

# Package ‘CspStandSegmentation’

October 2, 2026

**Type** Package

**Title** Comparative Shortest Path Forest Stand Segmentation from LiDAR Data

**Version** 0.2.1

**Description** Functionality for segmenting individual trees from a forest stand scanned with a close-range (e.g., terrestrial or mobile) laser scanner. The complete workflow from a raw point cloud to a complete tabular forest inventory is provided. The package contains several algorithms for detecting tree bases and a graph-based algorithm to attach all remaining points to these tree bases. It builds heavily on the 'lidR' package. A description of the segmentation algorithm can be found in Larysch et al. (2025) <[doi:10.1007/s10342-025-01796-z](https://doi.org/10.1007/s10342-025-01796-z)>.

**URL** <https://github.com/JulFrey/CspStandSegmentation>

**BugReports** <https://github.com/JulFrey/CspStandSegmentation/issues>

**License** GPL-3

**Encoding** UTF-8

**Imports** Rcpp (>= 1.0.5), lidR, dbscan, igraph, foreach, doParallel, magrittr, data.table, sf, terra, RCSF, conicfit, rgl, RANN, colorspace, methods, stats

**LinkingTo** Rcpp, RcppArmadillo, lidR, BH

**Depends** R (>= 4.1.0)

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** yes

**Author** Julian Frey [aut, cre] (ORCID: <<https://orcid.org/0000-0001-7895-702X>>),  
Zoe Schindler [ctb] (ORCID: <<https://orcid.org/0000-0003-2972-1920>>),  
Katja Kröner [ctb] (ORCID: <<https://orcid.org/0009-0001-9896-8057>>)

**Maintainer** Julian Frey <julian.frey@wwd.uni-freiburg.de>

**Repository** CRAN

**Date/Publication** 2026-10-02 16:00:21 UTC

Contents

add\_geometry . . . . . 2

add\_las\_attributes . . . . . 3

add\_voxel\_coordinates . . . . . 4

color\_ids . . . . . 5

comparative\_shortest\_path . . . . . 6

csp\_cost\_segmentation . . . . . 7

eigen\_decomposition . . . . . 9

fast\_unlist . . . . . 10

fast\_unlist\_dist . . . . . 10

fds . . . . . 11

find\_base\_coordinates\_geom . . . . . 12

find\_base\_coordinates\_raster . . . . . 13

forest\_inventory . . . . . 14

forest\_inventory\_simple . . . . . 15

get\_pal . . . . . 16

las\_merge . . . . . 17

plot\_inventory . . . . . 18

point\_center\_angle . . . . . 19

point\_circle\_distance . . . . . 19

p\_dist . . . . . 20

p\_mat\_dist . . . . . 20

ransac\_circle\_fit . . . . . 21

suppress\_cat . . . . . 22

voxelize\_points\_mean\_attributes . . . . . 22

Index 24

---

|              |                                        |
|--------------|----------------------------------------|
| add_geometry | Add geometric features to a LAS object |
|--------------|----------------------------------------|

---

Description

The function calls a fast cpp multi-core function to calculate eigenvalues for the points in a point cloud based on the k nearest neighbors. Afterwards it adds geometric features like Curvature, Linearity, Planarity, Sphericity, Anisotrophy and Verticlity to the points itself.

Usage

```
add_geometry(las, k = 10L, n_cores = 1)
```

Arguments

|         |                                                               |
|---------|---------------------------------------------------------------|
| las     | A LAS object (see lidR::LAS)                                  |
| k       | the k nearest neighbors to use for the eigenvalue calculation |
| n_cores | The number of CPU cores to use                                |

**Details**

Details of the metrics can be found in: Hackel, T., Wegner, J.D. & Schindler, K. (2016) Contour Detection in Unstructured 3D Point Clouds. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp. 1610-1618.

**Value**

The function returns a single LAS object with the geometric features attached to it in the LAS@data section.

