Package 'TreeLS'

August 25, 2020

Type Package

Title Terrestrial Point Cloud Processing of Forest Data

Version 2.0.2

Date 2020-08-21

Description High performance algorithms for manipulation of terrestrial 'Li-
DAR' (but not only) point clouds for use in research and forest monitoring applications, be-
ing fully compatible with the 'LAS' infrastructure of 'lidR'. For in depth descriptions of stem de-
noising and segmentation algorithms re-
fer to Conto et al. (2017) <doi:10.1016/j.compag.2017.10.019>.

URL https://github.com/tiagodc/TreeLS

Depends R (>= 3.3.0), data.table (>= 1.12.0), magrittr (>= 1.5), lidR (>= 3.0.0)

Imports rgl, raster, sp, deldir, dismo, nabor, benchmarkme, rlas, glue, mathjaxr

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo RcppArmadillo,Rcpp,BH,RcppEigen

RoxygenNote 7.1.1

RdMacros mathjaxr

R topics documented:

`circleFit` *Point cloud circle fit*

Description

Fits a 2D horizontally-aligned circle on a set of 3D points.

Usage

```
circleFit(las, method = "irls", n = 5, inliers = 0.8, conf = 0.99, n_best = 0)
```

Arguments

las	LAS object.
method	method for estimating the circle parameters. Currently available: "qr", "nm", "irls" and "ransac".
n	numeric - number of points selected on every RANSAC iteration.
inliers	numeric - expected proportion of inliers among stem segments' point cloud chunks.
conf	numeric - confidence level.
n_best	integer - estimate optimal RANSAC parameters as the median of the n_best estimations with lowest error.

Value

vector of parameters

---

cylinderFit *Point cloud cylinder fit*

Description

Fits a cylinder on a set of 3D points.

Usage

```
cylinderFit(
  las,
  method = "ransac",
  n = 5,
  inliers = 0.9,
  conf = 0.95,
  max_angle = 30,
  n_best = 20
)
```

Arguments

las	LAS object.
method	method for estimating the cylinder parameters. Currently available: "nm", "irls", "ransac" and "bf".
n	numeric - number of points selected on every RANSAC iteration.
inliers	numeric - expected proportion of inliers among stem segments' point cloud chunks.
conf	numeric - confidence level.

max_angle
: numeric - used when method == "bf". The maximum tolerated deviation, in degrees, from an absolute vertical line (Z = c(0,0,1)).

n_best
: integer - estimate optimal RANSAC parameters as the median of the n_best estimations with lowest error.

Value

vector of parameters

---

fastPointMetrics *Calculate point neighborhood metrics*

Description

Get statistics for every point in a LAS object. Neighborhood search methods are prefixed by ptm.

Usage

```
fastPointMetrics(
  las,
  method = ptm.voxel(),
  which_metrics = ENABLED_POINT_METRICS$names
)
```

Arguments

las
: LAS object.

method
: neighborhood search algorithm. Currently available: ptm.voxel and ptm.knn.

which_metrics
: optional character vector - list of metrics (by name) to be calculated. Check out fastPointMetrics.available for a list of all metrics.

Details

Individual or voxel-wise point metrics build up the basis for many studies involving TLS in forestry. This function is used internally in other *TreeLS* methods for tree mapping and stem denoising, but also may be useful to users interested in developing their own custom methods for point cloud classification/filtering of vegetation features or build up input datasets for machine learning classifiers.

fastPointMetrics provides a way to calculate several geometry related metrics (listed below) in an optimized way. All metrics are calculated internally by C++ functions in a single pass (*O(n)* time), hence *fast*. This function is provided for convenience, as it allows very fast calculations of several complex variables on a single line of code, speeding up heavy work loads. For a more flexible approach that allows user defined metrics check out point_metrics from the *lidR* package.

In order to avoid excessive memory use, not all available metrics are calculated by default. The calculated metrics can be specified every time fastPointMetrics is run by naming the desired metrics into the which_metrics argument, or changed globally for the active R session by setting new default metrics using fastPointMetrics.available.

Value

LAS object.

List of available point metrics

* *EVi* = *i*-th 3D eigen value

* *EV2Di* = *i*-th 2D eigen value

- `N`: number of nearest neighbors
- `MinDist`: minimum distance among neighbors
- `MaxDist`: maximum distance among neighbors
- `MeanDist`: mean distance
- `SdDist`: standard deviation of within neighborhood distances
- `Linearity`: linear saliency, $(EV_1 + EV_2)/EV_1$
- `Planarity`: planar saliency, $(EV_2 + EV_3)/EV_1$
- `Scattering`: EV_3/EV_1
- `Omnivariance`: $(EV_2 + EV_3)/EV_1$
- `Anisotropy`: $(EV_1 - EV_3)/EV_1$
- `Eigentropy`: $-\sum_{i=1}^{n=3} EV_i * ln(EV_i)$
- `EigenSum`: sum of eigenvalues, $\sum_{i=1}^{n=3} EV_i$
- `Curvature`: surface variation, $EV_3/EigenSum$
- `KnnRadius`: 3D neighborhood radius
- `KnnDensity`: 3D point density (N / sphere volume)
- `Verticality`: absolute vertical deviation, in degrees
- `ZRange`: point neighborhood height difference
- `ZSd`: standard deviation of point neighborhood heights
- `KnnRadius2d`: 2D neighborhood radius
- `KnnDensity2d`: 2D point density (N / circle area)
- `EigenSum2d`: sum of 2D eigenvalues, $\sum_{i=1}^{n=2} EV2D_i$
- `EigenRatio2d`: $EV2D_2/EV2D_1$
- `EigenValuei`: 3D eigenvalues
- `EigenVectorij`: 3D eigenvector coefficients, *i*-th load of *j*-th eigenvector

References

Wang, D.; Hollaus, M.; Pfeifer, N., 2017. Feasibility of machine learning methods for separating wood and leaf points from terrestrial laser scanning data. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume IV-2/W4.

