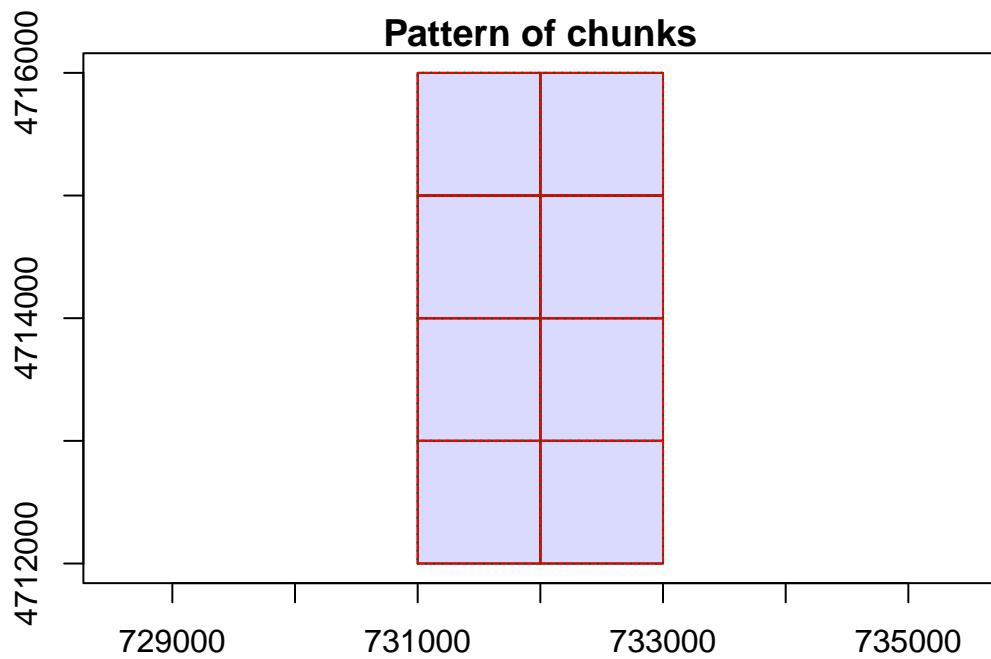


Figure 10: Digital Terrain Model for the full study area

Now we rasterize a CHM for the entire area:

```
# derive a canopy height model
# now we also restrict only to first returns and drop noise classes --> this works different
opt_filter(las_ctg) = "-keep_first -drop_class 7 18"
dsm_ctg_tin = rasterize_canopy(las_ctg, res = 1, algorithm = dsmtin(highest = FALSE))
```



Chunk 1 of 8 (12.5%): state

[=====>	] 9% ETA: 18s
[=====>	] 9% ETA: 18s
[=====>	] 9% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s
[=====>	] 10% ETA: 18s



Chunk 2 of 8 (25%): state

```
[==>          ] 5% ETA: 33s
[==>          ] 5% ETA: 32s
[==>          ] 6% ETA: 32s
[==>          ] 7% ETA: 32s
```



Chunk 4 of 8 (37.5%): state

```
[=====>          ] 12% ETA: 16s
[=====>          ] 13% ETA: 16s
```

Chunk 3 of 8 (50%): state

```
[===>          ] 6% ETA: 28s
[===>          ] 6% ETA: 27s
[===>          ] 7% ETA: 27s
```

```
[===> ] 7% ETA: 27s  
[===> ] 7% ETA: 26s  
[===> ] 8% ETA: 54s  
[===> ] 8% ETA: 106s
```

Chunk 5 of 8 (62.5%): state

```
[===> ] 8% ETA: 21s  
[===> ] 8% ETA: 21s
```



Chunk 8 of 8 (75%): state

```
[=====]> ] 9% ETA: 19s  
[=====]> ] 10% ETA: 18s  
[=====]> ] 11% ETA: 18s  
[=====]> ] 12% ETA: 18s  
[=====]> ] 12% ETA: 71s
```

Chunk 6 of 8 (87.5%): state

```
[==>          ] 5% ETA: 33s
[==>          ] 5% ETA: 32s
[==>          ] 5% ETA: 32s
[==>          ] 6% ETA: 31s
[==>          ] 7% ETA: 31s
[==>          ] 7% ETA: 31s
[==>          ] 7% ETA: 31s
```

```
[==>          ] 7% ETA: 31s
[==>          ] 7% ETA: 30s
[==>          ] 8% ETA: 94s
[==>          ] 8% ETA: 174s
```