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**Examples**

```
LASfile <- system.file("extdata", "beech.las", package="CspStandSegmentation")
las <- lidR::readLAS(LASfile, select = "xyz")

las <- add_geometry(las, k = 5, n_cores = 2)
summary(las@data)
```

---

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| add_las_attributes | <i>Add all las_attributes from las@data to the header of a las element</i> |
|--------------------|----------------------------------------------------------------------------|

---

**Description**

The helper function adds all headings from las@data which are not part of lidR::LASATTRIBUTES to the las header using lidR::add\_lasattribute. Only attributes that are included in the header got saved when using lidR::writeLAS, this is a convenient way to add them.

**Usage**

```
add_las_attributes(las)
```

**Arguments**

|     |                               |
|-----|-------------------------------|
| las | an element of lidR::LAS class |
|-----|-------------------------------|

**Value**

the las file with updated header

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**Examples**

```

file <- system.file("extdata", "beech.las", package="CspStandSegmentation")
las <- lidR::readTLAS(file)

las@data$noise <- runif(nrow(las@data))
las@data$noiseZ <- las@data$var1 * las@data$Z

las <- add_las_attributes(las)

```

---

add\_voxel\_coordinates *Add voxel coordinates to a las file*

---

**Description**

Adds the columns `x_vox`, `y_vox` and `z_vox` in the given resolution to the las element. This is convenient if information has been derived in voxel space and these should be attached to the original points.

**Usage**

```
add_voxel_coordinates(las, res)
```

**Arguments**

|                  |                                            |
|------------------|--------------------------------------------|
| <code>las</code> | an element of <code>lidR::LAS</code> class |
| <code>res</code> | voxel resolution in [m]                    |

**Details**

Voxel coordinates derived with this function are identical to those derived by `lidR::voxelize`.

**Value**

las file with additional voxel coordinates

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**Examples**

```

file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLAS(file)

las <- add_voxel_coordinates(las, res = 1)

```

---

|           |                                                                                      |
|-----------|--------------------------------------------------------------------------------------|
| color_ids | <i>Assign colours to instances in a LAS object using a graph-colouring heuristic</i> |
|-----------|--------------------------------------------------------------------------------------|

---

## Description

color\_ids computes a per-instance colour assignment and merges RGB (R,G,B) and a numeric color\_ID back into las@data. Colour assignment is driven by spatial adjacency (k-nearest neighbours on instance centroids) and maximizing colour LAB distance from neighbouring instances.

## Usage

```
color_ids(
  las,
  col = "sky",
  n_col = 10,
  n_neighbors = 10,
  instance_id = "TreeID",
  ground_id = 0,
  ground_color = "#ffffff",
  overwrite_rgb = TRUE
)
```

## Arguments

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| las           | A LAS object (from the <b>lidR</b> package).                                                                            |
| col           | Either a character string naming a palette supported by get_pal() or a character vector of hex colours. Default: "sky". |
| n_col         | Integer. Number of discrete colours to generate from the palette. Default: 10.                                          |
| n_neighbors   | Integer. Number of nearest neighbours to treat as adjacency when assigning colours. Default: 10.                        |
| instance_id   | Character. Column name in las@data that identifies instances (for example tree or stand IDs). Default: "TreeID".        |
| ground_id     | Value used to identify ground or background instance; assigned ground_color. Default: 0.                                |
| ground_color  | Character. Hex colour used for the ground instance. Default: "#ffffff".                                                 |
| overwrite_rgb | Logical. If TRUE existing R/G/B columns in las@data will be overwritten. Default: TRUE.                                 |

## Details

The function computes instance centroids, builds a k-nearest-neighbour graph, sorts instances by local neighbour density and assigns discrete colours greedily so neighbouring instances receive maximally different colours in LAB space.

**Value**

A LAS object with R, G, B (0-255 integers) and color\_ID merged into las@data. If `overwrite_rgb` is FALSE and the LAS already had RGB columns, those channels are preserved.