Zhou, J. et. al., 2019. Separating leaf and wood points in terrestrial scanning data using multiple optimal scales. Sensors, 19(8):1852.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file, select='xyz')

all_metrics = fastPointMetrics.available()
my_metrics = all_metrics[c(1,4,6)]

tls = fastPointMetrics(tls, ptm.knn(10), my_metrics)
head(tls@data)
plot(tls, color='Linearity')
```

---

fastPointMetrics.available

Print available point metrics

Description

Print the list of available metrics for `fastPointMetrics`.

Usage

```
fastPointMetrics.available(enable = ENABLED_POINT_METRICS$names)
```

Arguments

`enable`	optional `integer` or `character` vector containing indices or names of the metrics you want to enable globally. Enabled metrics are calculated every time you run `fastPointMetrics` by default. Only metrics used internally in other *TreeLS* methods are enabled out-of-the-box.

Value

`character` vector of all metrics.

List of available point metrics

* *EVi* = *i*-th 3D eigen value

* *EV2Di* = *i*-th 2D eigen value

- `N`: number of nearest neighbors
- `MinDist`: minimum distance among neighbors
- `MaxDist`: maximum distance among neighbors
- `MeanDist`: mean distance
- `SdDist`: standard deviation of within neighborhood distances
- `Linearity`: linear saliency, $(EV_1 + EV_2)/EV_1$
- `Planarity`: planar saliency, $(EV_2 + EV_3)/EV_1$

- `Scattering`: EV_3/EV_1
- `Omnivariance`: $(EV_2 + EV_3)/EV_1$
- `Anisotropy`: $(EV_1 - EV_3)/EV_1$
- `Eigentropy`: $-\sum_{i=1}^{n=3} EV_i * ln(EV_i)$
- `EigenSum`: sum of eigenvalues, $\sum_{i=1}^{n=3} EV_i$
- `Curvature`: surface variation, $EV_3/EigenSum$
- `KnnRadius`: 3D neighborhood radius
- `KnnDensity`: 3D point density (N / sphere volume)
- `Verticality`: absolute vertical deviation, in degrees
- `ZRange`: point neighborhood height difference
- `ZSd`: standard deviation of point neighborhood heights
- `KnnRadius2d`: 2D neighborhood radius
- `KnnDensity2d`: 2D point density (N / circle area)
- `EigenSum2d`: sum of 2D eigenvalues, $\sum_{i=1}^{n=2} EV2D_i$
- `EigenRatio2d`: $EV2D_2/EV2D_1$
- `EigenValuei`: 3D eigenvalues
- `EigenVectorij`: 3D eigenvector coefficients, *i*-th load of *j*-th eigenvector

Examples

```
m = fastPointMetrics.available()
length(m)
```

`gpsTimeFilter` *Filter points based on the* `gpstime` *field*

Description

This is a simple wrapper to `filter_poi` that takes as input relative values instead of absolute time stamps for filtering `LAS` object based on the gpstime. This function is particularly useful to check narrow intervals of point cloud frames from mobile scanning data.

Usage

```
gpsTimeFilter(las, from = 0, to = 1)
```

Arguments

`las`	`LAS` object.
`from, to`	`numeric` - gpstime percentile thresholds (from 0 to 1) to keep points in between.

Value

`LAS` object.

map.eigen.knn *Tree mapping algorithm: KNN point geometry*

Description

This function is meant to be used inside treeMap. It applies a KNN filter to select points with specific neighborhood features. For more details on geometry features, check out fastPointMetrics.

Usage

```
map.eigen.knn(
  max_curvature = 0.1,
  max_verticality = 10,
  max_mean_dist = 0.1,
  max_d = 0.5,
  min_h = 1.5,
  max_h = 3
)
```

Arguments

max_curvature	numeric - maximum curvature (from 0 to 1) accepted when filtering a point neighborhood.
max_verticality	numeric - maximum deviation of a point neighborhood's orientation from an absolute vertical axis (Z = c(0,0,1)), in *degrees* (from 0 to 90).
max_mean_dist	numeric - maximum mean distance tolerated from a point to its nearest neighbors.
max_d	numeric - largest tree diameter expected in the point cloud.
min_h, max_h	numeric - height thresholds applied to filter a point cloud before processing.

Details

Point metrics are calculated for every point. Points are then removed depending on their point metrics parameters and clustered to represent individual tree regions. Clusters are defined as a function of the expected maximum diameter. Any fields added to the point cloud are described in fastPointMetrics.

Eigen Decomposition of Point Neighborhoods

Point filtering/classification methods that rely on eigen decomposition rely on shape indices calculated for point neighborhoods (knn or voxel). To derive these shape indices, eigen decomposition is performed on the XYZ columns of a point cloud patch. Metrics related to object curvature are calculated upon ratios of the resulting eigen values, and metrics related to object orientation are caltulated from approximate normals obtained from the eigen vectors.

For instance, a point neighborhood that belongs to a perfect flat surface will have all of its variance explained by the first two eigen values, and none explained by the third eigen value. The 'normal' of such surface, i.e. the vector oriented in the direction orthogonal to the surface, is therefore represented by the third eigenvector.

Methods for both tree mapping and stem segmentation use those metrics, so in order to speed up the workflow one might apply `fastPointMetrics` to the point cloud before other methods. The advantages of this approach are that users can parameterize the point neighborhoods themselves when calculating their metrics. Those calculations won't be performed again internally in the tree mapping or stem denoising methods, reducing the overall processing time.

`map.eigen.voxel` *Tree mapping algorithm: Voxel geometry*

Description

This function is meant to be used inside `treeMap`. It applies a filter to select points belonging to voxels with specific features. For more details on geometry features, check out `fastPointMetrics`.

Usage

