LiDAR Metrics Calculation for FIA Plot Data

Mukti Subedi

September 8, 2024

Table of contents

1 LiDAR Data

I am currently awaiting the download of other state's LiDAR data, as well as more local storage space. I will meet with Andres on Thursday (17th of August) to discuss the progress on the Soil Data Access, and Climate Data (**ClimateNA**). In the meantime, I have prepared a detailed LiDAR data analysis workflow, which I will discuss in this document.

1.1 Access to LiDAR tiles

The National Map is the primary repository for USGS base geospatial data. LiDAR data can be accessed using

1. 3DEP LiDAR Explorer
2. The National Map Download Client

With series of spatial, aspatial LiDAR information within the selected study states, we have several files that needs to be downloaded. Due to mismatch of project, sub-projects and sometimes tiles information from the data acquisition vendor, and USGS project. It is imperative that we calculate bounding box for our final data processing.

This will potentially give us an ideas left out areas for which above mention sources can be used to supplement data.

1.2 Getting Bounding Box: lidar tiles

As I mentioned before, we need to calculate bounding boxes for the LiDAR tiles data that we have downloaded by state. This will allow us to look at the data coverage and identify any gaps. One thing to keep in mind is that some of the projects in some states are named after another state. For example, there may be a project in Kansas that is named after MO. If we simply match the project name to the state name, we may miss these projects.

However, these types of projects are typically not that large may overlap along the state border. We can easily supplement our data by using the National Map Download Client to download the LiDAR tiles for these projects, especially when files are less than 5000.

```r
# function fast_bind sf

fast_rbind_sf <- function(x, check = FALSE) {
  # the code is copied from following sources
  # https://gist.github.com/kadyb
  # https://github.com/r-spatial/sf/issues/798
  if (length(x) == 0) stop("Empty list")
  if (isTRUE(check)) {
    ref_crs <- sf::st_crs(x[[1]])
    ref_colnames <- colnames(x[[1]])
    for (i in seq_len(length(x))) {
      if (isFALSE(sf::st_crs(x[[i]]) == ref_crs)) stop("Diffrent CRS")
      if (isFALSE(all(colnames(x[[i]]) == ref_colnames))) stop("Diffrent columns")
    }
  }
  geom_name <- attr(x[[1]], "sf_column")
  x <- collapse::unlist2d(x, idcols = FALSE, recursive = FALSE)
  x[[geom_name]] <- sf::st_sfc(x[[geom_name]], recompute_bbox = TRUE)
  x <- sf::st_as_sf(x)
}
```

The function `fast_bind_sf` is alternative to `rbind()` function. This is fast, as it uses `collapse::unlist2d()` function. The `sf()` functions did not work properly with `rbind()`.

1.2.1 Getting Tile Extent: A Function

The `get_tile_extent_parallel` function uses parallel approach to calculate bounding boxes of LiDAR tiles. Lets look at the processing time. Single core processing slightly more than one hour for 1937 files.

Before running function, we need to create arguments that goes inside it.

1.2.2 Run `get_tile_extent_parallel()` Function

I am using **an absolute path for following code chunk** you will not be able to run this code in your machine. However, I copied and pasted the file created using this function within the child directory (**02__tileExtent**) of a parent folder **LidarMetrics__Example**. If you want to knit this file, replace #— **replace here** with #|**eval: false**. This document is created usign Quarto. If you wish to run as Rmarkdwon, you can use eval = FALSE in your code chunk heading.

```r
#--- replace here
#-------------------------------------------------------------------------------#
library(doParallel)      # for parallel processing
library(parallel)        # for detecting number of cores
library(magrittr)        # For pipe operator
library(stringr)         # match string
library(dplyr)           # selecting and filter
library(sf)              # Simple feature data
library(rvest)           # Web scraping
library(lidR)            # For working with laz and las data forma
library(terra)           # for raster manipulation
library(rayshader)       # 3D plotting (rgl)
#-------------------------------------------------------------------------------#
st_abbr <- c(
  "AL", "AR", "CO", "DE", "FL", "GA", "IA", "IL", "IN", "KS",
  "KY", "LA", "MD", "MI", "MN",
  "MO", "MS", "NC", "NE", "NJ", "OH",
  "OK", "PA", "SC", "TN", "TX", "VA", "WV"
)

state <- st_abbr[10] # Kansas

# create directory if it does not exist
# if(!dir.exists(paste0("../00_lidTilesBB/", state))){
#   dir.create(path = paste0("../00_lidTilesBB/", state))
# }

LiDAR_path <- "H:/03_LiDARDataByState/KS/"

tile_list <- list.files("H:/03_LiDARDataByState/KS/",
  pattern = ".laz$",
  full.names = FALSE
)
# F drive is labelled as FPA_LiDAR1
tile_path <- "H:/03_LiDARDataByState/KS/"
gpkg_path <- "H:/chk_parallel/"

#------------------------------------------------------------------------------#
t1<- tictoc::tic()
# Call the modified function with provided arguments
get_tile_extent_parallel(
  tile_list = tile_list,
  state = state,
  LiDAR_path = LiDAR_path,
  gpkg_path = gpkg_path,
  num_cores = 5
)
```

```
[1] "total of  5 are used, out of 8 cores available"

  |
  |                                                                        |   0%Writing layer
  `H:/chk_parallel/BB_chk_KS.gpkg' using driver `GPKG'
Writing 1937 features with 2 fields and geometry type Polygon.
```

```
t2<- tictoc::toc()
```

```
17.87 sec elapsed
```

This is awesome it took only 17.87 sec elapsed. The dataset has 1937 features/polygons.

2 Calcualte LiDAR Metrics

My end goal is to wrap everything to calculate plot and sub-plot level lidar metrics into one function. It will calculate plot and subplot level lidar metrics so that the FIA personnel can run our analysis without having to do any of the code related activities by themselves. This will save us time and effort, and it will make the analysis process more efficient.

However, for now I will be using without function version of it. Just in case if we need to change anything. Moreover, I haven't tried if this can be processed in parallel. Some of the process/function uses parallel processing.

2.1 Calculate plot and subplot coordinate

I wrote one function that will calculate the coordinate pair (xy) for subplots based on FIA layout.

```
# ![FIA plot design: source(Colorado State Forest Service)](https://csfs.colostate.edu/wp-
 calculate_subplot_loc <- \(data, coords = c("LON", "LAT"),subplot = "subp",earth_radius =
  #
  # if(!is.numeric(earth_radius)||!is.numeric(R)){
  #   earth_radius = 6378.1
  #   warning("earth radius is not numeric, the default of 6378.1 km will be used")
  # }
  if(is.null(earth_radius)||is.na(earth_radius)||!is.numeric(earth_radius)||missing(earth_
    R <- 6378.1
    warning( "Earth radius is not specified,  the default of 6378.1 km will be used")
  }
  coord_ind <- tidyselect::eval_select(rlang::enquo(coords), data = data)

 # coords   <- cbind(data[, coord_ind[1]], data[, coord_ind[2]])

  sub_plot <- tidyselect::eval_select(rlang::enquo(subplot), data = data)
  #----------------------------------------------------------------------------#
  # calculate coordinates

 df_sub <- data %>%
```

```r
    # calculate bearing; using angle and distance method
    mutate(brng = case_when(
      data[, sub_plot] == 2L ~ (360 * pi / 180),
      data[, sub_plot] == 3L ~ (120 * pi / 180),
      data[, sub_plot] == 4L ~ (240 * pi / 180),
      data[, sub_plot] == 1L ~ (0 * pi / 180)
      ),
     dist_subp = case_when(
      data[, sub_plot] == 1L ~ 0,
      data[, sub_plot] != 1L ~ (120 / 3.2808) / 1000
     ),

     lon_rad = (pi / 180) * data[, coord_ind[1]],
     lat_rad = (pi / 180) * data[, coord_ind[2]],

     lat2 = asin(sin(lat_rad) * cos(dist_subp / R) +
                   cos(lat_rad) * sin(dist_subp / R) * cos(brng)),

     lon2 = lon_rad + atan2(
       sin(brng) * sin(dist_subp / R) * cos(lat_rad),
       cos(dist_subp / R) - sin(lat_rad) * sin(lat_rad)
     ),
     rlat = lat2 * 180 / pi,
     rlong = lon2 * 180 / pi
   ) %>%
     select(-c(dist_subp, lat_rad, lon_rad, lon2, lat2))
  return(df_sub)

} # close function

#calculate_subplot_loc(data = data,coords = c("LON", "LAT"), subplot = "subp")
```

For now I will not use `calculate_subplot_loc()` function, we will be using this for detecting subplot coordinate with **site tree**.

