Package: crownsegmentr (via r-universe)

May 12, 2026

Title Tree Crown Segmentation in Airborne LiDAR Point Clouds

Version 1.0.1

Maintainer Timon Miesner <timon.miesner@thuenen.de>

Description Provides a function that performs the adaptive mean shift algorithm for individual tree crown delineation in 3D point clouds as proposed by Ferraz et al. (2016) <doi:10.1016/j.rse.2016.05.028>, as well as supporting functions.

License GPL (>= 3) + file LICENSE

URL https://github.com/Lenostatos/crownsegmentr

Depends R (>= 4.0.0)

Imports assertthat, data.table, dbscan, lidR (>= 4.0.0), methods, Rcpp (>= 1.0.0), sf, terra

Suggests EBImage, future, testthat (>= 3.0.0), raster

LinkingTo BH (>= 1.75.0-0), progress, Rcpp (>= 1.0.0)

Config/testthat/edition 3

Contact timon.miesner@thuenen.de, Leon.Steinmeier@posteo.net, nikolai.knapp@thuenen.de

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements C++17, GNU make

Repository https://lenostatos.r-universe.dev

Date/Publication 2025-12-02 08:55:02 UTC

RemoteUrl https://github.com/lenostatos/crownsegmentr

RemoteRef HEAD

RemoteSha ec0ce0996557e250a625f2c52390cf4cb4d91b9a

Contents

assert_that_raster_covers_data_frame_point_cloud . . . 2
assert_that_raster_covers_las_point_cloud . . . 3
calculate_centroids_flexible . . . 3
extract_coordinate_values . . . 6
li_diameter_raster . . . 7
li_diameter_raster,LAS-method . . . 8
match_any . . . 10
remove_small_trees . . . 11
segment_tree_crowns . . . 13
segment_tree_crowns_core . . . 19
validate_scale_n_offset_are_consistent . . . 22
validate_write_crown_id_also_to_file_for_LAScatalogs . . . 23

Index **24**

`assert_that_raster_covers_data_frame_point_cloud`
Assert that the extent of a raster covers that of a data.frame point cloud

---

Description

Assert that the extent of a raster covers that of a data.frame point cloud

Usage

```
assert_that_raster_covers_data_frame_point_cloud(
  raster,
  data_frame_point_cloud,
  message
)
```

Arguments

`raster`	A SpatRaster.
`data_frame_point_cloud`	Point cloud data in `data.frame()` format.
`message`	Length-one character vector. Message to be used on assertion failure.

assert_that_raster_covers_las_point_cloud

Assert that the extent of a raster covers that of a LAS point cloud

Description

Assert that the extent of a raster covers that of a LAS point cloud

Usage

```
assert_that_raster_covers_las_point_cloud(raster, las_point_cloud, message)
```

Arguments

raster	A SpatRaster.
las_point_cloud	Point cloud data in lidR::LAS format.
message	Length-one character vector. Message to be used on assertion failure.

calculate_centroids_flexible

Searches modes with the AMS3D algorithm for a lidar point cloud of a forest

Description

Employs the 3D adaptive mean shift algorithm (Ferraz et al., 2016) to estimate the mode of each point in a point cloud which is assumed to contain trees. In this context the mode is a theoretical "center of mass" of a tree crown point cloud, that is usually located shortly below the crown apex.

Usage

```
calculate_centroids_flexible(
  coordinate_table,
  min_point_height_above_ground,
  ground_height_data,
  crown_diameter_to_tree_height_data,
  crown_length_to_tree_height_data,
  crown_diameter_constant,
  crown_length_constant,
  centroid_convergence_distance,
  max_iterations_per_point,
  also_return_all_centroids,
  show_progress_bar
)
```

```
calculate_centroids_normalized(
  coordinate_table,
  min_point_height_above_ground,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  centroid_convergence_distance,
  max_iterations_per_point,
  also_return_all_centroids,
  show_progress_bar
)

calculate_centroids_terraneous(
  coordinate_table,
  min_point_height_above_ground,
  ground_height_grid_data,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  centroid_convergence_distance,
  max_iterations_per_point,
  also_return_all_centroids,
  show_progress_bar
)
```

Arguments

`coordinate_table`
: A `data.frame`. The first three columns are treated as the x-, y-, and z-coordinates of an airborne lidar point cloud.

`min_point_height_above_ground`
: A single positive number. The minimum point height above ground at which the function will calculate centroids.

`ground_height_data`
: A list containing either a single ground height value (named "value") or a set of elements that make up a ground height raster covering the whole area of the point cloud. Such a set has to consist of the named elements described in the section "Raster argument structure" below.

`crown_diameter_to_tree_height_data`
: A list containing either a single numeric value (named "value") or the data for a raster of values (see section "Raster argument structure" below for how the raster data has to be stored in the list). The values indicate the estimated ratio of crown diameter to tree height for the whole plot or individual raster pixels respectively.