**Author(s)**

Zoe Schindler

**Examples**

```
## Not run:
# read example data
\donttest{
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLAS(file)

# Find tree positions as starting points for segmentation
map <- CspStandSegmentation::find_base_coordinates_raster(las)

# segment trees
segmented <- las |>
CspStandSegmentation::csp_cost_segmentation(map, 1, S_w = 0.5)
las_col <- CspStandSegmentation::color_ids(segmented,
  col = "sky",
  instance_id = "TreeID",
  ground_color = "#ff0000")
lidR::plot(las_col, color = "RGB")
}

## End(Not run)
```

---

comparative\_shortest\_path

*helper function for csp\_cost\_segemntation*

---

**Description**

The function performs a Dijkstra algorithm on a 3D voxel file to assign every voxel to the closest seed point using the igraph package.

**Usage**

```
comparative_shortest_path(
  vox = vox,
  adjacency_df = adjacency_df,
  seeds,
  v_w = 0,
```

```

    l_w = 0,
    s_w = 0,
    N_cores = parallel::detectCores() - 1,
    Voxel_size,
    N_trees = 1
  )

```

### Arguments

|               |                                                                                                                                                                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| vox           | a LAS S4 element with XYZ voxel coordinates in the @data slot.                                                                                                                                                                                       |
| adjacency_df  | a data.frame with voxel ids (row numbers) in the first column and a neighboring voxel ID in the second column and the weight (distance) in the third column. Might be generated using the dbscan::frNN function (which requires reshaping the data). |
| seeds         | seed points for tree positions.                                                                                                                                                                                                                      |
| v_w, l_w, s_w | weights for verticality, linearity spericity see <a href="#">csp_cost_segmentation</a>                                                                                                                                                               |
| N_cores       | Number of CPU cores for multi-threading                                                                                                                                                                                                              |
| Voxel_size    | Edge length used to create the voxels. This is only important to gain comparable distance weights on different voxel sizes. Should be greater than 0.                                                                                                |
| N_trees       | The number of closest stem locations to add to the point cloud If > 1 the distances will be added as well.                                                                                                                                           |

### Value

voxels with the TreeID in the data slot

### Author(s)

Julian Frey <julian.frey@wwd.uni-freiburg.de>

### See Also

[csp\\_cost\\_segmentation](#)

---

csp\_cost\_segmentation *Comparative Shortest Path with cost weighting tree segmentation*

---

### Description

Segments single trees from forest point clouds based on tree positions (xyz-coordinates) provided in the map argument.

**Usage**

```
csp_cost_segmentation(
  las,
  map,
  Voxel_size = 0.3,
  V_w = 0,
  L_w = 0,
  S_w = 0,
  N_cores = 1,
  N_trees = 1
)
```

**Arguments**

|            |                                                                                                                                                                                                                           |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| las        | A lidR LAS S4 object.                                                                                                                                                                                                     |
| map        | A data.frame, including the columns X, Y, Z, TreeID, with X and Y depicting the location of the trees. Can be generated using CspStandSegmentation::find_base_coordinates_raster                                          |
| Voxel_size | The voxel size (3D resolution) for the routing graph to determine the nearest map location for every point in the point cloud.                                                                                            |
| V_w        | verticality weight. Since trunks are vertical structures, routing through voxels with high verticality can be rated 'cheaper.' should be a number between 0 and 1 with 0 meaning no benefit for more vertical structures. |
| L_w        | Linearity weight. Similar to V_w but for linearity, higher values indicate a malus for linear shapes (usually branches).                                                                                                  |
| S_w        | Sphericity weight. Similar to V_w but for sphericity, higher values indicate a malus for spherical shapes (usually small branches and leaves).                                                                            |
| N_cores    | number of CPU cores used for parallel routing using the foreach package.                                                                                                                                                  |
| N_trees    | The number of closest stem locations to add to the point cloud                                                                                                                                                            |

**Details**

The whole point cloud is voxelized in the given resolution and the center of gravity for the points inside is calculated as voxel coordinate. A graph is build, which connects the voxel coordinates based on a db-scan algorithm. The distances between the voxel coordinates is weighted based on geometric features computed for the points in the voxels. Distances along planar and/or vertical faces like stems are weighted shorter than distances through voxels with high sphericity like leaves and clusters of twigs. This avoids small individuals spreading into the upper canopy. For every voxel center, the weighted distance in the network is calculated to all tree locations from the map argument. The TreeID of the map argument with the shortest distance is assigned to the voxel. All points in the point cloud receive the TreeID from their parent voxel.