2.2 Miscellaneous Functions

To be able to calculate LiDAR metrics, several other functions are necessary, some that can fit within a single line will be included in the code chunk below, other are placed within the separate file, which can be sourced into final function.

These functions are basic such as `skewness()` `kurtosis()` `mode()` `quantiles()` or `percentiles()`

2.3 Prepare data

```r
# for calculating xy of subplot based on xy of subplot location
# this function is included in the above code chunk
# source('./13_subplot_coordinates_Calculation.R')
#--- I placed this code in previous code chunck
#------------------------------------------------ calculate_sub_plot_location
# Load skewness, kurtosis, and percentiles functions
source('./14_Miscellaneous_Funs.R')
# This miscellanues_Funs has functions for skewness, kurtosis and percentiles with
# type = 8
##-----------------------------------------------------------------------------##
#### - 4. Pre processing tabular and spatial data -####
## - Step 1- read Shape file - ##

## gdb location G drive (local disk 2 tb; my passport ultra)
gdb_path <- "../01_Geodatabase/UGA_LiDAR_Mukti_Andreas.gdb/"

## LiDAR tiles extent path (cut paste from FPA_lidar1/chk_parallel)
## This will be separate geopackage file for each state

# I copied the geopackage created above to another director for this purpose

tile_ext_path <- "../02_tileExtent/BB_oo_KS.gpkg"

## lidar Tiles path

tile_path <- "../03_tile_path/"
# Based on the FIA PLOT CN. I know which tiles will be used. I copied them and
# placed into this folder

##- Step 2 - fia plot and tile extent
# In the final version we will be using data actual coordinates file for location
# information, which will be a basis for lidar derived metrics.
# This will be depending on FIA Research Station computer.
#-----------------------------------------------------------------------------#
##  fia location
fia_loc <- st_read(gdb_path, layer = "state_28_plots_points")
```

```
Reading layer `state_28_plots_points' from data source
  `G:\LidarMetrics_Example\01_Geodatabase\UGA_LiDAR_Mukti_Andreas.gdb'
  using driver `OpenFileGDB'
Simple feature collection with 73784 features and 17 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -109.0547 ymin: 24.66552 xmax: -73.99914 ymax: 49.3489
Geodetic CRS:  NAD83
```

```r
state <- "KS"
### clip fia location to state boundary
state_bdr <- st_read(gdb_path, layer = "State_28_projectArea") %>%
  dplyr::filter(STATE_A == state)
```

```
Reading layer `State_28_projectArea' from data source
  `G:\LidarMetrics_Example\01_Geodatabase\UGA_LiDAR_Mukti_Andreas.gdb'
  using driver `OpenFileGDB'
Simple feature collection with 28 features and 4 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -109.0602 ymin: 24.52321 xmax: -73.90273 ymax: 49.38447
Geodetic CRS:  NAD83
```

```r
## Filter plot within State extent
fia_sel_loc <- st_intersection(fia_loc, state_bdr) %>% dplyr::select(pltID)

tile_ext <- st_read(tile_ext_path, layer = "BB_oo_KS")
```

```
Reading layer `BB_oo_KS' from data source
  `G:\LidarMetrics_Example\02_tileExtent\BB_oo_KS.gpkg' using driver `GPKG'
Simple feature collection with 1937 features and 2 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -511430.2 ymin: 1546702 xmax: 123996.2 ymax: 1904330
Projected CRS: NAD83 / Conus Albers
```

```r
##--- For now this is for the visualization purpose
## Due to large size of the data, we should avoid potting data tile extent
```

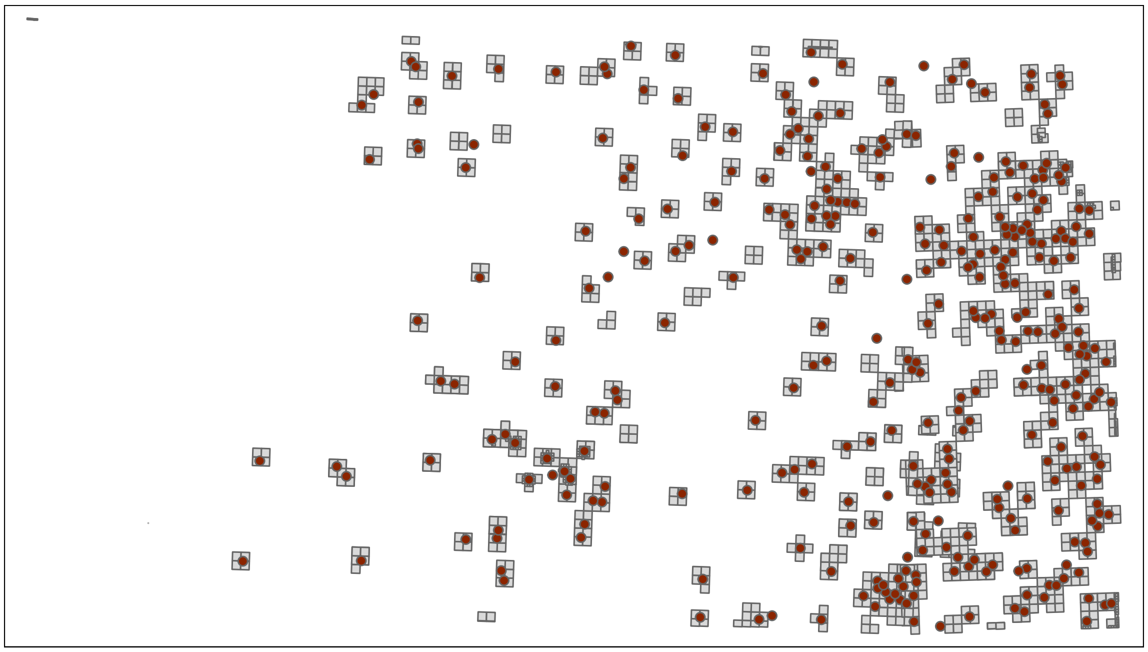
2.4 Plot lidar extent along with plot coordinates

Remember, the extent polygons are based on actual coordinates and we are using fuzzy coordinate for now.

```r
#---------------- setting tmap mode
tmap::tmap_mode("plot") ## use "view" mode for html
```

```
tmap mode set to plotting
```

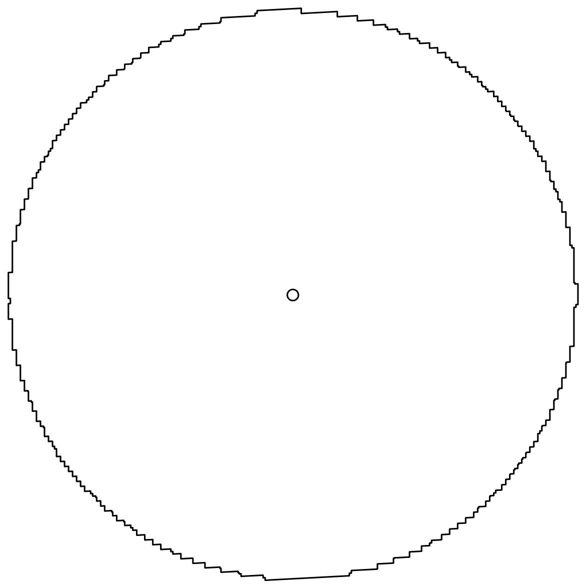
```r
tmap::tm_shape(tile_ext)+
tmap::tm_polygons()+
tmap::tm_shape(fia_sel_loc)+
tmap::tm_bubbles(size = 1/10, col = "orangered4",shape = 21)
```



```
#------------------------------------------------------------------------#
loc <- fia_sel_loc %>% dplyr::filter(pltID == "2_20_11_20360")

loc_buffer <- sf::st_buffer(loc, dist = units::set_units(1, "km"))

plot(st_geometry(loc_buffer))
plot(st_geometry(loc), add = TRUE)
```



```
# tmap::tm_shape(tile_ext) +
#   tmap::tm_polygons() +
#   tmap::tm_shape(loc_buffer) +
#   tmap::tm_polygons()
## select lidar tiles that intersects or touches loc_buffer

tile_read <- st_intersection(
  x = loc_buffer,
  y = st_transform(tile_ext,
                   crs = st_crs(loc_buffer)
  )
) %>%
  st_drop_geometry() %>%
  dplyr::select("tile_id")
```

```
Warning: attribute variables are assumed to be spatially constant throughout
all geometries
```

2.5 Read LiDAR Tile(s)

Based on polygons that were resulted from spatial intersection of buffered plot location and LiDAR tiles, the next step is about reading and cutting these shapes tiles. All the work done up to this point, including adjusting the area covered by LiDAR tiles, was leading to this. Earlier on, the index for tiles didn't have clear names or positions.