`crown_length_to_tree_height_data`
: A list containing either a single numeric value (named "value") or the data for a

raster of values (see section "Raster argument structure" below for how the raster data has to be stored in the list). The values indicate the estimated ratio of crown height to tree height for the whole plot or individual raster pixels respectively.

`crown_diameter_constant`
: Single number >=0. Intercept for the linear function determining the kernel diameter (bandwidth) in relationship to the height above ground.

`crown_length_constant`
: Single number >=0. Intercept for the linear function determining the kernel height (bandwidth) in relationship to the height above ground.

`centroid_convergence_distance`
: Numeric Scalar. Distance at which it is assumed that subsequently calculated centroids have converged to the nearest mode.

`max_iterations_per_point`
: Integer Scalar. Maximum number of centroids calculated before the search for the nearest mode stops.

`also_return_all_centroids`
: Boolean Scalar. Should all centroid coordinates be returned as well?

`show_progress_bar`
: Boolean Scalar. Should a progress bar be shown during the computation?

`crown_diameter_to_tree_height`, `crown_length_to_tree_height`
: Single numbers. Determine the size of the search kernel (bandwidth) of the algorithm, as a function of height above ground. The kernel should have roughly the size of the expected tree crowns. If the intercepts are zero, the slopes translate to ratios of crown diameter to tree height or crown length to tree height, respectively.

`ground_height_grid_data`
: A list containing a set of elements that make up a ground height raster covering the whole area of the point cloud. The set has to consist of the named elements described in the section "Raster argument structure" below.

Value

A list with either one or two elements:

- The first element (named "terminal_coordinates") contains the terminal centroids for all points in the `coordinate_table`. These are stored in a `data.frame` with three columns that hold the x-, y-, and z-coordinates and they are stored in the same order as their respective points in the `coordinate_table`.
- The second element (named "centroid_coordinates") is only present if `also_return_all_centroids` was set to `TRUE` and contains the centroids calculated during the mode finding process. The prior centroids are stored in a `data.frame` with xyz-coordinate columns like the terminal centroids. To enable grouping of these centroids by the point they belong to, there is one additional column (named "point_index") which holds row indices of the corresponding points in the `coordinate_table`.

Functions

- `calculate_centroids_flexible()`: Can take either a single value or raster data for both the ground height and the `crown_diameter_to_tree_height` and `crown_length_to_tree_height` parameters.
- `calculate_centroids_terraneous()`: Use a ground height raster to find modes in a non-normalized point cloud.

Raster argument structure

Raster data has to be passed as a list comprising the following named elements:

- values: Numeric vector holding the values.
- num_rows: Integer number indicating the number of rows.
- num_cols: Integer number indicating the number of columns.
- x_min: Number indicating the lowest x coordinate covered.
- x_max: Number indicating the largest x coordinate covered.
- y_min: Number indicating the lowest y coordinate covered.
- y_max: Number indicating the largest y coordinate covered.

References

Ferraz, A., S. Saatchi, C. Mallet, and V. Meyer (2016) *Lidar detection of individual tree size in tropical forests*. Remote Sensing of Environment 183:318–333. doi:10.1016/j.rse.2016.05.028.

`extract_coordinate_values` *Extract coordinate data from a data.frame-like object*

Description

This function extracts three numeric columns from the input table. If possible, columns which are named x/X, y/Y, or z/Z.

Usage

```
extract_coordinate_values(coordinate_table)
```

Arguments

`coordinate_table`
: An object which is valid according to `validate_coordinate_table()` (i.e. data.frame-like and contains at least three numeric columns).

Value

A `base::data.frame()` with just three columns that are expected to hold the x-, y-, and z-coordinates in that order.

`li_diameter_raster` *Calculate a raster of crown diameter for tree height for AMS3D*

Description

The function calculates a raster with values for crown_diameter_to_tree_height as input for the AMS3D algorithm. It segments the tree crowns with the Li2012 algorithm, calculates a ratio of crown diameter to tree height for each tree, and converts this into a raster.