**Value**

Returns a copy of the las point cloud with an additional field for the TreeID.

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**See Also**[comparative\\_shortest\\_path](#)**Examples**

```
# read example data

file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLSLAS(file)

# Find tree positions as starting points for segmentation
map <- CspStandSegmentation::find_base_coordinates_raster(las)

# segment trees
segmented <- las |>
CspStandSegmentation::add_geometry() |>
CspStandSegmentation::csp_cost_segmentation(map, 1, S_w = 0.5)
```

---

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| eigen_decomposition | <i>Fast Eigenvalues decomposition for k nearest neighbors using a C++ function</i> |
|---------------------|------------------------------------------------------------------------------------|

---

**Description**

C++ helper function to compute eigenvalues for geometric feature calculation.

**Usage**

```
eigen_decomposition(las, k, ncpu = 1L)
```

**Arguments**

|      |                            |
|------|----------------------------|
| las  | LAS element                |
| k    | k nearest neighbors        |
| ncpu | number of cpu cores to use |

**Value**

The function returns for every point the 3 eigenvalues and the third element of the third eigenvector. These values are needed to compute planarity, linerity, verticality etc. in the add\_geometry function

**Author(s)**

Julian Frey <julian.frey@iww.uni-freiburg.de>

**See Also**[add\\_geometry](#)

---

|             |                                                                |
|-------------|----------------------------------------------------------------|
| fast_unlist | <i>helper function to unlist IDs generated by dbscan::frNN</i> |
|-------------|----------------------------------------------------------------|

---

**Description**

creates a vector of indices from a nested list created by dbscan::frNN

**Usage**

```
fast_unlist(list, l)
```

**Arguments**

|      |                                        |
|------|----------------------------------------|
| list | a list element created by dbscan::frNN |
| l    | the expected length of the result      |

**Value**

Returns a vector with the values in the list.

**Author(s)**

Julian Frey <julian.frey@iww.uni-freiburg.de>

---

|                  |                                                                     |
|------------------|---------------------------------------------------------------------|
| fast_unlist_dist | <i>helper function to unlist distances computed by dbscan::frNN</i> |
|------------------|---------------------------------------------------------------------|

---

**Description**

extracts the distances from a nested list created by dbscan::frNN

**Usage**

```
fast_unlist_dist(list, l)
```

**Arguments**

|      |                                        |
|------|----------------------------------------|
| list | a list element created by dbscan::frNN |
| l    | the expected length of the result      |

**Value**

Returns a vector with the values in the list.

**Author(s)**

Julian Frey <julian.frey@iww.uni-freiburg.de>

---

 fds

*Farthest Distance Sampling (Farthest Point Sampling)*


---

**Description**

This function selects *n* points from a matrix of points such that the minimum distance between any two points is maximized. Either a fixed number of points can be selected (using the ‘*n*’ parameter) or points can be selected until a specified minimum spacing is violated (using the ‘*spacing*’ parameter). This version is memory efficient and can handle large matrices.

**Usage**

```
fds(mat, n = NULL, spacing = NULL, ret = "idx", scale = FALSE)
```

**Arguments**

|                |                                                                                                                                              |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>mat</i>     | a matrix of points with one row for each point and one column for each dimension, can also be a las object then only XYZ will be used        |
| <i>n</i>       | the number of points to select, or if <1 the proportion of points to select                                                                  |
| <i>spacing</i> | If supplied, sampling continues until the minimum nearest- neighbour distance among selected points drops below ‘ <i>spacing</i> ’.          |
| <i>ret</i>     | the type of output to return. Options are "idx" (default) to return the indices of the selected points, "mat" to return the selected points. |
| <i>scale</i>   | logical. If TRUE, the dimensions are scaled to have a mean of 0 and a standard deviation of 1 before calculating distances.                  |