For the specific plot we're focused on, which has a buffer of 1 kilometer, the related LiDAR tiles are: **USGS_LPC_KS_Statewide_2018_A18_15S_UB_2485.laz, USGS_LPC_KS_Statewide_2018_A18_15S_UB_2985.laz**. To speed things up, I'm currently reading a version of these tiles that's already clipped to the area we need. The code for this is commented out in the following code chunk.

```
# For this rendering I m avoiding reading
#  lasf <- lidR::readALSLAS(files  = paste0(tile_path,tile_read$tile_id)) # check if this
# # chained
#  las_crs <- st_crs(lasf)
#
# lasf_clip <- clip_roi(las = lasf, geometry = st_transform(loc_buffer, crs = las_crs))
#
# lidR::writeLAS(lasf_clip, file = "./lasf_clip.las")
#----------------------- read the lasf file ---------------------------------#
lasf_clip <- readLAS("./lasf_clip.las")
```

```
Warning: There are 2793332 points flagged 'withheld'.
```

```
las_crs <- st_crs(lasf_clip)
## ------------- filter points that are necessary
# "-keep_first -drop_z_below 5 -drop_z_above 50")
```

```r
# -drop_withheld
## -------------
# data.table way of filter for faster calculation

lasf_clip@data <- lasf_clip@data[!(ReturnNumber > NumberOfReturns)]

lasf_clip@data <- lasf_clip@data[Withheld_flag == FALSE]
lasf_clip@data <- lasf_clip@data[(Z < 300)] # there is one extreme value

#plot(lasf_clip)
```

```r
## only for visualization purpose

dtm <- lidR::rasterize_terrain(lasf_clip,use_class = c(2L,9L), res = 5, algorithm = tin())
```

```
Delaunay rasterization[================================------------------] 64% (4 threads)
Delaunay rasterization[================================------------------] 65% (4 threads)
Delaunay rasterization[=================================-----------------] 66% (4 threads)
Delaunay rasterization[=================================-----------------] 67% (4 threads)
Delaunay rasterization[==================================----------------] 68% (4 threads)
Delaunay rasterization[==================================----------------] 69% (4 threads)
Delaunay rasterization[===================================---------------] 70% (4 threads)
Delaunay rasterization[===================================---------------] 71% (4 threads)
Delaunay rasterization[====================================--------------] 72% (4 threads)
Delaunay rasterization[====================================--------------] 73% (4 threads)
Delaunay rasterization[=====================================-------------] 74% (4 threads)
Delaunay rasterization[=====================================-------------] 75% (4 threads)
Delaunay rasterization[======================================------------] 76% (4 threads)
Delaunay rasterization[======================================------------] 77% (4 threads)
Delaunay rasterization[=======================================-----------] 78% (4 threads)
Delaunay rasterization[=======================================-----------] 79% (4 threads)
Delaunay rasterization[========================================----------] 80% (4 threads)
Delaunay rasterization[========================================----------] 81% (4 threads)
Delaunay rasterization[=========================================---------] 82% (4 threads)
Delaunay rasterization[=========================================---------] 83% (4 threads)
Delaunay rasterization[==========================================--------] 84% (4 threads)
Delaunay rasterization[==========================================--------] 85% (4 threads)
Delaunay rasterization[===========================================-------] 86% (4 threads)
Delaunay rasterization[===========================================-------] 87% (4 threads)
Delaunay rasterization[============================================------] 88% (4 threads)
Delaunay rasterization[============================================------] 89% (4 threads)
Delaunay rasterization[=============================================-----] 90% (4 threads)
Delaunay rasterization[=============================================-----] 91% (4 threads)
Delaunay rasterization[==============================================----] 92% (4 threads)
Delaunay rasterization[==============================================----] 93% (4 threads)
Delaunay rasterization[===============================================---] 94% (4 threads)
Delaunay rasterization[===============================================---] 95% (4 threads)
Delaunay rasterization[================================================--] 96% (4 threads)
Delaunay rasterization[================================================--] 97% (4 threads)
Delaunay rasterization[=================================================-] 98% (4 threads)
Delaunay rasterization[=================================================-] 99% (4 threads)
Delaunay rasterization[==================================================] 100% (4 threads)
```

```
# norm_lidar <- lidR::normalize_height(las = lasf_clip,algorithm = tin(),
#                                       use_class = c(2L,9L), dtm = dtm)

elev_mat <- rayshader::raster_to_matrix(dtm)

plt <- elev_mat %>%
  sphere_shade(texture = "imhof1") %>%
  add_water(detect_water(elev_mat), color = "imhof1") %>%
  add_shadow(ray_shade(elev_mat, zscale = 3), 0.5)

pal <- colorRampPalette(height.colors(10))
col <- pal(30)[as.numeric(cut(lasf_clip$Z, breaks = 30))]

plt %>%
  plot_3d(elev_mat, zscale = 1, fov = 0, theta = 135, zoom = 0.75,
          phi = 45, windowsize = c(800, 800),
           waterdepth = 0,
           wateralpha = 0.5,
          watercolor = "lightblue",
           waterlinecolor = "white",
           waterlinealpha = 0.5
          )
render_points(extent = ext(dtm),
              long     = lasf_clip$X,
              lat      = lasf_clip$Y,
              altitude = lasf_clip$Z,
              size     = 2,
              color    = col)
# library("plotly")
# las_plot<- lasf_clip[, c(X,Y,Z)]
# ggplot(lasf_clip@data, aes(X,Z, color = Z))+
#   geom_point(size = 0.5) +
#   coord_equal() +
#   theme_minimal() +
#   scale_color_gradientn(colours = height.colors(50))
```

3 Calculating Variables

Forest structure influences many important ecological processes, including species richness and diversity,biogeochemical cycling, and others. Forest structural diversity quantification over the large are using lidar data is considered to be feasible and robust compared to traditional sample based estimation. However, lidar data poses sampling, statistical, and methodological constrains.LiDAR data acquisition is often intended to provide detailed data for a specific target areas which varies in sensor types, plant phenology (leaf on vs leaf off), and year of acquisition.

3.1 Digital Elevation/Terrain Model based Variables

Digital elevation models (DEMs) are widely used in landscape and forest ecology to understand the distribution of species. Generally, DEM is used to retrieve elevation or computing primary terrain attributes such as slope aspect. Primary terrain attributes can inform about underlie biophysical processes at local or regional scales, especially in mountainous areas. Primary terrain attributes have been used as proxies for factors such as solar radiation, overland and subsurface flow, transpiration, available soil moisture or soil water content, soil characteristics, wind, erosion/deposition rate and others.