Usage

```
li_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)
```

Arguments

`point_cloud`	the input point cloud, either as LAS or as data.frame.
`crown_diameter_constant`	a fixed value for crown_diameter_constant, which reduces the crown diameters by the given value before calculating the ratio of crown diameter to tree height
`limits`	a numeric vector with minimum and maximum values for the ratio, at which every tree's ratio will be capped
`ground_height`	(optional) either • NULL, indicating that the point cloud is normalized, or • a SpatRaster digital terrain model, or • a list of arguments to the lidR rasterize_terrain() function to normalize the point cloud.
`smoothing_radius`	The radius of the filter used for smoothing the diameter-to-height ratio from individual trees.
`...`	further parameters will be passed to the function `lidR::li2012()`

Value

terra SpatRaster

Details

The output raster can serve as input for the parameter "crown_diameter_to_tree_height" for the function segment_tree_crowns. It averages the ratio of crown diameter to tree height for a given radius, for trees that were detected with the Li2012 tree segmentation algorithm.

li_diameter_raster,LAS-method *Calculate a raster of crown diameter to tree height using watershed segmentation*

Description

The function calculates a raster with values for crown_diameter_to_tree_height as input for the AMS3D algorithm. It segments the tree crowns with the watershed algorithm from lidR, calculates a ratio of crown diameter to tree height for each tree, and converts this into a raster.

Usage

```
## S4 method for signature 'LAS'
li_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

## S4 method for signature 'data.frame'
li_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

## S4 method for signature 'LAScatalog'
li_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

watershed_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
```

```
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

## S4 method for signature 'LAS'
watershed_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

## S4 method for signature 'data.frame'
watershed_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)

## S4 method for signature 'LAScatalog'
watershed_diameter_raster(
  point_cloud,
  crown_diameter_constant = 0,
  limits = c(0, 1),
  ground_height = NULL,
  smoothing_radius = 5,
  ...
)
```

Arguments

point_cloud	the input point cloud, either as LAS or as data.frame.
crown_diameter_constant	a fixed value for crown_diameter_constant, which reduces the crown diameters by the given value before calculating the ratio of crown diameter to tree height
limits	a numeric vector with minimum and maximum values for the ratio, at which every tree's ratio will be capped
ground_height	(optional) either • NULL, indicating that the point cloud is normalized, or • a SpatRaster digital terrain model, or • a list of arguments to the lidR rasterize_terrain() function to normalize the point cloud.

`smoothing_radius`
: The radius of the filter used for smoothing the diameter-to-height ratio from individual trees.

`...`
: further parameters will be passed to the function `lidR::watershed()`

Value

a terra SpatRaster

Functions

- `li_diameter_raster(LAS)`: Calculate a raster of crown diameter for tree height using li2012 segmentation
- `li_diameter_raster(data.frame)`: Calculate a raster of crown diameter for tree height using li2012 segmentation
- `li_diameter_raster(LAScatalog)`: Calculate a raster of crown diameter for tree height using li2012 segmentation
- `watershed_diameter_raster(LAS)`: Calculate a raster of crown diameter for tree height using watershed segmentation
- `watershed_diameter_raster(data.frame)`: Calculate a raster of crown diameter for tree height using watershed segmentation
- `watershed_diameter_raster(LAScatalog)`: Calculate a raster of crown diameter for tree height using watershed segmentation

Details

The output raster can serve as input for the parameter "crown_diameter_to_tree_height" for the function segment_tree_crowns. It averages the ratio of crown diameter to tree height for a given radius, for trees that were detected with watershed segmentation. The Bioconductor package "EBImage" is required to use this function.

`match_any` *Find all exact matches with at least one of the provided patterns*

Description

Find all exact matches with at least one of the provided patterns

Usage

```
match_any(patterns, targets)
```

Arguments

`patterns`	Objects which will be matched to `targets` via the == operator.
`targets`	Objects which will be matched to each of the `patterns`.

Value

A boolean vector of the same length as `targets`.

`remove_small_trees` *Remove small clusters from segmented point cloud*

Description

The function takes a point cloud in which trees were segmented, and removes tree clusters that are smaller than a certain radius or a certain height

Usage

```
remove_small_trees(
  point_cloud,
  min_radius = 1,
  min_height = -Inf,
  crown_id_column_name = "crown_id"
)

## S4 method for signature 'data.frame'
remove_small_trees(
  point_cloud,
  min_radius = 1,
  min_height = -Inf,
  crown_id_column_name = "crown_id"
)

## S4 method for signature 'LAS'
remove_small_trees(
  point_cloud,
  min_radius = 1,
  min_height = -Inf,
  crown_id_column_name = "crown_id"
)

## S4 method for signature 'LAScatalog'
remove_small_trees(
  point_cloud,
  min_radius = 1,
  min_height = -Inf,
  crown_id_column_name = "crown_id"
)
```

Arguments

point_cloud	a point cloud, either as data.frame/data.table, or as lidR::LAS object.
min_radius	(Numeric >= 0) the threshold for crown radius, below which trees will be removed
min_height	(Numeric) the threshold for crown height, below which trees will be removed. Works only if las is normalized.
crown_id_column_name	the name of the column in which the id of the crown is saved

Value

lidR LAS

Functions

- `remove_small_trees(data.frame)`: removes small tree clusters in a segmented LAS object.
- `remove_small_trees(LAS)`: removes small tree clusters in a segmented LAS object.
- `remove_small_trees(LAScatalog)`: removes small tree clusters in a segmented LAScatalog.