**Value**

a vector of indices or a matrix of points

**Examples**

```
mat <- matrix(rnorm(1000), ncol = 10)
sample <- fds(mat, n = 50, ret = "mat")
str(sample)
```

---

`find_base_coordinates_geom`*Find stem base position using a geometric feature filtering and clustering approach*

---

## Description

Find stem base position using a geometric feature filtering and clustering approach

## Usage

```
find_base_coordinates_geom(  
  las,  
  zmin = 0.5,  
  zmax = 2,  
  res = 0.5,  
  min_verticality = 0.9,  
  min_planarity = 0.5,  
  min_cluster_size = NULL  
)
```

## Arguments

|                               |                                                                                                        |
|-------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>las</code>              | an element of <code>lidR::LAS</code> class                                                             |
| <code>zmin</code>             | lower search boundary                                                                                  |
| <code>zmax</code>             | upper search boundary                                                                                  |
| <code>res</code>              | cluster search radius                                                                                  |
| <code>min_verticality</code>  | minimum verticality >0 & <1 for a point to be considered a stem point                                  |
| <code>min_planarity</code>    | minimum planarity >0 & <1 for a point to be considered a stem point                                    |
| <code>min_cluster_size</code> | minimum number of points in the cluster to be considered a tree, if NULL median cluster size is chosen |

## Value

data.frame with X, Y, Z and TreeID for stem base positions

## Author(s)

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**Examples**

```
# read example data
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
tls = lidR::readTLSLAS(file)

# Find tree positions
map <- CspStandSegmentation::find_base_coordinates_geom(tls)
```

---

`find_base_coordinates_raster`*Find stem base position using a density raster approach*

---

**Description**

Find stem base position using a density raster approach

**Usage**

```
find_base_coordinates_raster(
  las,
  res = 0.1,
  zmin = 0.5,
  zmax = 2,
  q = 0.975,
  eps = 0.2
)
```

**Arguments**

|                   |                                                    |
|-------------------|----------------------------------------------------|
| <code>las</code>  | an element of <code>lidR::LAS</code> class         |
| <code>res</code>  | raster resolution                                  |
| <code>zmin</code> | lower search boundary                              |
| <code>zmax</code> | upper search boundary                              |
| <code>q</code>    | quantile of raster density to assign a tree region |
| <code>eps</code>  | search radius to merge base points                 |

**Value**

data.frame with X, Y, Z and TreeID for stem base positions

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

## Examples

```
# read example data
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
tls = lidR::readTLSLAS(file)

# Find tree positions
map <- CspStandSegmentation::find_base_coordinates_raster(tls)
```

---

|                  |                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------|
| forest_inventory | <i>Function to perform a forest inventory based on a segmented las object (needs to contain TreeID)</i> |
|------------------|---------------------------------------------------------------------------------------------------------|

---

## Description

This function estimates a taper curve for every tree and returns the DBH at 1.3m, its position in XY coordinates, the tree height and the trees 2D projection area.

## Usage

```
forest_inventory(
  las,
  slice_min = 0.3,
  slice_max = 4,
  increment = 0.2,
  width = 0.1,
  max_dbh = 1,
  n_cores = 1,
  tree_id_col = "TreeID",
  non_tree_id = 0,
  use_stem_segmentation = FALSE,
  semantic_colname = NULL,
  stem_semantic_label = NULL
)
```

## Arguments

|             |                                                                     |
|-------------|---------------------------------------------------------------------|
| las         | lidR las object with the segmented trees                            |
| slice_min   | the minimum height of a slice for stems to estimate the taper curve |
| slice_max   | the maximum height of a slice for stems to estimate the taper curve |
| increment   | the increment of the slices                                         |
| width       | the width of the slices                                             |
| max_dbh     | the maximum DBH allowed                                             |
| n_cores     | number of cores to use                                              |
| tree_id_col | Column name for the instance segmentation ID (TreeIDs)              |
| non_tree_id | tree_id_col value for non tree elements (can be a vector of IDs)    |

```

use_stem_segmentation
    logical whether to use only points classified as stem for DBH estimation
semantic_colname
    character name of the semantic segmentation column (only needed if use_stem_segmentation
    is TRUE)
stem_semantic_label
    integer semantic label value for stem points (only needed if use_stem_segmentation
    is TRUE)