```
map.eigen.voxel(
  max_curvature = 0.15,
  max_verticality = 15,
  voxel_spacing = 0.1,
  max_d = 0.5,
  min_h = 1.5,
  max_h = 3
)
```

Arguments

`max_curvature`	`numeric` - maximum curvature (from 0 to 1) accepted when filtering a point neighborhood.
`max_verticality`	`numeric` - maximum deviation of a point neighborhood's orientation from an absolute vertical axis (Z = c(0,0,1)), in *degrees* (from 0 to 90).
`voxel_spacing`	`numeric` - voxel side length, in point cloud units.
`max_d`	`numeric` - largest tree diameter expected in the point cloud.
`min_h, max_h`	`numeric` - height thresholds applied to filter a point cloud before processing.

Details

Point metrics are calculated for every voxel. Points are then removed depending on their voxel's metrics metrics parameters and clustered to represent individual tree regions. Clusters are defined as a function of the expected maximum diameter. Any fields added to the point cloud are described in `fastPointMetrics`.

Eigen Decomposition of Point Neighborhoods

Point filtering/classification methods that rely on eigen decomposition rely on shape indices calculated for point neighborhoods (knn or voxel). To derive these shape indices, eigen decomposition is performed on the XYZ columns of a point cloud patch. Metrics related to object curvature are calculated upon ratios of the resulting eigen values, and metrics related to object orientation are caltulated from approximate normals obtained from the eigen vectors.

For instance, a point neighborhood that belongs to a perfect flat surface will have all of its variance explained by the first two eigen values, and none explained by the third eigen value. The 'normal' of such surface, i.e. the vector oriented in the direction orthogonal to the surface, is therefore represented by the third eigenvector.

Methods for both tree mapping and stem segmentation use those metrics, so in order to speed up the workflow one might apply `fastPointMetrics` to the point cloud before other methods. The advantages of this approach are that users can parameterize the point neighborhoods themselves when calculating their metrics. Those calculations won't be performed again internally in the tree mapping or stem denoising methods, reducing the overall processing time.

`map.hough` *Tree mapping algorithm: Hough Transform*

Description

This function is meant to be used inside `treeMap`. It applies an adapted version of the Hough Transform for circle search. Mode details are given in the sections below.

Usage

```
map.hough(
  min_h = 1,
  max_h = 3,
  h_step = 0.5,
  pixel_size = 0.025,
  max_d = 0.5,
  min_density = 0.1,
  min_votes = 3
)
```

Arguments

`min_h, max_h`	`numeric` - height thresholds applied to filter a point cloud before processing.
`h_step`	`numeric` - height interval to perform point filtering/assignment/classification.
`pixel_size`	`numeric` - pixel side length to discretize the point cloud layers while performing the Hough Transform circle search.
`max_d`	`numeric` - largest tree diameter expected in the point cloud.
`min_density`	`numeric` - between 0 and 1 - minimum point density within a pixel evaluated on the Hough Transform - i.e. only *dense* point clousters will undergo circle search.
`min_votes`	`integer` - Hough Transform parameter - minimum number of circle intersections over a pixel to assign it as a circle center candidate.

LAS@data Special Fields

Each point in the LAS object output represents a pixel center that is *possibly* also a stem cross-section center.

The variables describing each point in the output are:

- Intensity: number of votes received by that point
- PointSourceID: unique stem segment ID (among all trees)
- Keypoint_flag: if TRUE, the point is the most likely circle center of its stem segment (PointSourceID)
- Radii: approximate radius estimated by that point - always a multiple of the pixel_size
- TreeID: unique tree ID of the point
- TreePosition: if TRUE, the point represents the tree's position coordinate

Adapted Hough Transform

The Hough Transform circle search algorithm used in TreeLS applies a constrained circle search on discretized point cloud layers. Tree-wise, the circle search is recursive, in which the search for circle parameters of a stem section is constrained to the *feature space* of the stem section underneath it. Initial estimates of the stem's *feature space* are performed on a *baselise* stem segment - i.e. a low height interval where a tree's bole is expected to be clearly visible in the point cloud. The algorithm is described in detail by Conto et al. (2017).

This adapted version of the algorithm is very robust against outliers, but not against forked or leaning stems.

Tree Selection

An initial tree filter is used to select *probable* trees in the input point cloud. Parallel stacked layers, each one as thick as hstep, undergo the circle search within the hmin/hmax limits. On every layer, pixels above the min_votes criterion are clustered, forming *probability zones*. *Probability zones* vertically aligned on at least 3/4 of the stacked layers are assigned as *tree occurrence regions* and exported in the output map.

References

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

x = plot(tls)

map = treeMap(tls, map.hough(h_step = 1, max_h = 4))
add_treeMap(x, map, color='red')
```

```
xymap = treeMap.positions(map)
```

`map.pick` *Tree mapping algorithm: pick trees manually*

Description

This function is meant to be used inside `treeMap`. It opens an interactive `rgl` plot where the user can specify tree locations by clicking.