Here we will calculate primary attributes such as mean elevation, slope, aspect. Similarly some secondary attributes such as topographic ruggedness index (TRI),topographic position index (TPI) will be calculated at plot level. TRI is the mean of the absolute differences between the value of a cell and its 8 surrounding cells. TPI is the difference between the value of a cell and the mean value of its 8 surrounding cells. Roughness (rough; code chunk below) is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells. For the sake of simplicity these three attributes are calculated using `focal()` function.

```
### -------------------------------------------------- DEM related variables
# Select the ground points
ground_plot <- rasterize_terrain(las = lasf_clip, res = 10, algorithm = tin(),
                                 use_class = c(2L, 9L))
```

```
Delaunay rasterization[================================------------------] 64% (4 threads)
Delaunay rasterization[================================------------------] 65% (4 threads)
Delaunay rasterization[=================================-----------------] 66% (4 threads)
Delaunay rasterization[=================================-----------------] 67% (4 threads)
Delaunay rasterization[==================================----------------] 68% (4 threads)
Delaunay rasterization[==================================----------------] 69% (4 threads)
Delaunay rasterization[===================================---------------] 70% (4 threads)
Delaunay rasterization[===================================---------------] 71% (4 threads)
Delaunay rasterization[====================================--------------] 72% (4 threads)
Delaunay rasterization[====================================--------------] 73% (4 threads)
Delaunay rasterization[=====================================-------------] 74% (4 threads)
Delaunay rasterization[=====================================-------------] 75% (4 threads)
Delaunay rasterization[======================================------------] 76% (4 threads)
Delaunay rasterization[======================================------------] 77% (4 threads)
Delaunay rasterization[=======================================-----------] 78% (4 threads)
Delaunay rasterization[=======================================-----------] 79% (4 threads)
Delaunay rasterization[========================================----------] 80% (4 threads)
Delaunay rasterization[========================================----------] 81% (4 threads)
Delaunay rasterization[=========================================---------] 82% (4 threads)
Delaunay rasterization[=========================================---------] 83% (4 threads)
Delaunay rasterization[==========================================--------] 84% (4 threads)
Delaunay rasterization[==========================================--------] 85% (4 threads)
Delaunay rasterization[===========================================-------] 86% (4 threads)
Delaunay rasterization[===========================================-------] 87% (4 threads)
Delaunay rasterization[============================================------] 88% (4 threads)
Delaunay rasterization[============================================------] 89% (4 threads)
Delaunay rasterization[=============================================-----] 90% (4 threads)
Delaunay rasterization[=============================================-----] 91% (4 threads)
Delaunay rasterization[==============================================----] 92% (4 threads)
Delaunay rasterization[==============================================----] 93% (4 threads)
Delaunay rasterization[===============================================---] 94% (4 threads)
Delaunay rasterization[===============================================---] 95% (4 threads)
Delaunay rasterization[================================================--] 96% (4 threads)
Delaunay rasterization[================================================--] 97% (4 threads)
Delaunay rasterization[=================================================-] 98% (4 threads)
Delaunay rasterization[=================================================-] 99% (4 threads)
Delaunay rasterization[==================================================] 100% (4 threads)
```

```
## may be use krigging in the final function # algorithm = kriging(k = 5)

## calculate slope and aspect
```

```r
TRI  <- terra::focal(ground_plot, w = 3, fun = \(x) sum(abs(x[-5] - x[5])) / 8)
TPI   <- terra::focal(ground_plot, w = 3, fun = \(x) x[5] - mean(x[-5]))
rough <- terra::focal(ground_plot, w = 3, fun = \(x) max(x) - min(x))
slope <- terra::terrain(ground_plot, v = "slope", neighbors = 8)
aspect <- terra::terrain(ground_plot, v = "aspect")
# slope <- terra::focal(ground_plot, w = matrix(1/9,nrow = 3, ncol = 3), fun = mean)
avg_dem <- terra::extract(
  x = ground_plot, y = st_transform(loc_buffer, crs = las_crs),
  FUN = mean, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(avg_dem = mean(.[, 1], na.rm = TRUE))
sd_dem <- terra::extract(
  x = ground_plot, y = st_transform(loc_buffer, crs = las_crs),
  FUN = sd, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarize(sd_dem = sd(.[, 1], na.rm = TRUE))

avg_TRI <- terra::extract(
  x = TRI, y = st_transform(loc_buffer, crs = las_crs),
  FUN = mean, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(mean_TRI = mean(.[, 1], na.rm = TRUE))
sd_TRI <- terra::extract(
  x = TRI, y = st_transform(loc_buffer, crs = las_crs),
  FUN = sd, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(sd_TRI = mean(.[, 1], na.rm = TRUE))

avg_TPI <- terra::extract(
  x = TPI, y = st_transform(loc_buffer, crs = las_crs),
  FUN = mean, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(mean_TPI = mean(.[, 1], na.rm = TRUE))
sd_TPI <- terra::extract(
  x = TPI, y = st_transform(loc_buffer, crs = las_crs),
  FUN = sd, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(sd_TPI = mean(.[, 1], na.rm = TRUE))

avg_slope <- terra::extract(
  x = slope, y = st_transform(loc_buffer, crs = las_crs),
  FUN = mean, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(mean_slope = mean(.[, 1], na.rm = TRUE))
sd_slope <- terra::extract(
  x = slope, y = st_transform(loc_buffer, crs = las_crs),
  FUN = sd, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(sd_slope = mean(.[, 1], na.rm = TRUE))

avg_aspect <- terra::extract(
  x = aspect, y = st_transform(loc_buffer, crs = las_crs),
  FUN = mean, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(mean_aspect = mean(.[, 1], na.rm = TRUE))
sd_aspect <- terra::extract(
  x = aspect, y = st_transform(loc_buffer, crs = las_crs),
  FUN = sd, na.rm = TRUE, ID = FALSE
) %>%
  dplyr::summarise(sd_aspect = mean(.[, 1], na.rm = TRUE))

df_dem <- data.frame(
  "avg_dem" = avg_dem,
  "sd_dem" = sd_dem,
  "avg_TRI" = avg_TRI,
  "sd_TRI" = sd_TRI,
  "avg_TPI" = avg_TPI,
  "sd_tpi" = sd_TPI,
  "avg_asp" = avg_aspect
)
```