```

**Value**

a data.frame with the TreeID, X, Y, DBH, quality\_flag, Height and ConvexHullArea

**Examples**

```

# read example data
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLSLAS(file)

# find tree positions as starting points for segmentation
map <- CspStandSegmentation::find_base_coordinates_raster(las)

# segment trees
segmented <- las |>
  CspStandSegmentation::add_geometry(n_cores = 2) |>
  CspStandSegmentation::csp_cost_segmentation(map, 1, N_cores = 2)

# perform inventory
inventory <- CspStandSegmentation::forest_inventory(segmented, n_cores = 2)

```

---

forest\_inventory\_simple

*Function to perform a forest inventory based on a segmented las object (needs to contain TreeID) This version is a faster but more simplistic approach than forest\_inventory() for the DBH estimates*

---

**Description**

Function to perform a forest inventory based on a segmented las object (needs to contain TreeID)  
This version is a faster but more simplistic approach than forest\_inventory() for the DBH estimates

**Usage**

```

forest_inventory_simple(
  las,
  slice_min = 1.2,
  slice_max = 1.4,

```

```

    max_dbh = 1,
    n_cores = max(c(1, parallel::detectCores()/2 - 1))
  )

```

### Arguments

|           |                                          |
|-----------|------------------------------------------|
| las       | lidR las object with the segmented trees |
| slice_min | the minimum height of a DBH slice        |
| slice_max | the maximum height of a DBH slice        |
| max_dbh   | the maximum DBH allowed                  |
| n_cores   | number of cores to use                   |

### Value

a data.frame with the TreeID, X, Y, DBH, quality\_flag, Height and ConvexHullArea

---

|         |                                             |
|---------|---------------------------------------------|
| get_pal | <i>Return a palette-generating function</i> |
|---------|---------------------------------------------|

---

### Description

get\_pal returns a function that produces a character vector of hex colours given a required length. It exposes a set of named palettes used by color\_ids().

### Usage

```
get_pal(name = "rainbow")
```

### Arguments

|      |                                                                                                                                                                     |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name | Character scalar. Name of the palette to use. Supported names: "sky", "sea", "cozy", "fairy", "winter", "rainbow", "pastel", "candy", "boring". Default: "rainbow". |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Value

A function with signature function(n) that returns n hex colour strings.

### Author(s)

Zoe Schindler

### Examples

```
get_pal("sky")
```

---

|           |                                                       |
|-----------|-------------------------------------------------------|
| las_merge | <i>Makes one las-object from multiple las-objects</i> |
|-----------|-------------------------------------------------------|

---

## Description

This function merges multiple las objects into one las object. The function checks if all inputs are las-objects and if they have the same CRS. The function will also add a column oci with the original cloud index to each las-object. The function will then rbind all data by the minimum set of columns. If the fill argument is set to False, columns which do not exist in all las objects will be removed. If the fill argument is set to True, missing columns will be filled with NA.

## Usage

```
las_merge(..., oci = FALSE, fill = FALSE)
```

## Arguments

|      |                                                                                                                  |
|------|------------------------------------------------------------------------------------------------------------------|
| ...  | any number of las objects or a list with las objects                                                             |
| oci  | add a column with the original cloud index                                                                       |
| fill | fill missing columns with NA if it is set to False columns which do not exist in all las-objects will be removed |

## Value

A single las-object with only the overlapping column name

## Examples

```
# number of points per las
n <- 100
las1 <- lidR::LAS(data.frame(X = runif(n), Y = runif(n), Z = runif(n)))
las2 <- lidR::LAS(data.frame(X = runif(n) + 2, Y = runif(n), Z = runif(n)))
las3 <- lidR::LAS(data.frame(X = runif(n) + 4, Y = runif(n), Z = runif(n)))
merged <- las_merge(las1, las2, las3)
lidR::npoints(merged) == (n*3)

lasList <- list(las1, las2, las3)
merged <- las_merge(lasList)
lidR::npoints(merged) == (n*3)
```