Usage

```
map.pick(map = NULL, min_h = 1, max_h = 5)
```

Arguments

`map`	optional tree map to be manually updated.
`min_h, max_h`	`numeric` - height thresholds applied to filter a point cloud before processing.

`nnFilter` *Nearest neighborhood point filter*

Description

Remove isolated points from a `LAS` point cloud based on their neighborhood distances.

Usage

```
nnFilter(las, d = 0.05, n = 2)
```

Arguments

`las`	`LAS` object.
`d`	`numeric` - search radius.
`n`	`numeric` - number of neighbors within `d` distance a point must have to be kept in the output.

Value

`LAS` object.

Examples

```
file = system.file("extdata", "spruce.laz", package="TreeLS")
tls = readTLS(file)
nrow(tls@data)

nn_tls = nnFilter(tls, 0.05, 3)
nrow(nn_tls@data)
```

`ptm.knn` *Point metrics algorithm: K Nearest Neighbors metrics*

Description

This function is meant to be used inside `fastPointMetrics`. It calculates metrics for every point using its nearest neighbors (KNN).

Usage

```
ptm.knn(k = 20, r = 0)
```

Arguments

k	`numeric` - number of nearest points to search per neighborhood.
r	`numeric` - search radius limit. If `r == 0`, no distance limit is applied.

`ptm.voxel` *Point metrics algorithm: Voxel metrics*

Description

This function is meant to be used inside `fastPointMetrics`. It calculates metrics per voxel.

Usage

```
ptm.voxel(d = 0.1, exact = FALSE)
```

Arguments

d	`numeric` - voxel spacing, in point cloud units.
exact	`logical` - use exact voxel search? If `FALSE`, applies approximate voxel search using integer index hashing, much faster on large point clouds (several million points).

readTLS — *Import a point cloud file into a LAS object*

Description

Wrapper to read point cloud files straight into LAS objects. Reads *las* or *laz* files with `readLAS`, *ply* files with `read.las` and other file formats with `fread` (txt, xyz, 3d or any other table like format).

Usage

```
readTLS(file, col_names = NULL, ...)
```

Arguments

`file`	file path.
`col_names`	optional - `character` vector. Only used for table-like objects. It states the column names. If not set, only the 3 first columns will be used and assigned to the XYZ fields.
`...`	further arguments passed down to `readLAS`, `read.las` or `fread`.

Value

`LAS` object.

Examples

```
cloud = matrix(runif(300), ncol=3)
file = tempfile(fileext = '.txt')
fwrite(cloud, file)
tls = readTLS(file)
summary(tls)
```

setTLS — *(Re-)Create a `LAS` object depending on the input's type*

Description

Reset the input's header if it is a `LAS` object, or generate a new `LAS` from a table-like input. For more information, checkout `lidR::LAS`.

Usage

```
setTLS(cloud, col_names = NULL)
```

Arguments

`cloud`	`LAS`, `data.frame`, `matrix` or similar object to be converted.
`col_names`	optional - `character` vector. Only used for table-like objects. It states the column names. If not set, only the 3 first columns will be used and assigned to the XYZ fields.

Value

`LAS` object.

Examples

```
cloud = matrix(runif(300, 0, 10), ncol=3)
cloud = setTLS(cloud)
summary(cloud)
```

`sgt.bf.cylinder` *Stem segmentation algorithm: Brute Force cylinder fit*

Description

This function is meant to be used inside `stemSegmentation`. It applies a least squares cylinder fit algorithm in a RANSAC fashion over stem segments. More details are given in the sections below.

Usage

```
sgt.bf.cylinder(tol = 0.1, n = 10, conf = 0.95, inliers = 0.9, z_dev = 30)
```

Arguments

`tol`	`numeric` - tolerance offset between absolute radii estimates and hough transform estimates.
`n`	`numeric` - number of points selected on every RANSAC iteration.
`conf`	`numeric` - confidence level.
`inliers`	`numeric` - expected proportion of inliers among stem segments' point cloud chunks.
`z_dev`	`numeric` - maximum angle deviation for brute force cylinder estimation (`bf`), i.e. angle, in degrees (0-90), that a cylinder can be tilted in relation to a perfect vertival axis (`Z = c(0,0,1)`).

Brute Force Cylinder Fit

The brute force cylinder fit approach estimates the axis rotation angles by brute force combined with 2D ransac circle fit. The coordinates of a point cloud representing a single cylinder are iteratively rotated up to a pre defined threshold, and for every iteration a circle is estimated after rotation is performed. The rotation that minimizes the circle parameters the most is used to describe the axis direction of the cylinder with the circle's radius.

The parameters returned by the brute force cylinder fit method are:

- `X,Y`: 2D circle center coordinates after rotation
- `Radius`: 3D circle radius, in point cloud units
- `Error`: model circle error from the RANSAC least squares fit, after rotation
- `DX,DY`: absolute rotation angles (in degrees) applied to the X and Y axes, respectively
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

---

sgt.irls.circle — *Stem segmentation algorithm: Iterated Reweighted Least Squares circle fit*

Description

This function is meant to be used inside `stemSegmentation`. It applies a reweighted least squares circle fit algorithm using M-estimators in order to remove outlier effects.