Let's examine the output table

```
head(df_dem)
```

3.2 Canopy Height Model based

Once the DEM related variables were computed, LiDAR points were normalized there by resulting only height information above the ground/terrain. This way all points will have absolute height above the ground. However, some points may be extreme (really height height) such as for Television or communication towers. In such cases these values should be removed. As per my analysis, height of 47.0 m was recorded for Yellow poplar in Georgia. However, I will be using height threshold of 45m in this analysis.

```
#### ----- define function for las metrics ------------------------------------#

calculate_surf_metrics <- \(lasf_clip, min_ht = 1.37,
                            cover_threshold = 2, max_ht = 45){
  # lasf_clip = clipped las file
  # min_ht = minimum height of tree (in use it is 1.37 m)
  # max_ht = define maximum height threshold. The maximum height recorded
  # in FIA data is around 47 m for Yellow Poplar.
  #-----------------------------------------------------------------------------#
  if (class(min_ht) != "numeric") {
    warning("The min_ht parameter is not numeric. Defaulting to 1.37 m")
    min_ht <- 1.37
  }
  if (class(cover_threshold) != "numeric") {
    warning("The ht_threshold parameter is not a numeric input. defaulting to 2 m")

    cover_threshold <- 2.0
  }
  if (class(max_ht) != "numeric") {
    warning("no maximum height threshold is defined (max_ht), therefore, no filter will be
  }
  ##### ----------- las clip is the data that was clip within the pre-determined
  #### ------------ buffer
  copy_las_clip <- lasf_clip
  # ground_plot  <- rasterize_terrain(las = lasf_clip,res = 10,algorithm = tin(),use_class
  abs_height <- lidR::normalize_height(las = lasf_clip, use_class = c(2L, 9L), algorithm =
  if(!is.null(max_ht)) {
    abs_height@data <- abs_height@data[(Z< max_ht )]
  }
  #---------------------------------------------------------------------- calculate
  data_size                      <- nrow(abs_height)
  all_return_min_ht              <- abs_height@data[(Z > min_ht)]
  first_return                   <- abs_height@data[(ReturnNumber == 1L)]
  all_return_cover_thresh        <- abs_height@data[(Z > cover_threshold)]
  first_return_cover_thresh      <- first_return[(Z > cover_threshold)]
  first_return_cover_thresh_mean <- first_return[(Z > mean(all_return_cover_thresh[["Z"]])
  # write Function for Mode see file 14 for miscellaneous functions
  first_return_cover_thresh_mode <- first_return[(Z > calc_mode(all_return_cover_thresh[["
  all_return_cover_thresh_mean   <- abs_height@data[(Z > mean(all_return_cover_thresh[["Z"
  h_cover_thresh_mode            <- calc_mode(all_return_cover_thresh[["Z"]])
  all_cover_thresh_mode          <- abs_height@data[(Z > h_cover_thresh_mode)]
```

```r
#-----------------------------------------------------------------------------#

## calculate metrics
total_return <- (data_size)
total_first_return <- nrow(first_return)
total_return_above_min_ht <- nrow(all_return_min_ht)
# return number
#----------------------------------------------------------------------------#
return_1_count <- nrow(all_return_min_ht[(ReturnNumber == 1L)])
return_2_count <- nrow(all_return_min_ht[(ReturnNumber == 2L)])
return_3_count <- nrow(all_return_min_ht[(ReturnNumber == 3L)])
return_4_count <- nrow(all_return_min_ht[(ReturnNumber == 4L)])
return_5_count <- nrow(all_return_min_ht[(ReturnNumber == 5L)])
return_6_count <- nrow(all_return_min_ht[(ReturnNumber == 6L)])
return_7_count <- nrow(all_return_min_ht[(ReturnNumber == 7L)])
return_8_count <- nrow(all_return_min_ht[(ReturnNumber == 8L)])
return_9_count <- nrow(all_return_min_ht[(ReturnNumber == 9L)])
#----------------------------------------------------------------------------#
h_min <- round(min(all_return_min_ht[["Z"]]), digits = 2)
h_max <- round(max(all_return_min_ht[["Z"]]), digits = 2)
h_mean <- round(mean(all_return_min_ht[["Z"]]), digits = 2)
h_median <- round(median(all_return_min_ht[["Z"]]), digits = 2)
h_mode <- round(calc_mode(all_return_min_ht[["Z"]]), digits = 2)
h_std_dev <- round(sd(all_return_min_ht[["Z"]]), digits = 2)
h_var <- round(var(all_return_min_ht[["Z"]]), digits = 2)
h_coef_var <- round(h_std_dev / h_mean * 100, digits = 2)
h_kurtosis <- round(calc_kurtosis(all_return_min_ht[["Z"]]), digits = 2)
h_skewness <- round(calc_skewness(all_return_min_ht[["Z"]]), digits = 2)
canopy_relief_ratio <- (h_mean - h_min) / (h_max - h_min)
#----------------------------------------------------------------------------#
# percentiles calculation
h_pct_01 <- round(pct_01(all_return_min_ht[["Z"]]), digits = 2)
h_pct_05 <- round(pct_05(all_return_min_ht[["Z"]]), digits = 2)
h_pct_10 <- round(pct_10(all_return_min_ht[["Z"]]), digits = 2)
h_pct_15 <- round(pct_15(all_return_min_ht[["Z"]]), digits = 2)
h_pct_20 <- round(pct_20(all_return_min_ht[["Z"]]), digits = 2)
h_pct_25 <- round(pct_25(all_return_min_ht[["Z"]]), digits = 2)
h_pct_30 <- round(pct_30(all_return_min_ht[["Z"]]), digits = 2)
h_pct_35 <- round(pct_35(all_return_min_ht[["Z"]]), digits = 2)
h_pct_40 <- round(pct_40(all_return_min_ht[["Z"]]), digits = 2)
h_pct_45 <- round(pct_45(all_return_min_ht[["Z"]]), digits = 2)
```

```
h_pct_50 <- round(pct_50(all_return_min_ht[["Z"]]), digits = 2)
h_pct_55 <- round(pct_55(all_return_min_ht[["Z"]]), digits = 2)
h_pct_60 <- round(pct_60(all_return_min_ht[["Z"]]), digits = 2)
h_pct_65 <- round(pct_65(all_return_min_ht[["Z"]]), digits = 2)
h_pct_70 <- round(pct_70(all_return_min_ht[["Z"]]), digits = 2)
h_pct_75 <- round(pct_75(all_return_min_ht[["Z"]]), digits = 2)
h_pct_80 <- round(pct_80(all_return_min_ht[["Z"]]), digits = 2)
h_pct_85 <- round(pct_85(all_return_min_ht[["Z"]]), digits = 2)
h_pct_90 <- round(pct_90(all_return_min_ht[["Z"]]), digits = 2)
h_pct_95 <- round(pct_95(all_return_min_ht[["Z"]]), digits = 2)
h_pct_98 <- round(pct_98(all_return_min_ht[["Z"]]), digits = 2)
h_pct_99 <- round(pct_99(all_return_min_ht[["Z"]]), digits = 2)
#-------------------------------------------------------------------------------#
# Intensity related metrics
#-------------------------------------------------------------------------------#
int_min <- round(min(all_return_min_ht[["Intensity"]]), digits = 2)
int_max <- round(max(all_return_min_ht[["Intensity"]]), digits = 2)
int_mean <- round(mean(all_return_min_ht[["Intensity"]]), digits = 2)
int_median <- round(median(all_return_min_ht[["Intensity"]]), digits = 2)
int_mode <- round(calc_mode(all_return_min_ht[["Intensity"]]), digits = 2)
int_std_dev <- round(sd(all_return_min_ht[["Intensity"]]), digits = 2)
int_var <- round(var(all_return_min_ht[["Intensity"]]), digits = 2)
int_coef_var <- round(int_std_dev / int_mean * 100, digits = 2)
int_kurtosis <- round(calc_kurtosis(all_return_min_ht[["Intensity"]]), digits = 2)
int_skwness <- round(calc_skewness(all_return_min_ht[["Intensity"]]), digits = 2)
#-------------------------------------------------------------------------------#
# percentiles calculation
int_pct_01 <- round(pct_01(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_05 <- round(pct_05(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_10 <- round(pct_10(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_15 <- round(pct_15(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_20 <- round(pct_20(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_25 <- round(pct_25(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_30 <- round(pct_30(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_35 <- round(pct_35(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_40 <- round(pct_40(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_45 <- round(pct_45(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_50 <- round(pct_50(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_55 <- round(pct_55(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_60 <- round(pct_60(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_65 <- round(pct_65(all_return_min_ht[["Intensity"]]), digits = 2)
```

```
int_pct_70 <- round(pct_70(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_75 <- round(pct_75(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_80 <- round(pct_80(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_85 <- round(pct_85(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_90 <- round(pct_90(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_95 <- round(pct_95(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_98 <- round(pct_98(all_return_min_ht[["Intensity"]]), digits = 2)
int_pct_99 <- round(pct_99(all_return_min_ht[["Intensity"]]), digits = 2)
#---------------------------------------------------------------------------#
## cover_threshold related metrics
pct_first_return_cover_thresh_return     = (nrow(first_return_cover_thresh)/nrow(first_r
pct_all_first_return_cover_thresh_return = (nrow(all_return_cover_thresh)/ nrow(first_re
pct_all_return_to_first_return    = (nrow(all_return_cover_thresh)/data_size)*100
first_return_cover_thresh         = (nrow(first_return_cover_thresh))
all_return_cover_thresh           = (nrow(all_return_cover_thresh))

pct_first_returns_above_mean      = (nrow(first_return_cover_thresh_mean)/nrow(first_ret
pct_first_returns_above_mode      = (nrow(first_return_cover_thresh_mode)/nrow(first_ret
pct_all_returns_above_mean        = (nrow(all_return_cover_thresh_mean)/data_size)*100