---

|                |                                                                                      |
|----------------|--------------------------------------------------------------------------------------|
| plot_inventory | <i>Function to plot the inventory results into a lidR 3d plot of the point cloud</i> |
|----------------|--------------------------------------------------------------------------------------|

---

### Description

Function to plot the inventory results into a lidR 3d plot of the point cloud

### Usage

```
plot_inventory(plot, inventory, col = NA, cex = 1.5, label_col = "white")
```

### Arguments

|           |                                       |
|-----------|---------------------------------------|
| plot      | lidR 3d plot                          |
| inventory | data.frame with the inventory results |
| col       | color of the spheres                  |
| cex       | numeric size of the labels            |
| label_col | character color of the labels         |

### Value

the plot with the inventory results

### Examples

```
# read example data
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLSLAS(file)

# find tree positions as starting points for segmentation
map <- CspStandSegmentation::find_base_coordinates_raster(las)

# segment trees
segmented <- las |>
  CspStandSegmentation::add_geometry(n_cores = 2) |>
  CspStandSegmentation::csp_cost_segmentation(map, 1, N_cores = 2)

# perform inventory
inventory <- CspStandSegmentation::forest_inventory(segmented, n_cores = 2)

# plot the results
## Not run:
x <- lidR::plot(segmented, color = "TreeID")
plot_inventory(x, inventory)

## End(Not run)
```

---

|                    |                                                                                  |
|--------------------|----------------------------------------------------------------------------------|
| point_center_angle | <i>Returns the angle between the center of the circle and a point in degrees</i> |
|--------------------|----------------------------------------------------------------------------------|

---

**Description**

Returns the angle between the center of the circle and a point in degrees

**Usage**

```
point_center_angle(point, circle)
```

**Arguments**

|        |                                                          |
|--------|----------------------------------------------------------|
| point  | numeric vector of length 2 c(X,Y)                        |
| circle | numeric vector of length 3 c(center_X, center_Y, radius) |

**Value**

numeric angle in degrees

---

|                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| point_circle_distance | <i>Helper function to compute distances from a point to the circle</i> |
|-----------------------|------------------------------------------------------------------------|

---

**Description**

Helper function to compute distances from a point to the circle

**Usage**

```
point_circle_distance(point, circle)
```

**Arguments**

|        |                                                          |
|--------|----------------------------------------------------------|
| point  | numeric vector of length 2 c(X,Y)                        |
| circle | numeric vector of length 3 c(center_X, center_Y, radius) |

**Value**

numeric distance from the point to the circle

---

|        |                                |
|--------|--------------------------------|
| p_dist | <i>Point distance function</i> |
|--------|--------------------------------|

---

**Description**

calculates euclidean distances for n dimensions

**Usage**

```
p_dist(p1, p2)
```

**Arguments**

|    |         |
|----|---------|
| p1 | point 1 |
| p2 | point 2 |

**Value**

the distance between the two points

**Examples**

```
p_dist(c(0,0), c(3,4))
```

---

|            |                                |
|------------|--------------------------------|
| p_mat_dist | <i>Point distance function</i> |
|------------|--------------------------------|

---

**Description**

calculates euclidean distances for n dimensions between a matrix of points and a single point

**Usage**

```
p_mat_dist(mat, p, nthreads = 0L)
```

**Arguments**

|          |                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------|
| mat      | matrix with points as rows                                                                        |
| p        | point to calculate distances                                                                      |
| nthreads | number of threads to use. If 0 or negative, the maximum number of threads available will be used. |

**Value**

the distances between every row of mat and p

**Examples**

```
p_mat_dist(as.matrix(cbind(runif(100),runif(100))), c(3,4))
```

---

|                   |                                                                                  |
|-------------------|----------------------------------------------------------------------------------|
| ransac_circle_fit | <i>RANSAC circle fitting algorithm specially adapted for tree DBH estimation</i> |
|-------------------|----------------------------------------------------------------------------------|

---

**Description**

This function fits a circle to a set of points using the RANSAC algorithm it maximizes the points that are in the circle and the number of filled 36 degree angle segments Therefore, this function searches for the most complete circle with the highest number of points represented.