Details

returns the same las object that was given as input, but with altered crown id's. Trees that are considered too small have their crown id set to NA, and all other crown id's are re-assigned so that they are without gaps

Examples

```
# Preparation ------------------------------------------------

# Load a point cloud of some trees included in the lidR package
point_cloud <- lidR::readLAS(system.file(
  "extdata/MixedConifer.laz",
  package = "lidR"
))

# Set up a plotting function for segmented point clouds
plot_segmented_point_cloud <- function(
    point_cloud) {

  # Generate random crown colors
  crown_colors <- lidR::pastel.colors(
    n = length(unique(point_cloud@data[["crown_id"]])))

  # Plot the segmented crown bodies
  lidR::plot(
    point_cloud,
    color = "crown_id",
    pal = crown_colors,
```

```
    nbreaks = length(crown_colors),
    size = 3,
    axis = TRUE
  )
}

# Usage workflow ----------------------------------------------------------

# Segment tree crowns
segmented_point_cloud <- segment_tree_crowns(
  point_cloud,
  crown_diameter_to_tree_height = 0.2,
  crown_length_to_tree_height = 0.5)

# Plot the segmented point cloud
plot_segmented_point_cloud(segmented_point_cloud)

# Remove small trees
processed_point_cloud_1 <- remove_small_trees(segmented_point_cloud)

# Plot the result
plot_segmented_point_cloud(processed_point_cloud_1)


# Vary some arguments -----------------------------------------------------

# increase crown radius threshold
processed_point_cloud_2 <- remove_small_trees(
  segmented_point_cloud,
  min_radius = 2)

# Plot the result
plot_segmented_point_cloud(processed_point_cloud_2)


# increase the height threshold
processed_point_cloud_3 <- remove_small_trees(
  segmented_point_cloud,
  min_height = 20)

# Plot the result
plot_segmented_point_cloud(processed_point_cloud_3)
```

`segment_tree_crowns` *Segment Tree Crowns in a 3D Point Cloud*

Description

Employs a variant of the mean shift algorithm (Ferraz et. al, 2016) and after that the DBSCAN algorithm in order to identify tree crowns in airborne lidar data.

Usage

```
segment_tree_crowns(
  point_cloud,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant = 0,
  crown_length_constant = 0,
  segment_crowns_only_above = 0,
  ground_height = NULL,
  crown_id_column_name = "crown_id",
  centroid_convergence_distance = 0.01,
  max_iterations_per_point = 500,
  dbscan_neighborhood_radius = 0.3,
  min_num_points_per_crown = 5,
  ...
)

## S4 method for signature 'data.frame'
segment_tree_crowns(
  point_cloud,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  segment_crowns_only_above,
  ground_height,
  crown_id_column_name,
  centroid_convergence_distance,
  max_iterations_per_point,
  dbscan_neighborhood_radius,
  min_num_points_per_crown,
  verbose = TRUE,
  also_return_terminal_centroids = FALSE,
  also_return_all_centroids = FALSE
)

## S4 method for signature 'LAS'
segment_tree_crowns(
  point_cloud,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  segment_crowns_only_above,
  ground_height,
  crown_id_column_name,
  centroid_convergence_distance,
  max_iterations_per_point,
```

```
  dbscan_neighborhood_radius,
  min_num_points_per_crown,
  verbose = TRUE,
  also_return_terminal_centroids = FALSE,
  also_return_all_centroids = FALSE,
  write_crown_id_also_to_file = FALSE,
  crown_id_file_description = crown_id_column_name
)

## S4 method for signature 'LAScatalog'
segment_tree_crowns(
  point_cloud,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  segment_crowns_only_above,
  ground_height,
  crown_id_column_name,
  centroid_convergence_distance,
  max_iterations_per_point,
  dbscan_neighborhood_radius,
  min_num_points_per_crown,
  write_crown_id_also_to_file = TRUE,
  crown_id_file_description = crown_id_column_name
)
```

Arguments

`point_cloud` A data set containing xyz-coordinates. Can be passed as either a data.frame, a data.table, a LAS object or a LAScatalog.
If it's a data.frame or a data.table the function searches for coordinate columns by looking for the first numeric columns named "x"/"X", "y"/"Y", or "z"/"Z". For each instance where it can't find one of those it selects the next available numeric column in the table and issues a warning.

`crown_diameter_to_tree_height` Single number or SpatRasters covering the area of the `point_cloud`. The diameter of the search kernel will be calculated by multiplying this value and the height above ground of the kernel center, and adding the crown_diameter_constant. For details see "How the algorithm works". Points will not be segmented wherever a raster contains `NA` values.