Usage

```
sgt.irls.circle(tol = 0.1, n = 500)
```

Arguments

`tol`	`numeric` - tolerance offset between absolute radii estimates and hough transform estimates.
`n`	`numeric` - maximum number of points to sample for fitting stem segments.

Iterative Reweighted Least Squares (IRLS) Algorithm

irls `circle` or `cylinder` estimation methods perform automatic outlier assigning through iterative reweighting with M-estimators, followed by a Nelder-Mead optimization of squared distance sums to determine the best circle/cylinder parameters for a given point cloud. The reweighting strategy used in *TreeLS* is based on Liang et al. (2012). The Nelder-Mead algorithm implemented in Rcpp was provided by kthohr/optim.

Least Squares Circle Fit

The circle fit methods applied in *TreeLS* estimate the circle parameters (its center's XY coordinates and radius) from a pre-selected (denoised) set of points in a least squares fashion by applying either QR decompostion, used in combination with the RANSAC algorithm, or Nelder-Mead simplex optimization combined the IRLS approach.

The parameters returned by the circle fit methods are:

- `X,Y`: 2D circle center coordinates
- `Radius`: 2D circle radius, in point cloud units
- `Error`: model circle error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

`sgt.irls.cylinder` *Stem segmentation algorithm: Iterated Reweighted Least Squares cylinder fit*

Description

This function is meant to be used inside `stemSegmentation`. It applies a reweighted least squares cylinder fit algorithm using M-estimators and Nelder-Mead optimization in order to remove outlier effects.

Usage

```
sgt.irls.cylinder(tol = 0.1, n = 100)
```

Arguments

`tol`	`numeric` - tolerance offset between absolute radii estimates and hough transform estimates.
`n`	`numeric` - maximum number of points to sample for fitting stem segments.

Iterative Reweighted Least Squares (IRLS) Algorithm

irls `circle` or `cylinder` estimation methods perform automatic outlier assigning through iterative reweighting with M-estimators, followed by a Nelder-Mead optimization of squared distance sums to determine the best circle/cylinder parameters for a given point cloud. The reweighting strategy used in *TreeLS* is based on Liang et al. (2012). The Nelder-Mead algorithm implemented in Rcpp was provided by kthohr/optim.

Least Squares Cylinder Fit

The cylinder fit methods implemented in *TreeLS* estimate a 3D cylinder‘s axis direction and radius. The algorithm used internally to optimize the cylinder parameters is the Nelder-Mead simplex, which takes as objective function the model describing the distance from any point to a modelled cylinder‘s surface on a regular 3D cylinder point cloud:

$$D_p = |(p - q) \times a| - r$$

where:

- *Dp*: distance from a point to the model cylinder‘s surface
- *p*: a point on the cylinder‘s surface
- *q*: a point on the cylinder‘s axis
- *a*: unit vector of cylinder‘s direction
- *r*: cylinder‘s radius

The Nelder-Mead algorithm minimizes the sum of squared *Dp* from a set of points belonging to a stem segment - in the context of *TreeLS*.

The parameters returned by the cylinder fit methods are:

- `rho,theta,phi,alpha`: 3D cylinder estimated axis parameters (Liang et al. 2012)
- `Radius`: 3D cylinder radius, in point cloud units
- `Error`: model cylinder error from the least squares fit
- `AvgHeight`: average height of the stem segment’s points
- `N`: number of points belonging to the stem segment
- `PX,PY,PZ`: absolute center positions of the stem segment points, in point cloud units (used for plotting)

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

sgt.ransac.circle — *Stem segmentation algorithm: RANSAC circle fit*

Description

This function is meant to be used inside stemSegmentation. It applies a least squares circle fit algorithm in a RANSAC fashion over stem segments. More details are given in the sections below.

Usage

```
sgt.ransac.circle(tol = 0.1, n = 10, conf = 0.99, inliers = 0.8)
```

Arguments

tol	numeric - tolerance offset between absolute radii estimates and hough transform estimates.
n	numeric - number of points selected on every RANSAC iteration.
conf	numeric - confidence level.
inliers	numeric - expected proportion of inliers among stem segments' point cloud chunks.

Random Sample Consensus (RANSAC) Algorithm

The **RAN**dom **SA**mple **C**onsensus algorithm is a method that relies on resampling a data set as many times as necessary to find a subset comprised of only inliers - e.g. observations belonging to a desired model. The RANSAC algorithm provides a way of estimating the necessary number of iterations necessary to fit a model using inliers only, at least once, as shown in the equation:

$$k = log(1 - p)/log(1 - w^n)$$

where:

- k: number of iterations
- p: confidence level, i.e. desired probability of success
- w: proportion of inliers expected in the *full* dataset
- n: number of observations sampled on every iteration

The models reiterated in *TreeLS* usually relate to circle or cylinder fitting over a set of 3D coordinates, selecting the best possible model through the RANSAC algorithm

For more information, checkout this wikipedia page.

Least Squares Circle Fit

The circle fit methods applied in *TreeLS* estimate the circle parameters (its center's XY coordinates and radius) from a pre-selected (denoised) set of points in a least squares fashion by applying either QR decompostion, used in combination with the RANSAC algorithm, or Nelder-Mead simplex optimization combined the IRLS approach.