**Usage**

```
ransac_circle_fit(
  data,
  n_iterations = 1000L,
  distance_threshold = 0.01,
  min_inliers = 3L
)
```

**Arguments**

|                    |                                                                                |
|--------------------|--------------------------------------------------------------------------------|
| data               | numeric matrix with 2 columns (X, Y) representing the point cloud              |
| n_iterations       | integer maximum number of iterations                                           |
| distance_threshold | numeric maximum distance from a point to the circle to be considered an inlier |
| min_inliers        | integer minimum number of inliers to consider the circle as valid              |

**Value**

a list with the following elements: circle: the center coordinates and radius of the circle inliers: number of points within the circles dist threshold angle\_segs: number of populated 10deg angular segments of the circle using the distance\_threshold n\_iter: number of iterations run

---

|              |                                       |
|--------------|---------------------------------------|
| suppress_cat | <i>Suppress only the cat() output</i> |
|--------------|---------------------------------------|

---

**Description**

Suppress only the cat() output

**Usage**

```
suppress_cat(f, ...)
```

**Arguments**

|     |                            |
|-----|----------------------------|
| f   | function to be called      |
| ... | parameters to the function |

**Value**

the return value of the function

---

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| voxelize_points_mean_attributes | <i>helper function to voxelize a las element</i> |
|---------------------------------|--------------------------------------------------|

---

**Description**

Calculate voxel mean values for all numeric attributes in the las@data table including the XYZ-coordinates.

**Usage**

```
voxelize_points_mean_attributes(las, res)
```

**Arguments**

|     |                           |
|-----|---------------------------|
| las | a lidR::LAS element       |
| res | voxel resolution in meter |

**Value**

a las element with XYZ-coordinates as the voxel center and X\_gr,Y\_gr,Z\_gr as the center of gravity (mean point coordinates) as well as all other numeric columns voxel mean values with their original name.

**Author(s)**

Julian Frey <julian.frey@wwd.uni-freiburg.de>

**See Also**

[voxelize\\_points](#)

**Examples**

```
# read example data
file = system.file("extdata", "beech.las", package="CspStandSegmentation")
las = lidR::readTLSLAS(file)
vox <- las |> voxelize_points_mean_attributes(1)
```

# Index

[add\\_geometry](#), [2](#), [10](#)  
[add\\_las\\_attributes](#), [3](#)  
[add\\_voxel\\_coordinates](#), [4](#)  
  
[color\\_ids](#), [5](#)  
[comparative\\_shortest\\_path](#), [6](#), [9](#)  
[csp\\_cost\\_segmentation](#), [7](#), [7](#)  
  
[eigen\\_decomposition](#), [9](#)  
  
[fast\\_unlist](#), [10](#)  
[fast\\_unlist\\_dist](#), [10](#)  
[fds](#), [11](#)  
[find\\_base\\_coordinates\\_geom](#), [12](#)  
[find\\_base\\_coordinates\\_raster](#), [13](#)  
[forest\\_inventory](#), [14](#)  
[forest\\_inventory\\_simple](#), [15](#)  
  
[get\\_pal](#), [16](#)  
  
[las\\_merge](#), [17](#)  
  
[p\\_dist](#), [20](#)  
[p\\_mat\\_dist](#), [20](#)  
[plot\\_inventory](#), [18](#)  
[point\\_center\\_angle](#), [19](#)  
[point\\_circle\\_distance](#), [19](#)  
  
[ransac\\_circle\\_fit](#), [21](#)  
  
[suppress\\_cat](#), [22](#)  
  
[voxelize\\_points](#), [23](#)  
[voxelize\\_points\\_mean\\_attributes](#), [22](#)