Chunk 7 of 8 (100%): state

```
[==>          ] 5% ETA: 32s
[==>          ] 6% ETA: 32s
[==>          ] 6% ETA: 32s
```



```
[====>          ] 7% ETA: 31s
[====>          ] 7% ETA: 85s
[====>          ] 7% ETA: 151s
```

```
chm_ctg_tin = dsm_ctg_tin - dtm_ctg
```

Plot the CHM:

```
# plot the CHM
plot(chm_ctg_tin)
```

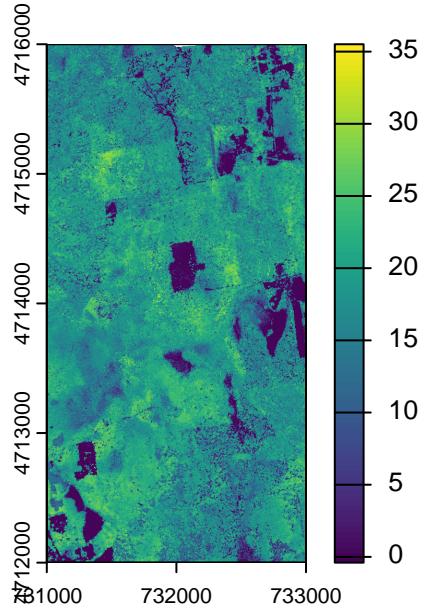


Figure 11: Canopy Height Model for the full study area

#### 4.5 Analysis: Forest Structure at Harvard Forest

What is the typical forest height?

The best way to find out is to use the CHM, and functions implemented in the “terra” package in R. Here’s a few functions that might be helpful:

### 💡 Useful Terra Functions

```
# check functions
?terra::global # calculate statistics for the entire raster
?terra::clamp # restrict the range of the data
?terra::zoom # zoom into a specific place, use only after terra::plot
?terra::draw # draw a polygon
?terra::crop # crop a scan to a polygon extent
?terra::mask # mask out areas with a polygon
?terra::extract # extract properties with a polygon
```

## 4.6 Tree-Based Analysis (Optional)

### 4.6.1 Using an Existing Data Set

We can use an existing dataset (<https://zenodo.org/records/10926344>) of AI-predicted tree crowns in the area. Note that the tree crowns were derived in 2022, so there is a 10 year difference to the scan date. Also note that the AI-derived tree crown extensions are given as rectangles. This is just an approximation.

```
# load pre-classified tree layer
trees = vect("data/trees_Weinstein2022/2022_HARV.shp")
trees = crop(trees, ext(chm_tin))
plot(crop(chm_tin, ext(chm_tin) * 0.1))
plot(crop(trees,ext(chm_tin) * 0.1), cex = 0.5, add = T)
```

We can also extract crown height (mean/max) from different CHMs:

```
# extract tree height
trees$height_tin_mean = terra::extract(chm_tin, trees, fun = "mean", ID = F)
trees$height_tin_max = terra::extract(chm_tin, trees, fun = "max", ID = F)
trees$height_highest_mean = terra::extract(chm_highest, trees, fun = "mean", ID = F)
trees$height_highest_max = terra::extract(chm_highest, trees, fun = "max", ID = F)

ggplot() +
  geom_histogram(data = as.data.table(trees), aes(x = height_tin_mean), fill = "red", alpha =
```

#### 4.6.2 Raster-Based Tree Segmentation

We can also segment trees ourselves. The idea is to first find tree tops and then segment trees. The approach works generally well in more open canopies, specifically conifer-dominated ones, but the approach does not work as well in closed canopy mixed and broadleaf forests. Trees are self-similar, so it is very hard to distinguish branches from crowns, or trees from each other.

##### ! Tree Segmentation Challenges

**A simple test:** If segmenting trees would be hard manually/visually, an AI/deterministic algorithm will probably not do better.