The parameters returned by the circle fit methods are:

- `X,Y`: 2D circle center coordinates
- `Radius`: 2D circle radius, in point cloud units
- `Error`: model circle error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

References

Olofsson, K., Holmgren, J. & Olsson, H., 2014. Tree stem and height measurements using terrestrial laser scanning and the RANSAC algorithm. Remote Sensing, 6(5), pp.4323–4344.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

`sgt.ransac.cylinder` *Stem segmentation algorithm: RANSAC cylinder fit*

Description

This function is meant to be used inside `stemSegmentation`. It applies a least squares cylinder fit algorithm in a RANSAC fashion over stem segments. More details are given in the sections below.

Usage

```
sgt.ransac.cylinder(tol = 0.1, n = 10, conf = 0.95, inliers = 0.9)
```

Arguments

`tol`	`numeric` - tolerance offset between absolute radii estimates and hough transform estimates.
`n`	`numeric` - number of points selected on every RANSAC iteration.
`conf`	`numeric` - confidence level.
`inliers`	`numeric` - expected proportion of inliers among stem segments' point cloud chunks.

Random Sample Consensus (RANSAC) Algorithm

The **RAN**dom **SA**mple **C**onsensus algorithm is a method that relies on resampling a data set as many times as necessary to find a subset comprised of only inliers - e.g. observations belonging to a desired model. The RANSAC algorithm provides a way of estimating the necessary number of iterations necessary to fit a model using inliers only, at least once, as shown in the equation:

$$k = log(1 - p)/log(1 - w^n)$$

where:

- k: number of iterations
- p: confidence level, i.e. desired probability of success
- w: proportion of inliers expected in the *full* dataset
- n: number of observations sampled on every iteration

The models reiterated in *TreeLS* usually relate to circle or cylinder fitting over a set of 3D coordinates, selecting the best possible model through the RANSAC algorithm

For more information, checkout this wikipedia page.

Least Squares Cylinder Fit

The cylinder fit methods implemented in *TreeLS* estimate a 3D cylinder‘s axis direction and radius. The algorithm used internally to optimize the cylinder parameters is the Nelder-Mead simplex, which takes as objective function the model describing the distance from any point to a modelled cylinder‘s surface on a regular 3D cylinder point cloud:

$$D_p = |(p - q) \times a| - r$$

where:

- *Dp*: distance from a point to the model cylinder‘s surface
- p: a point on the cylinder‘s surface
- q: a point on the cylinder‘s axis
- a: unit vector of cylinder‘s direction
- r: cylinder‘s radius

The Nelder-Mead algorithm minimizes the sum of squared *Dp* from a set of points belonging to a stem segment - in the context of *TreeLS*.

The parameters returned by the cylinder fit methods are:

- `rho,theta,phi,alpha`: 3D cylinder estimated axis parameters (Liang et al. 2012)
- `Radius`: 3D cylinder radius, in point cloud units
- `Error`: model cylinder error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment
- `PX,PY,PZ`: absolute center positions of the stem segment points, in point cloud units (used for plotting)

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Olofsson, K., Holmgren, J. & Olsson, H., 2014. Tree stem and height measurements using terrestrial laser scanning and the RANSAC algorithm. Remote Sensing, 6(5), pp.4323–4344.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

---

shapeFit *Point cloud cylinder/circle fit*

Description

Fits a 3D cylinder or 2D circle on a set of 3D points, retrieving the optimized parameters.

Usage

```
shapeFit(
  stem_segment = NULL,
  shape = "circle",
  algorithm = "ransac",
  n = 10,
  conf = 0.95,
  inliers = 0.9,
  n_best = 10,
  z_dev = 30
)
```

Arguments

`stem_segment`	`NULL` or a `LAS` object with a single stem segment. When `NULL` returns a parameterized function to be used as input in other functions (e.g. `tlsInventory`).
`shape`	`character`, either `"circle"` or `"cylinder"`.
`algorithm`	optimization method for estimating the shape's parameters. Currently available: `"ransac"`, `"irls"`, `"nm"`, `"qr"` (circle only) ,`"bf"` (cylinder only).
`n`	`numeric` - number of points selected on every RANSAC iteration.
`conf`	`numeric` - confidence level.
`inliers`	`numeric` - expected proportion of inliers among stem segments' point cloud chunks.
`n_best`	`integer` - estimate optimal RANSAC parameters as the median of the `n_best` estimations with lowest error.
`z_dev`	`numeric` - maximum angle deviation for brute force cylinder estimation (`bf`), i.e. angle, in degrees (0-90), that a cylinder can be tilted in relation to a perfect vertival axis (`Z = c(0,0,1)`).

Details

The `ransac` and `irls` methods are *robust*, which means they estimate the circle/cylinder parameters in a way that takes into consideration outlier effects (noise). If the input data is already noise free, the `nm` or `qr` algorithms can be used with as good reliability, while being much faster.

Least Squares Circle Fit

The circle fit methods applied in *TreeLS* estimate the circle parameters (its center's XY coordinates and radius) from a pre-selected (denoised) set of points in a least squares fashion by applying either QR decompostion, used in combination with the RANSAC algorithm, or Nelder-Mead simplex optimization combined the IRLS approach.