`crown_length_to_tree_height` Single number or SpatRasters covering the area of the `point_cloud`. The height of the search kernel will be calculated by multiplying this value and the height above ground of the kernel center, and adding the crown_length_constant. For details see "How the algorithm works". Points will not be segmented wherever a raster contains `NA` values.

`crown_diameter_constant, crown_length_constant` Single number >=0. Used to determine the dimensions of the search kernel,

together with the respective ratios to tree height. For details see "How the algorithm works".

`segment_crowns_only_above`
: A single positive number denoting the minimum height above ground at which crown IDs will be calculated.

  Note that points directly below this threshold will still be considered during the segmentation if they are within reach of search kernels constructed at the `segment_crowns_only_above` height. See "How the algorithm works" to learn about the search kernels.

`ground_height`
: One of

  - `NULL`, indicating that `point_cloud` is normalized with ground height at zero.
  - A SpatRaster providing ground heights for the area of the (not normalized) `point_cloud`.
  - A list of (ideally named) arguments to the lidR rasterize_terrain() function, which will be used to generate a ground height grid from `point_cloud`. Currently not supported with point clouds stored in data.frames. The list should not contain an argument to the "las" parameter of rasterize_terrain().

  Points will not be segmented wherever ground heights are NA.

`crown_id_column_name`
: A character string. The column or attribute name under which IDs for segmented bodies should be stored.

`centroid_convergence_distance`
: A single number. Distance at which it is assumed that subsequently calculated centroids have converged to the nearest mode. See "How the algorithm works" to learn about centroids and modes in the context of the AMS3D algorithm.

`max_iterations_per_point`
: A single integer. Maximum number of centroids calculated before the search for the nearest mode stops. See "How the algorithm works" to learn about centroids and modes in the context of the AMS3D algorithm.

`dbscan_neighborhood_radius`
: A single number. Radius for the spherical DBSCAN neighborhood around a mode. See "How the algorithm works" to learn about neighborhoods in the context of the DBSCAN algorithm.

`min_num_points_per_crown`
: A single integer. The minimum number of converged centroids within a DBSCAN neighborhood at which the centroid in the neighborhood's center will be treated as a core point. See "How the algorithm works" to learn about neighborhoods and core points in the context of the DBSCANb algorithm.

`...`
: Unused.

`verbose`
: `TRUE` or `FALSE`. Should the function show a progress bar and other runtime information in the console?

`also_return_terminal_centroids`
: `TRUE` or `FALSE`. Should mode coordinates be returned as well?

`also_return_all_centroids`
: `TRUE` or `FALSE`. Should all centroid coordinates be returned as well? This slows down processing by a little bit and will return a data set which requires at least ~10 times more memory than the input point cloud.

`write_crown_id_also_to_file`
: `TRUE` or `FALSE`. When writing the returned LAS object to disk, should the IDs of segmented bodies be written into that file as well? See the lidR function add_lasattribute() for additional details. Will also be used for all attributes of the LAS object(s) which are returned if `also_return_terminal_centroids` and/or `also_return_all_centroids` were set to `TRUE`.

  For LAScatalogs, this is only used if the result is returned as a LAS object in memory. If the LAScatalog is set up to write the segmented point clouds into files, the IDs of segmented bodies will always be written to these files as well.

`crown_id_file_description`
: A character string. If `write_crown_id_also_to_file` is set to `TRUE` this will be used as an additional description of the IDs of segmented bodies when the LAS object is written to disk. See the "desc" parameter of the lidR function add_lasattribute() for additional details.

Value

The point cloud which was passed to the function but extended with a column/attribute holding for each point the ID of a segmented body. IDs with the value `NA` indicate that a point was not assigned to any body.

If `also_return_terminal_centroids` and/or `also_return_all_centroids` were set to `TRUE`, a list with at most three named elements in the following order:

segmented_point_cloud The segmented point cloud which would have been returned directly if `also_return_terminal_centroids` and `also_return_all_centroids` had been set to `FALSE`.

terminal_centroids If `also_return_terminal_centroids` was set to `TRUE`, a point cloud of the same type as the input point cloud holding the terminal centroids calculated with the AMS3D algorithm and two additional columns/attributes. One of these columns/attributes holds IDs of the segmented bodies that the modes belong to and the other (named "point_index") holds indices to the points in the input point cloud.

centroids If `also_return_all_centroids` was set to `TRUE`, a point cloud of the same type as the input point cloud holding the centroids calculated with the AMS3D algorithm and two additional columns/attributes. One of these columns/attributes holds IDs of the segmented bodies that the centroids belong to and the other (named "point_index") holds indices to the points in the input point cloud.

The method for LASCatalogs works just like any other lidR function that accepts them, i.e. it returns either an in-memory LAS object or writes the processed chunks to individual files and returns those file's names. Please refer to the LASCatalog documentation for more details.