**A general guideline:** The best segmentations are usually achieved with:

- high-resolution photography (< 20 cm resolution), or
- a combination of photography and laser scanning point clouds

```
# now we try our own tree segmentation
kernel = matrix(1,5,5)

# remove noisy pits and spikes
chm_tin_smoothed = terra:::focal(chm_tin, w = kernel, fun = median, na.rm = TRUE)

# locate tree tops
ttops_tin = locate_trees(chm_tin, lmf(5))

# plot CHM + tree tops
plot(crop(chm_tin, ext(chm_tin) * 0.1))
plot(crop(vect(ttops_tin),ext(chm_tin) * 0.1), col = "black", cex = 0.5, add = T)
```

Try again, now with “highest” CHM (note: not robust to pulse density, but can be better for preserving canopy structure).

```
# repeat with highest CHM
kernel = matrix(1,5,5)
chm_highest_smoothed = terra:::focal(chm_highest, w = kernel, fun = median, na.rm = TRUE)
ttops_highest = locate_trees(chm_highest, lmf(5))

plot(crop(chm_highest, ext(chm_highest) * 0.1))
plot(crop(vect(ttops_highest),ext(chm_highest) * 0.1), col = "black", cex = 0.5, add = T)
```

Then we segment the trees:

```

# now segment tree crowns around the tree tops
algo_tin = dalponte2016(chm = chm_tin, treetops = ttops_tin)
algo_highest = dalponte2016(chm = chm_highest, treetops = ttops_highest)

# this uses the CHM to delineate crowns, a point cloud could also be used
trees_tin = rast(algo_tin()) # segment point cloud
trees_highest = rast(algo_highest()) # segment point cloud

# plot results
plot(trees_tin, col = pastel.colors(200))

plot(trees_highest, col = pastel.colors(200))

```

Also convert to polygons:

```

# convert to polygons / SpatVector data
trees_tin = as.polygons(trees_tin)
trees_highest = as.polygons(trees_highest)

# plot results
plot(chm_tin)
plot(trees_tin,, cex = 0.5, add = T)

plot(chm_highest)
plot(trees_highest, cex = 0.5, add = T)

```

**How stable are these segmentations?**

## 5 Summary and Key Takeaways

### 6 Data Quality

- Always check pulse density before analysis
- Evaluate scan angle and coverage patterns
- Remove noise and outliers systematically
- Consider temporal effects when comparing scans

## 7 Processing Workflow

- Start with ground classification (DTM)
- Create robust canopy height models (CHM)
- Use TIN method for more stable results
- Validate results across different methods

## 8 Tree Detection

Automated segmentation works best with:

- Open canopy forests
- High pulse density scans ( $> 10$  pulses/m<sup>2</sup>)
- Conifer-dominated stands
- High-resolution photography integration

Challenges:

- Closed canopy mixed forests
- Self-similar tree structures
- Distinguishing branches from crowns

## 9 Best Practices

- Prefer upper canopy data over within-canopy data
- Use simple metrics over complex ones
- Always conduct sensitivity analyses
- Document all processing steps
- Validate with field measurements when possible

## 10 Conclusion

This tutorial provides a practical workflow for processing airborne laser scanning (ALS) data in R using the lidR and terra packages. The methods demonstrated here can be applied to various forest structure analyses and remote sensing applications.

The key steps covered include:

1. **Data quality assessment** - evaluating pulse density and scan characteristics
2. **Terrain modeling** - creating digital terrain models using ground-classified points
3. **Canopy analysis** - generating canopy height models using different algorithms

#### 4. Tree detection - locating individual trees and segmenting crowns

##### Next Steps

To further develop your ALS processing skills:

- **Temporal analysis:** Compare multiple scans over time to detect forest changes
- **Structural metrics:** Calculate additional forest structure metrics (e.g., LAI, canopy cover, vertical profiles)
- **Integration:** Combine ALS with other remote sensing data (multispectral, hyperspectral)
- **Validation:** Conduct field surveys to validate remote sensing products
- **Automation:** Develop scalable processing pipelines for operational monitoring

## 11 Additional Resources

- **lidR handbook:** <https://r-lidar.github.io/lidRbook/>
- **NEON tutorials:** <https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics>
- **LAStools:** <https://rapidlasso.de/product-overview/>
- **Terra package:** <https://rspatial.org/terra/>

---

*This tutorial was designed to provide hands-on experience with ALS data processing, focusing on practical applications in forest structure analysis. The modular approach allows adaptation to specific research questions and study systems.*

---