The parameters returned by the circle fit methods are:

- `X,Y`: 2D circle center coordinates
- `Radius`: 2D circle radius, in point cloud units
- `Error`: model circle error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

Least Squares Cylinder Fit

The cylinder fit methods implemented in *TreeLS* estimate a 3D cylinder‘s axis direction and radius. The algorithm used internally to optimize the cylinder parameters is the Nelder-Mead simplex, which takes as objective function the model describing the distance from any point to a modelled cylinder‘s surface on a regular 3D cylinder point cloud:

$$D_p = |(p - q) \times a| - r$$

where:

- *Dp*: distance from a point to the model cylinder‘s surface
- *p*: a point on the cylinder‘s surface
- *q*: a point on the cylinder‘s axis
- *a*: unit vector of cylinder‘s direction
- *r*: cylinder‘s radius

The Nelder-Mead algorithm minimizes the sum of squared *Dp* from a set of points belonging to a stem segment - in the context of *TreeLS*.

The parameters returned by the cylinder fit methods are:

- `rho,theta,phi,alpha`: 3D cylinder estimated axis parameters (Liang et al. 2012)
- `Radius`: 3D cylinder radius, in point cloud units
- `Error`: model cylinder error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment
- `PX,PY,PZ`: absolute center positions of the stem segment points, in point cloud units (used for plotting)

Random Sample Consensus (RANSAC) Algorithm

The **RAN**dom **SA**mple **C**onsensus algorithm is a method that relies on resampling a data set as many times as necessary to find a subset comprised of only inliers - e.g. observations belonging to a desired model. The RANSAC algorithm provides a way of estimating the necessary number of iterations necessary to fit a model using inliers only, at least once, as shown in the equation:

$$k = log(1 - p)/log(1 - w^n)$$

where:

- *k*: number of iterations
- *p*: confidence level, i.e. desired probability of success
- *w*: proportion of inliers expected in the *full* dataset
- *n*: number of observations sampled on every iteration

The models reiterated in *TreeLS* usually relate to circle or cylinder fitting over a set of 3D coordinates, selecting the best possible model through the RANSAC algorithm

For more information, checkout this wikipedia page.

Iterative Reweighted Least Squares (IRLS) Algorithm

irls `circle` or `cylinder` estimation methods perform automatic outlier assigning through iterative reweighting with M-estimators, followed by a Nelder-Mead optimization of squared distance sums to determine the best circle/cylinder parameters for a given point cloud. The reweighting strategy used in *TreeLS* is based on Liang et al. (2012). The Nelder-Mead algorithm implemented in Rcpp was provided by kthohr/optim.

Brute Force Cylinder Fit

The brute force cylinder fit approach estimates the axis rotation angles by brute force combined with 2D ransac circle fit. The coordinates of a point cloud representing a single cylinder are iteratively rotated up to a pre defined threshold, and for every iteration a circle is estimated after rotation is performed. The rotation that minimizes the circle parameters the most is used to describe the axis direction of the cylinder with the circle's radius.

The parameters returned by the brute force cylinder fit method are:

- `X,Y`: 2D circle center coordinates after rotation
- `Radius`: 3D circle radius, in point cloud units
- `Error`: model circle error from the RANSAC least squares fit, after rotation
- `DX,DY`: absolute rotation angles (in degrees) applied to the X and Y axes, respectively
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Olofsson, K., Holmgren, J. & Olsson, H., 2014. Tree stem and height measurements using terrestrial laser scanning and the RANSAC algorithm. Remote Sensing, 6(5), pp.4323–4344.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
segment = filter_poi(tls, Z > 1 & Z < 2)
pars = shapeFit(segment, shape='circle', algorithm='irls')

segment@data %$% plot(Y ~ X, pch=20, asp=1)
pars %$% points(X,Y,col='red', pch=3, cex=2)
pars %$% lines(c(X,X+Radius),c(Y,Y), col='red',lwd=2,lty=2)
```

`shapeFit.forks` *EXPERIMENTAL: Point cloud multiple circle fit*

Description

Search and fit multiple 2D circles on a point cloud layer from a single tree, i.e. a forked stem segment.

Usage

```
shapeFit.forks(
  dlas,
  pixel_size = 0.02,
  max_d = 0.4,
  votes_percentile = 0.7,
  min_density = 0.25,
  plot = FALSE
)
```

Arguments

`dlas`	`LAS` object.
`pixel_size`	`numeric` - pixel side length to discretize the point cloud layers while performing the Hough Transform circle search.
`max_d`	`numeric` - largest tree diameter expected in the point cloud.

votes_percentile
: numeric - use only estimates with more votes than votes_percentile.

min_density
: numeric - between 0 and 1 - minimum point density within a pixel evaluated on the Hough Transform - i.e. only *dense* point clousters will undergo circle search.

plot
: logical - plot the results?

---

smp.randomize *Point sampling algorithm: random sample*

Description

This function is meant to be used inside tlsSample. It selects points randomly, returning a fraction of the input point cloud.

Usage

```
smp.randomize(p = 0.5)
```

Arguments

p
: numeric - sampling probability (from 0 to 1).

---

smp.voxelize *Point sampling algorithm: systematic voxel grid*

Description

This function is meant to be used inside tlsSample. It selects one random point per voxel at a given spatial resolution.

Usage

```
smp.voxelize(spacing = 0.05)
```

Arguments

spacing
: numeric - voxel side length, in point cloud units.

`stemPoints` *Stem points classification*

Description

Classify stem points of all trees in a **normalized** point cloud. Stem denoising methods are prefixed by `stm`.

Usage