pct_all_returns_above_mode        = (nrow(all_cover_thresh_mode)/data_size)*100

##---
pct_all_return_above_mean_total_first_return = (nrow(first_return_cover_thresh_mean)/nro
pct_all_return_above_mode_total_first_return = (nrow(first_return_cover_thresh_mode)/nro
first_return_above_mean = nrow(first_return_cover_thresh_mean)
first_returns_aobve_mean = nrow(first_return_cover_thresh_mode)
all_returns_above_mean = nrow(all_return_cover_thresh_mean)
all_returns_above_mode = nrow(all_cover_thresh_mode)
#------------------------------------------------------------------------------#
# Other canopy related metrics
# Install canopyLazR from GitHub
# install_github("akamoske/canopyLazR")
#------------------------------------------------------------------------------#
chm <- rasterize_canopy(las = abs_height,res = 1,algorithm = dsmtin())
# height below 1.37; set NA
chm[chm$Z< min_ht]<- NA # OR

#chm[chm[3,]< min_ht]<- NA
# calculate rumple index

#calculate rumple, a ratio of outer canopy surface area to ground surface area
```

```r
rumple <- rumple_index(chm)
# top rugosity, the standard deviation of pixel values in chm and is a measure of
# canopy roughness

top_rugosity <- h_std_dev
# Deep gap and deep gap fraction
# number of cells in raster
pixels   <- nrow(chm[,,1]) #terra::nrow(chm)*terra::ncol(chm) nrow
# extract Z attributes
chm_arr  <- chm[,,1]
## remoe NAs to zeros
chm_arr[is.na(chm_arr)] <- 0
#chm_zero <- chm
#chm_zero[is.na(chm_zero)] <- 0
zeros <- chm_arr[chm_arr[, ]==0,]
# deep gaps
deep_gaps <- length(zeros)
# deep-gap fraction
deep_gap_fraction <- deep_gaps/pixels
# cover fraction 1/ deep gap_fraction
cover_fraction <- 1 - deep_gap_fraction
#height SD, the standard deviation of height values for all points
#in the plot point cloud
vert_sd <- sd(abs_height$Z, na.rm = TRUE)
#rasterize plot point cloud and calculate the standard deviation
#of height values at a resolution of 1 m^2
sd_1m2  <- grid_metrics(abs_height, sd(Z), 1)
sd_sd   <- sd(sd_1m2[,3], na.rm = TRUE)
# following function won't handle NAs, so we need
# to filter these out of a vector of Z points
Zs <-  chm_arr
#chm_arr[is.na(chm_arr)] <- 0
Zs_na <- chm_arr[!is.na(chm_arr)]
#entropy, quantifies diversity & evenness of point cloud heights
#by = 1 partitions point cloud in 1 m tall horizontal slices
#ranges from 0-1, with 1 being more evenly distributed points
#across the 1 m tall slices
can_entropy <- entropy(Zs_na, by = 1)
#gap fraction profile, assesses the distribution of gaps in the
#canopy volume
#dz = 1 partitions point cloud in 1 m horizontal slices
```

```
#z0 is set to a reasonable height based on the age and height of
#the study sites
gap_frac <- gap_fraction_profile(Zs_na, dz = 1, z0=3)
#defines gap fraction profile as the average gap fraction in each
#1 m horizontal slice assessed in the previous line
gap_fac_avg <- mean(gap_frac$gf)
#leaf area density, assesses leaf area in the canopy volume
#k = 0.5 is a standard extinction coefficient for foliage
#dz = 1 partitions point cloud in 1 m horizontal slices
#z0 is set to the same height as gap fraction profile above

leaf_density <- LAD(Zs_na, dz = 2, k = 0.5, z0 = 3)
#vegetation area index, sum of leaf area density values for
#all horizontal slices assessed in previous line
vai_sum <- sum(leaf_density$lad, na.rm = TRUE)
#vertical complexity index, fixed normalization of entropy
#metric calculated above
#set zmax comofortably above maximum canopy height
#by = 1 assesses the metric based on 1 m horizontal slices in
#the canopy
vci_avg <- VCI(Zs_na, by = 1, zmax = max(Zs_na))
fhd = can_entropy* log(max(Zs_na))

#-----------------------------------------------------------------------------#
### assign variables
# this creates golobal variables
#-----------------------------------------------------------------------------#
assign(paste0("total_return_above_",   min_ht), total_return_above_min_ht)
assign(paste0("Return_1_count_above_", min_ht), return_1_count)
assign(paste0("Return_2_count_above_", min_ht), return_2_count)
assign(paste0("Return_3_count_above_", min_ht), return_3_count)
assign(paste0("Return_4_count_above_", min_ht), return_4_count)
assign(paste0("Return_5_count_above_", min_ht), return_5_count)
assign(paste0("Return_6_count_above_", min_ht), return_6_count)
assign(paste0("Return_7_count_above_", min_ht), return_7_count)
assign(paste0("Return_8_count_above_", min_ht), return_8_count)
assign(paste0("Return_9_count_above_", min_ht), return_9_count)
###------------------------------------------------------------

assign(paste0("pct_first_return_above_", cover_threshold), pct_first_return_cover_thresh
assign(paste0 ("pct_all_return_above_", cover_threshold) , pct_all_first_return_cover_th
assign(paste0("pct_all_return_above_", cover_threshold,"_to_tot_first_return"), pct_all_
assign(paste0("first_return_above_", cover_threshold), first_return_cover_thresh)
assign(paste0("all_return_above_", cover_threshold) , all_return_cover_thresh)


###------------ return data.frame
return_df <-  data.frame(
  "total_return"                           =  data_size,
  "total_first_return"                     =  total_first_return,
  mget(paste0("total_return_above_",   min_ht)),
  mget(paste0("Return_1_count_above_", min_ht)),
  mget(paste0("Return_2_count_above_", min_ht)),
  mget(paste0("Return_3_count_above_", min_ht)),
  mget(paste0("Return_4_count_above_", min_ht)),
  mget(paste0("Return_5_count_above_", min_ht)),
  mget(paste0("Return_6_count_above_", min_ht)),
  mget(paste0("Return_7_count_above_", min_ht)),
  mget(paste0("Return_8_count_above_", min_ht)),
  mget(paste0("Return_9_count_above_", min_ht)),

  #--------------------------------------------------------------------------------#
  "h_min" = h_min,
  "h_max" = h_max,
  "h_mean"  = h_mean,
  "h_median" = h_median,
  "h_mode"  = h_mode,
  "h_sd"    = h_std_dev,
  "h_var"  = h_var,
  "h_cv"   = h_coef_var,
  "h_kurtosis" = h_kurtosis,
  "h_skwness"  = h_skewness,
  "canopy_relief_ratio" = canopy_relief_ratio,
  #-------------------------------------------------------------------------------#
  "h_pct_01"      =  h_pct_01,
  "h_pct_05"      =  h_pct_05,
  "h_pct_10"      =  h_pct_10,
  "h_pct_15"      =  h_pct_15,
  "h_pct_20"      =  h_pct_20,
  "h_pct_25"      =  h_pct_25,
  "h_pct_30"      =  h_pct_30,
  "h_pct_35"      =  h_pct_35,
  "h_pct_40"      =  h_pct_40,
  "h_pct_45"      =  h_pct_45,
  "h_pct_50"      =  h_pct_50,
  "h_pct_55"      =  h_pct_55,
  "h_pct_60"      =  h_pct_60,
  "h_pct_65"      =  h_pct_65,
  "h_pct_70"      =  h_pct_70,
  "h_pct_75"      =  h_pct_75,
  "h_pct_80"      =  h_pct_80,
  "h_pct_85"      =  h_pct_85,
  "h_pct_90"      =  h_pct_90,
  "h_pct_95"      =  h_pct_95,
  "h_pct_98"      =  h_pct_98,
  "h_pct_99"      =  h_pct_99,
  #------------------------------------------------------------------------------#
  "int_min"         = int_min,
  "int_max"         = int_max,
  "int_mean"        = int_mean,
  "int_median"      = int_median,
  "int_mode"        = int_mode,
  "int_sd"          = int_std_dev,
  "int_var"         = int_var,
  "int_cv"          = int_coef_var,
  "int_kurtosis"    = int_kurtosis,
  "int_skwness"     = int_skwness,
  #------------------------------------------------------------------------------#
  "int_pct_01"      =  int_pct_01,
  "int_pct_05"      =  int_pct_05,
  "int_pct_10"      =  int_pct_10,
  "int_pct_15"      =  int_pct_15,
  "int_pct_20"      =  int_pct_20,
  "int_pct_25"      =  int_pct_25,
  "int_pct_30"      =  int_pct_30,
  "int_pct_35"      =  int_pct_35,
  "int_pct_40"      =  int_pct_40,
  "int_pct_45"      =  int_pct_45,
  "int_pct_50"      =  int_pct_50,
  "int_pct_55"      =  int_pct_55,
  "int_pct_60"      =  int_pct_60,
  "int_pct_65"      =  int_pct_65,
  "int_pct_70"      =  int_pct_70,
  "int_pct_75"      =  int_pct_75,
  "int_pct_80"      =  int_pct_80,
  "int_pct_85"      =  int_pct_85,
  "int_pct_90"      =  int_pct_90,
  "int_pct_95"      =  int_pct_95,
  "int_pct_98"      =  int_pct_98,
  "int_pct_99"      =  int_pct_99,
  #--------------------------------------------------------------------------------#
  mget(paste0("pct_first_return_above_",cover_threshold)),
  mget(paste0 ("pct_all_return_above_",cover_threshold)),
  mget(paste0("pct_all_return_above_", cover_threshold,"_to_tot_first_return")),
  mget(paste0("first_return_above_",cover_threshold)),
  mget(paste0("all_return_above_",cover_threshold)),
  "pct_first_returns_above_mean"       = pct_first_returns_above_mean,
  "pct_first_returns_above_mode"       = pct_first_returns_above_mode,
  "pct_all_returns_above_mean"          = pct_all_returns_above_mean,
  "pct_all_returns_above_mode"          = pct_all_returns_above_mode,
  "pct_all_return_above_mean_to_tot_first_return"= pct_all_return_above_mean_total_first
  "pct_all_return_above_mode_to_tot_first_return"= pct_all_return_above_mode_total_first
  "first_return_above_mean"  = first_return_above_mean,
  "first_returns_aobve_mean" =  first_returns_aobve_mean,
  "all_returns_above_mean"   = all_returns_above_mean,
  "all_returns_above_mode"  = all_returns_above_mode,
  "rumple_index" = rumple,
"deep_gap"   = deep_gaps,
"deep_gap_faction" = deep_gap_fraction ,
"cover_fraction"   = cover_fraction ,
"top_rugosity" = top_rugosity,
"sd_sd"        = sd_sd,
"canopy_entropy" = can_entropy ,
"gap_fac_avg" =  gap_fac_avg ,
"vci_avg" = vci_avg,
"fhd" = fhd
)

}
```