Functions

- `segment_tree_crowns(data.frame)`: Segments coordinates stored as three columns in a data.frame or data.table.

- `segment_tree_crowns(LAS)`: Segments the point cloud data of a LAS object.
- `segment_tree_crowns(LAScatalog)`: Segments the point cloud data of a LAScatalog. This method does not support additionally returning centroids. Instead of the verbose parameter use the LAScatalog's progress option (see the LAScatalog documentation -> "Processing options" -> "progress").

How the algorithm works

The basic assumption is that tree crowns form local maxima of point density and height within lidar point clouds. These local maxima are called *modes*. The algorithm tries to find the nearest mode for each point. This is done by looking at the surrounding points and moving into the direction of the highest point density until the nearest mode is (almost) reached.

The surrounding points are found with a search kernel (a three-dimensional search window) which has the shape of a vertical cylinder. According to literature, the algorithm works best if the search kernel has roughly the size of the surrounding crowns. Therefore, the parameters controlling the kernels dimension are simplistically called `crown_diameter_to_tree_height`, `crown_diameter_constant`, and `crown_lenght...` respectively. The diameter of the kernel is calculated from the height above ground of the kernels center times the value for crown_diameter_to_tree_height, plus the crown_diameter constant. The height of the kernel is calculated respectively.

The direction of the highest point density is found by calculating the average position of all points within the cylinder, the cylinder's so called *centroid*. In order to move further into the direction of the highest point density, a new cylinder is placed on the centroid and a new centroid is calculated for that cylinder. This continues on until the cylinders "stop moving", i.e. until two subsequently calculated centroids are closer to each other than `centroid_convergence_distance`. At this point, the most recently calculated centroid, hence called 'terminal centroid', is assumed to be close enough to the mode, so that the original point can be linked to the respective tree top.

It sometimes happens that centroids converge only after a lot of iterations. In order to prevent situations where an excessive number of centroids is calculated for just one point, the parameter `max_iterations_per_point` is used to stop the centroid calculations after a certain number of them has been performed. Nonetheless, the last centroid found before stopping is still taken as a good enough guess of the nearest mode's position.

After the terminal centroids of the individual points have been calculated, it can be seen that terminal centroids of points belonging to the same tree crown are positioned very close to each other, shortly below the crown's apex. These dense clusters of terminal centroids are identified with the DBSCAN algorithm which assigns a cluster ID to every one of them. The cluster IDs are then finally connected back to the points of the point cloud and used as crown IDs.

The DBSCAN clustering is explained nicely in Wikipedia but here is a quick sketch of what it does: The DBSCAN algorithm classifies points as either core points, border points, or noise and assigns core and border points to the same cluster if they are close enough to at least one other core point of the cluster.

In order to be core points, points need to have enough neighbors. The parameter `dbscan_neighborhood_radius` determines the radius of the neighborhood and the parameter `min_num_points_per_crown` determines the minimum number of points in the neighborhood (including the to-be-classified one), which are needed for a core point.

Border points are within the neighborhood of core points but don't have enough neighbors to be core points themselves. Noise points are not within the neighborhood of any core point and also don't have enough neighbors to be core points.

Clusters are identified by iterating over the points and classifying them one by one. For each point the neighborhood is scanned and the point is classified accordingly. If the point is a core or border point, the neighboring points are classified next. As long as it is possible to directly connect to new core or border points in this way, the same cluster ID is assigned to each encountered point.

References

Ferraz, A., S. Saatchi, C. Mallet, and V. Meyer (2016) *Lidar detection of individual tree size in tropical forests*. Remote Sensing of Environment 183:318–333. doi:10.1016/j.rse.2016.05.028

Ferraz, A., F. Bretar, S. Jaquemond, G. Gonçalves, L. Pereira, M. Tomé, and P. Soares (2012) *3-D mapping of a multi-layered Mediteranean forest using ALS data*. Remote Sensing of Environment, 121:210-223. doi:10.1016/j.rse.2012.01.020

`segment_tree_crowns_core`

Calls the C++ back-end and the DBSCAN algorithm to perform the segmentation

---

Description

This functions is meant to be used internally by methods of the `segment_tree_crowns` generic.

Usage

```
segment_tree_crowns_core(
  coordinate_table,
  segment_crowns_only_above,
  ground_height,
  crown_diameter_to_tree_height,
  crown_length_to_tree_height,
  crown_diameter_constant,
  crown_length_constant,
  verbose,
  centroid_convergence_distance,
  max_iterations_per_point,
  dbscan_neighborhood_radius,
  min_num_points_per_crown,
  also_return_terminal_centroids,
  also_return_all_centroids
)
```

Arguments

`coordinate_table`

A data.frame or data.table which is a valid coordinate table according to `validate_coordinate_table`.