3.2.1 Run Surface Metrics Function

```r
calc_met<- calculate_surf_metrics(lasf_clip = lasf_clip,min_ht = 1.37,cover_threshold = 2,
```

```
Delaunay rasterization[======--------------------------------------------] 12% (4 threads)
Delaunay rasterization[======--------------------------------------------] 13% (4 threads)
Delaunay rasterization[=======-------------------------------------------] 14% (4 threads)
Delaunay rasterization[=======-------------------------------------------] 15% (4 threads)
Delaunay rasterization[========------------------------------------------] 16% (4 threads)
Delaunay rasterization[========------------------------------------------] 17% (4 threads)
Delaunay rasterization[=========-----------------------------------------] 18% (4 threads)
Delaunay rasterization[=========-----------------------------------------] 19% (4 threads)
Delaunay rasterization[==========----------------------------------------] 20% (4 threads)
Delaunay rasterization[==========----------------------------------------] 21% (4 threads)
Delaunay rasterization[===========---------------------------------------] 22% (4 threads)
Delaunay rasterization[===========---------------------------------------] 23% (4 threads)
Delaunay rasterization[============--------------------------------------] 24% (4 threads)
Delaunay rasterization[============--------------------------------------] 25% (4 threads)
Delaunay rasterization[=============-------------------------------------] 26% (4 threads)
Delaunay rasterization[=============-------------------------------------] 27% (4 threads)
Delaunay rasterization[==============------------------------------------] 28% (4 threads)
Delaunay rasterization[==============------------------------------------] 29% (4 threads)
Delaunay rasterization[===============-----------------------------------] 30% (4 threads)
Delaunay rasterization[===============-----------------------------------] 31% (4 threads)
Delaunay rasterization[================----------------------------------] 32% (4 threads)
Delaunay rasterization[================----------------------------------] 33% (4 threads)
Delaunay rasterization[=================---------------------------------] 34% (4 threads)
Delaunay rasterization[=================---------------------------------] 35% (4 threads)
Delaunay rasterization[==================--------------------------------] 36% (4 threads)
Delaunay rasterization[==================--------------------------------] 37% (4 threads)
Delaunay rasterization[===================-------------------------------] 38% (4 threads)
Delaunay rasterization[===================-------------------------------] 39% (4 threads)
Delaunay rasterization[====================------------------------------] 40% (4 threads)
Delaunay rasterization[====================------------------------------] 41% (4 threads)
Delaunay rasterization[=====================-----------------------------] 42% (4 threads)
Delaunay rasterization[=====================-----------------------------] 43% (4 threads)
Delaunay rasterization[======================----------------------------] 44% (4 threads)
Delaunay rasterization[======================----------------------------] 45% (4 threads)
Delaunay rasterization[=======================---------------------------] 46% (4 threads)
Delaunay rasterization[=======================---------------------------] 47% (4 threads)
Delaunay rasterization[========================--------------------------] 48% (4 threads)
Delaunay rasterization[========================--------------------------] 49% (4 threads)
Delaunay rasterization[=========================-------------------------] 50% (4 threads)
Delaunay rasterization[=========================-------------------------] 51% (4 threads)
Delaunay rasterization[==========================------------------------] 52% (4 threads)
Delaunay rasterization[==========================------------------------] 53% (4 threads)
Delaunay rasterization[===========================-----------------------] 54% (4 threads)
Delaunay rasterization[===========================-----------------------] 55% (4 threads)
Delaunay rasterization[============================----------------------] 56% (4 threads)
Delaunay rasterization[============================----------------------] 57% (4 threads)
Delaunay rasterization[=============================---------------------] 58% (4 threads)
Delaunay rasterization[=============================---------------------] 59% (4 threads)
Delaunay rasterization[==============================--------------------] 60% (4 threads)
Delaunay rasterization[==============================--------------------] 61% (4 threads)
Delaunay rasterization[===============================-------------------] 62% (4 threads)
Delaunay rasterization[===============================-------------------] 63% (4 threads)
Delaunay rasterization[================================------------------] 64% (4 threads)
Delaunay rasterization[================================------------------] 65% (4 threads)
Delaunay rasterization[=================================-----------------] 66% (4 threads)
Delaunay rasterization[=================================-----------------] 67% (4 threads)
Delaunay rasterization[==================================----------------] 68% (4 threads)
Delaunay rasterization[==================================----------------] 69% (4 threads)
Delaunay rasterization[===================================---------------] 70% (4 threads)
Delaunay rasterization[===================================---------------] 71% (4 threads)
Delaunay rasterization[====================================--------------] 72% (4 threads)
Delaunay rasterization[====================================--------------] 73% (4 threads)
Delaunay rasterization[=====================================-------------] 74% (4 threads)
Delaunay rasterization[=====================================-------------] 75% (4 threads)
Delaunay rasterization[======================================------------] 76% (4 threads)
Delaunay rasterization[======================================------------] 77% (4 threads)
Delaunay rasterization[=======================================-----------] 78% (4 threads)
Delaunay rasterization[=======================================-----------] 79% (4 threads)
Delaunay rasterization[========================================----------] 80% (4 threads)
Delaunay rasterization[========================================----------] 81% (4 threads)
Delaunay rasterization[=========================================---------] 82% (4 threads)
Delaunay rasterization[=========================================---------] 83% (4 threads)
Delaunay rasterization[==========================================--------] 84% (4 threads)
Delaunay rasterization[==========================================--------] 85% (4 threads)
Delaunay rasterization[===========================================-------] 86% (4 threads)
Delaunay rasterization[===========================================-------] 87% (4 threads)
Delaunay rasterization[============================================------] 88% (4 threads)
Delaunay rasterization[============================================------] 89% (4 threads)
Delaunay rasterization[=============================================-----] 90% (4 threads)
Delaunay rasterization[=============================================-----] 91% (4 threads)
Delaunay rasterization[==============================================----] 92% (4 threads)
Delaunay rasterization[==============================================----] 93% (4 threads)
Delaunay rasterization[===============================================---] 94% (4 threads)
Delaunay rasterization[===============================================---] 95% (4 threads)
Delaunay rasterization[================================================--] 96% (4 threads)
Delaunay rasterization[================================================--] 97% (4 threads)
Delaunay rasterization[=================================================-] 98% (4 threads)
Delaunay rasterization[=================================================-] 99% (4 threads)
Delaunay rasterization[==================================================] 100% (4 threads)
Delaunay rasterization[=========================-------------------------] 50% (4 threads)
Delaunay rasterization[=========================-------------------------] 51% (4 threads)
Delaunay rasterization[==========================------------------------] 52% (4 threads)
Delaunay rasterization[==========================------------------------] 53% (4 threads)
Delaunay rasterization[===========================-----------------------] 54% (4 threads)
Delaunay rasterization[===========================-----------------------] 55% (4 threads)
Delaunay rasterization[============================----------------------] 56% (4 threads)
Delaunay rasterization[============================----------------------] 57% (4 threads)
Delaunay rasterization[=============================---------------------] 58% (4 threads)
Delaunay rasterization[=============================---------------------] 59% (4 threads)
Delaunay rasterization[==============================--------------------] 60% (4 threads)
Delaunay rasterization[==============================--------------------] 61% (4 threads)
Delaunay rasterization[===============================-------------------] 62% (4 threads)
Delaunay rasterization[===============================-------------------] 63% (4 threads)
Delaunay rasterization[================================------------------] 64% (4 threads)
Delaunay rasterization[================================------------------] 65% (4 threads)
Delaunay rasterization[=================================-----------------] 66% (4 threads)
Delaunay rasterization[=================================-----------------] 67% (4 threads)
Delaunay rasterization[==================================----------------] 68% (4 threads)
Delaunay rasterization[==================================----------------] 69% (4 threads)
Delaunay rasterization[===================================---------------] 70% (4 threads)
Delaunay rasterization[===================================---------------] 71% (4 threads)
Delaunay rasterization[====================================--------------] 72% (4 threads)
Delaunay rasterization[====================================--------------] 73% (4 threads)
Delaunay rasterization[=====================================-------------] 74% (4 threads)
Delaunay rasterization[=====================================-------------] 75% (4 threads)
Delaunay rasterization[======================================------------] 76% (4 threads)
Delaunay rasterization[======================================------------] 77% (4 threads)
Delaunay rasterization[=======================================-----------] 78% (4 threads)
Delaunay rasterization[=======================================-----------] 79% (4 threads)
Delaunay rasterization[========================================----------] 80% (4 threads)
Delaunay rasterization[========================================----------] 81% (4 threads)
Delaunay rasterization[=========================================---------] 82% (4 threads)
Delaunay rasterization[=========================================---------] 83% (4 threads)
Delaunay rasterization[==========================================--------] 84% (4 threads)
Delaunay rasterization[==========================================--------] 85% (4 threads)
Delaunay rasterization[===========================================-------] 86% (4 threads)
Delaunay rasterization[===========================================-------] 87% (4 threads)
Delaunay rasterization[============================================------] 88% (4 threads)
Delaunay rasterization[============================================------] 89% (4 threads)
Delaunay rasterization[=============================================-----] 90% (4 threads)
Delaunay rasterization[=============================================-----] 91% (4 threads)
Delaunay rasterization[==============================================----] 92% (4 threads)
Delaunay rasterization[==============================================----] 93% (4 threads)
Delaunay rasterization[===============================================---] 94% (4 threads)
Delaunay rasterization[===============================================---] 95% (4 threads)
Delaunay rasterization[================================================--] 96% (4 threads)
Delaunay rasterization[================================================--] 97% (4 threads)
Delaunay rasterization[=================================================-] 98% (4 threads)
Delaunay rasterization[=================================================-] 99% (4 threads)
Delaunay rasterization[==================================================] 100% (4 threads)
```

3.2.2 Display variables

```
row_var<- tidyr::pivot_longer(calc_met,cols = everything(),names_to = "Variable", values_t

row_var[1:102, 1]
```

```
# A tibble: 102 x 1
   Variable
   <chr>
 1 total_return
 2 total_first_return
 3 total_return_above_1.37
 4 Return_1_count_above_1.37
 5 Return_2_count_above_1.37
 6 Return_3_count_above_1.37
 7 Return_4_count_above_1.37
 8 Return_5_count_above_1.37
 9 Return_6_count_above_1.37
10 Return_7_count_above_1.37
# i 92 more rows
```

```
## all variables
row.names(calc_met)<- "value"
```

```
t(calc_met)
```

```
                                                 value
total_return                              1.182151e+07
total_first_return                        1.060510e+07
total_return_above_1.37                   1.654544e+06
Return_1_count_above_1.37                 1.436634e+06
Return_2_count_above_1.37                 2.122070e+05
Return_3_count_above_1.37                 5.685000e+03
Return_4_count_above_1.37                 1.800000e+01
Return_5_count_above_1.37                 0.000000e+00
Return_6_count_above_1.37                 0.000000e+00
Return_7_count_above_1.37                 0.000000e+00
Return_8_count_above_1.37                 0.000000e+00
Return_9_count_above_1.37                 0.000000e+00
h_min                                     1.370000e+00
h_max                                     2.632000e+01
h_mean                                    7.200000e+00
h_median                                  6.660000e+00
h_mode                                    5.380000e+00
h_sd                                      3.850000e+00
h_var                                     1.480000e+01
h_cv                                      5.347000e+01
h_kurtosis                                3.400000e+00
h_skwness                                 7.900000e-01
canopy_relief_ratio                       2.336673e-01
h_pct_01                                  1.540000e+00
h_pct_05                                  2.040000e+00
h_pct_10                                  2.630000e+00
h_pct_15                                  3.180000e+00
h_pct_20                                  3.700000e+00
h_pct_25                                  4.200000e+00
h_pct_30                                  4.690000e+00
h_pct_35                                  5.180000e+00
h_pct_40                                  5.660000e+00
h_pct_45                                  6.160000e+00
h_pct_50                                  6.660000e+00
h_pct_55                                  7.160000e+00
h_pct_60                                  7.690000e+00
h_pct_65                                  8.230000e+00
h_pct_70                                  8.830000e+00
h_pct_75                                  9.480000e+00
```

```
h_pct_80                                    1.025000e+01
h_pct_85                                    1.122000e+01
h_pct_90                                    1.251000e+01
h_pct_95                                    1.452000e+01
h_pct_98                                    1.681000e+01
h_pct_99                                    1.826000e+01
int_min                                     0.000000e+00
int_max                                     6.553500e+04
int_mean                                    1.402110e+03
int_median                                  8.580000e+02
int_mode                                    8.180000e+02
int_sd                                      2.024690e+03
int_var                                     4.099366e+06
int_cv                                      1.444000e+02
int_kurtosis                                1.035300e+02
int_skwness                                 8.140000e+00
int_pct_01                                  3.390000e+02
int_pct_05                                  4.280000e+02
int_pct_10                                  4.870000e+02
int_pct_15                                  5.380000e+02
int_pct_20                                  5.780000e+02
int_pct_25                                  6.180000e+02
int_pct_30                                  6.580000e+02
int_pct_35                                  6.980000e+02
int_pct_40                                  7.400000e+02
int_pct_45                                  7.980000e+02
int_pct_50                                  8.580000e+02
int_pct_55                                  9.350000e+02
int_pct_60                                  1.032000e+03
int_pct_65                                  1.137000e+03
int_pct_70                                  1.286000e+03
int_pct_75                                  1.481000e+03
int_pct_80                                  1.776000e+03
int_pct_85                                  2.202000e+03
int_pct_90                                  2.767000e+03
int_pct_95                                  3.453000e+03
int_pct_98                                  4.170000e+03
int_pct_99                                  1.220000e+04
pct_first_return_above_2                    1.305977e+01
pct_all_return_above_2                      1.139007e+02
pct_all_return_above_2_to_tot_first_return  1.334455e+01
first_return_above_2                        1.385002e+06
all_return_above_2                          1.577527e+06
```

```
pct_first_returns_above_mean                    6.221638e+00
pct_first_returns_above_mode                    9.103346e+00
pct_all_returns_above_mean                      5.885714e+00
pct_all_returns_above_mode                      8.798972e+00
pct_all_return_above_mean_to_tot_first_return 6.221638e+00
pct_all_return_above_mode_to_tot_first_return 9.103346e+00
first_return_above_mean                         6.598110e+05
first_returns_aobve_mean                        9.654190e+05
all_returns_above_mean                          6.957800e+05
all_returns_above_mode                          1.040171e+06
rumple_index                                    2.667330e+00
deep_gap                                        3.662147e+06
deep_gap_faction                                8.899939e-01
cover_fraction                                  1.100061e-01
top_rugosity                                    3.850000e+00
sd_sd                                           4.124294e-02
canopy_entropy                                  1.958582e-01
gap_fac_avg                                     9.956656e-01
vci_avg                                         1.958571e-01
fhd                                             6.366883e-01
```