`segment_crowns_only_above`
: A single positive number denoting the minimum height above ground at which crown IDs will be calculated.

  Note that points directly below this threshold will still be considered during the segmentation if they are within reach of search kernels constructed at the `segment_crowns_only_above` height. See "How the algorithm works" to learn about the search kernels.

`ground_height`
: One of

  - `NULL`, indicating that the point cloud stored in `coordinate_table` is normalized with ground height at zero.
  - A SpatRaster providing ground heights for the area of the (not normalized) point cloud stored in `coordinate_table`.

`crown_diameter_to_tree_height`
: Single number or SpatRasters covering the area of the `point_cloud`. The diameter of the search kernel will be calculated by multiplying this value and the height above ground of the kernel center, and adding the crown_diameter_constant. For details see "How the algorithm works". Points will not be segmented wherever a raster contains `NA` values.

`crown_length_to_tree_height`
: Single number or SpatRasters covering the area of the `point_cloud`. The height of the search kernel will be calculated by multiplying this value and the height above ground of the kernel center, and adding the crown_length_constant. For details see "How the algorithm works". Points will not be segmented wherever a raster contains `NA` values.

`crown_diameter_constant`, `crown_length_constant`
: Single number >=0. Used to determine the dimensions of the search kernel, together with the respective ratios to tree height. For details see "How the algorithm works".

`verbose`
: `TRUE` or `FALSE`. Should the function show a progress bar and other runtime information in the console?

`centroid_convergence_distance`
: A single number. Distance at which it is assumed that subsequently calculated centroids have converged to the nearest mode. See "How the algorithm works" to learn about centroids and modes in the context of the AMS3D algorithm.

`max_iterations_per_point`
: A single integer. Maximum number of centroids calculated before the search for the nearest mode stops. See "How the algorithm works" to learn about centroids and modes in the context of the AMS3D algorithm.

`dbscan_neighborhood_radius`
: A single number. Radius for the spherical DBSCAN neighborhood around a mode. See "How the algorithm works" to learn about neighborhoods in the context of the DBSCAN algorithm.

`min_num_points_per_crown`
: A single integer. The minimum number of converged centroids within a DBSCAN neighborhood at which the centroid in the neighborhood's center will be treated as a core point. See "How the algorithm works" to learn about neighborhoods and core points in the context of the DBSCANb algorithm.

`also_return_terminal_centroids`
: `TRUE` or `FALSE`. Should mode coordinates be returned as well?

`also_return_all_centroids`
: `TRUE` or `FALSE`. Should all centroid coordinates be returned as well? This slows down processing by a little bit and will return a data set which requires at least ~10 times more memory than the input point cloud.

Value

A list with at most three elements:

crown_ids A vector of IDs of segmented bodies.

terminal_coordinates If `also_return_terminal_centroids` was set to `TRUE`, a data.table with mode coordinates as the second list element. The table has two additional columns:

- **crown_id** Holds the IDs also returned with the first list element.
- **point_index** Holds row indices of the original points in the input `coordinate_table`.

centroid_coordinates If `also_return_all_centroids` was set to `TRUE`, a data.table with centroid coordinates as the last list element. The table has two additional columns:

- **crown_id** Holds the IDs also returned with the first list element.
- **point_index** Holds row indices of the original points in the input `coordinate_table`.

How the algorithm works

The basic assumption is that tree crowns form local maxima of point density and height within lidar point clouds. These local maxima are called *modes*. The algorithm tries to find the nearest mode for each point. This is done by looking at the surrounding points and moving into the direction of the highest point density until the nearest mode is (almost) reached.

The surrounding points are found with a search kernel (a three-dimensional search window) which has the shape of a vertical cylinder. According to literature, the algorithm works best if the search kernel has roughly the size of the surrounding crowns. Therefore, the parameters controlling the kernels dimension are simplistically called `crown_diameter_to_tree_height`, `crown_diameter_constant`, and `crown_lenght...` respectively. The diameter of the kernel is calculated from the height above ground of the kernels center times the value for crown_diameter_to_tree_height, plus the crown_diameter constant. The height of the kernel is calculated respectively.

The direction of the highest point density is found by calculating the average position of all points within the cylinder, the cylinder's so called *centroid*. In order to move further into the direction of the highest point density, a new cylinder is placed on the centroid and a new centroid is calculated for that cylinder. This continues on until the cylinders "stop moving", i.e. until two subsequently calculated centroids are closer to each other than `centroid_convergence_distance`. At this point, the most recently calculated centroid, hence called 'terminal centroid', is assumed to be close enough to the mode, so that the original point can be linked to the respective tree top.

It sometimes happens that centroids converge only after a lot of iterations. In order to prevent situations where an excessive number of centroids is calculated for just one point, the parameter `max_iterations_per_point` is used to stop the centroid calculations after a certain number of them has been performed. Nonetheless, the last centroid found before stopping is still taken as a good enough guess of the nearest mode's position.

After the terminal centroids of the individual points have been calculated, it can be seen that terminal centroids of points belonging to the same tree crown are positioned very close to each other, shortly below the crown's apex. These dense clusters of terminal centroids are identified with the DBSCAN algorithm which assigns a cluster ID to every one of them. The cluster IDs are then finally connected back to the points of the point cloud and used as crown IDs.

The DBSCAN clustering is explained nicely in Wikipedia but here is a quick sketch of what it does: The DBSCAN algorithm classifies points as either core points, border points, or noise and assigns core and border points to the same cluster if they are close enough to at least one other core point of the cluster.

In order to be core points, points need to have enough neighbors. The parameter `dbscan_neighborhood_radius` determines the radius of the neighborhood and the parameter `min_num_points_per_crown` determines the minimum number of points in the neighborhood (including the to-be-classified one), which are needed for a core point.

Border points are within the neighborhood of core points but don't have enough neighbors to be core points themselves. Noise points are not within the neighborhood of any core point and also don't have enough neighbors to be core points.

Clusters are identified by iterating over the points and classifying them one by one. For each point the neighborhood is scanned and the point is classified accordingly. If the point is a core or border point, the neighboring points are classified next. As long as it is possible to directly connect to new core or border points in this way, the same cluster ID is assigned to each encountered point.

`validate_scale_n_offset_are_consistent`
Asserts that all files referenced by a LAScatalog have the same scale and offset values.

Description

Asserts that all files referenced by a LAScatalog have the same scale and offset values.

Usage

```
validate_scale_n_offset_are_consistent(LAScatalog)
```

Arguments

`LAScatalog`	The LAScatalog to be tested.

`validate_write_crown_id_also_to_file_for_LAScatalogs` *Ensures that crown IDs are written to output files of the LAScatalog*

Description

Issues a warning if the user wanted to write the output to files but not store IDs of segmented bodies.

Usage

```
validate_write_crown_id_also_to_file_for_LAScatalogs(
  write_crown_id_also_to_file,
  LAScatalog
)
```

Arguments

`write_crown_id_also_to_file`	The to-be-validated parameter.
`LAScatalog`	The LAScatalog whose settings are compared to the value of `write_crown_id_also_to_file`.

Value

A possibly corrected value for `write_crown_id_also_to_file`.

Index

assert_that_raster_covers_data_frame_point_cloud, 2
assert_that_raster_covers_las_point_cloud, 3

base::data.frame(), *6*

calculate_centroids_flexible, 3
calculate_centroids_normalized (calculate_centroids_flexible), 3
calculate_centroids_terraneous (calculate_centroids_flexible), 3

data.frame, *15*, *17*, *19*
data.frame(), *2*
data.frames, *16*
data.table, *15*, *17*, *19*, *21*

extract_coordinate_values, 6

LAS object, *12*, *15*, *17, 18*
LAS object(s), *17*
LASCatalog, *17*
LAScatalog, *12*, *15*, *18*, *22, 23*
LAScatalog documentation, *18*
LASCatalogs, *17*
LAScatalogs, *17*
li_diameter_raster, 7
li_diameter_raster,data.frame-method (li_diameter_raster,LAS-method), 8
li_diameter_raster,LAS-method, 8
li_diameter_raster,LAScatalog-method (li_diameter_raster,LAS-method), 8
lidR function add_lasattribute(), *17*
lidR rasterize_terrain(), *7*, *9*, *16*
lidR::LAS, *3*
lidR::li2012(), *7*
lidR::watershed(), *10*

match_any, 10

rasterize_terrain(), *16*
remove_small_trees, 11
remove_small_trees,data.frame-method (remove_small_trees), 11
remove_small_trees,LAS-method (remove_small_trees), 11
remove_small_trees,LAScatalog-method (remove_small_trees), 11

segment_tree_crowns, 13
segment_tree_crowns,data.frame-method (segment_tree_crowns), 13
segment_tree_crowns,LAS-method (segment_tree_crowns), 13
segment_tree_crowns,LAScatalog-method (segment_tree_crowns), 13
segment_tree_crowns_core, 19
SpatRaster, *2, 3*, *7*, *9*, *16*, *20*
SpatRasters, *15*, *20*

validate_scale_n_offset_are_consistent, 22
validate_write_crown_id_also_to_file_for_LAScatalogs, 23

watershed_diameter_raster (li_diameter_raster,LAS-method), 8
watershed_diameter_raster,data.frame-method (li_diameter_raster,LAS-method), 8
watershed_diameter_raster,LAS-method (li_diameter_raster,LAS-method), 8
watershed_diameter_raster,LAScatalog-method (li_diameter_raster,LAS-method), 8