```
stemPoints(las, method = stm.hough())
```

Arguments

`las`	`LAS` object.
`method`	stem denoising algorithm. Currently available: `stm.hough`, `stm.eigen.knn` and `stm.eigen.voxel`.

Value

`LAS` object.

Examples

```
### single tree
file = system.file("extdata", "spruce.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  stemPoints(stm.hough(h_base = c(.5,2)))
plot(tls, color='Stem')

### entire forest plot
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

map = treeMap(tls, map.hough())
tls = treePoints(tls, map, trp.crop(circle=FALSE))
tls = stemPoints(tls, stm.hough(pixel_size = 0.03))
tlsPlot(tls)
```

`stemSegmentation` *Stem segmentation*

Description

Measure stem segments from a point cloud with assigned stem points. Stem segmentation methods are prefixed by `sgt`.

Usage

```
stemSegmentation(las, method = sgt.ransac.circle())
```

Arguments

`las`	`LAS` object.
`method`	stem segmentation algorithm. Currently available: `sgt.ransac.circle`, `sgt.ransac.cylinder`, `sgt.irls.circle`, `sgt.irls.cylinder` and `sgt.bf.cylinder`.

Details

All stem segmentation algorithms return estimations for every stem `Segment` of every `TreeID` (if the input `LAS` has multiple trees). For more details and a list of all outputs for each method check the sections below.

Value

signed `data.table` of stem segments.

Random Sample Consensus (RANSAC) Algorithm

The **RAN**dom **SA**mple **C**onsensus algorithm is a method that relies on resampling a data set as many times as necessary to find a subset comprised of only inliers - e.g. observations belonging to a desired model. The RANSAC algorithm provides a way of estimating the necessary number of iterations necessary to fit a model using inliers only, at least once, as shown in the equation:

$$k = log(1 - p)/log(1 - w^n)$$

where:

- k: number of iterations
- p: confidence level, i.e. desired probability of success
- w: proportion of inliers expected in the *full* dataset
- n: number of observations sampled on every iteration

The models reiterated in *TreeLS* usually relate to circle or cylinder fitting over a set of 3D coordinates, selecting the best possible model through the RANSAC algorithm

For more information, checkout this wikipedia page.

Iterative Reweighted Least Squares (IRLS) Algorithm

irls `circle` or `cylinder` estimation methods perform automatic outlier assigning through iterative reweighting with M-estimators, followed by a Nelder-Mead optimization of squared distance sums to determine the best circle/cylinder parameters for a given point cloud. The reweighting strategy used in *TreeLS* is based on Liang et al. (2012). The Nelder-Mead algorithm implemented in Rcpp was provided by kthohr/optim.

Least Squares Circle Fit

The circle fit methods applied in *TreeLS* estimate the circle parameters (its center's XY coordinates and radius) from a pre-selected (denoised) set of points in a least squares fashion by applying either QR decompostion, used in combination with the RANSAC algorithm, or Nelder-Mead simplex optimization combined the IRLS approach.

The parameters returned by the circle fit methods are:

- `X,Y`: 2D circle center coordinates
- `Radius`: 2D circle radius, in point cloud units
- `Error`: model circle error from the least squares fit
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

Least Squares Cylinder Fit

The cylinder fit methods implemented in *TreeLS* estimate a 3D cylinder‘s axis direction and radius. The algorithm used internally to optimize the cylinder parameters is the Nelder-Mead simplex, which takes as objective function the model describing the distance from any point to a modelled cylinder‘s surface on a regular 3D cylinder point cloud:

$$D_p = |(p - q) \times a| - r$$

where:

- *Dp*: distance from a point to the model cylinder‘s surface
- *p*: a point on the cylinder‘s surface
- *q*: a point on the cylinder‘s axis
- *a*: unit vector of cylinder‘s direction
- *r*: cylinder‘s radius

The Nelder-Mead algorithm minimizes the sum of squared *Dp* from a set of points belonging to a stem segment - in the context of *TreeLS*.

The parameters returned by the cylinder fit methods are:

- `rho,theta,phi,alpha`: 3D cylinder estimated axis parameters (Liang et al. 2012)
- `Radius`: 3D cylinder radius, in point cloud units
- `Error`: model cylinder error from the least squares fit

- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment
- `PX,PY,PZ`: absolute center positions of the stem segment points, in point cloud units (used for plotting)

Brute Force Cylinder Fit

The brute force cylinder fit approach estimates the axis rotation angles by brute force combined with 2D ransac circle fit. The coordinates of a point cloud representing a single cylinder are iteratively rotated up to a pre defined threshold, and for every iteration a circle is estimated after rotation is performed. The rotation that minimizes the circle parameters the most is used to describe the axis direction of the cylinder with the circle's radius.

The parameters returned by the brute force cylinder fit method are:

- `X,Y`: 2D circle center coordinates after rotation
- `Radius`: 3D circle radius, in point cloud units
- `Error`: model circle error from the RANSAC least squares fit, after rotation
- `DX,DY`: absolute rotation angles (in degrees) applied to the X and Y axes, respectively
- `AvgHeight`: average height of the stem segment's points
- `N`: number of points belonging to the stem segment

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Olofsson, K., Holmgren, J. & Olsson, H., 2014. Tree stem and height measurements using terrestrial laser scanning and the RANSAC algorithm. Remote Sensing, 6(5), pp.4323–4344.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize

tls = stemPoints(tls, stm.hough())
sgt = stemSegmentation(tls, sgt.ransac.circle(n=20))
tlsPlot(tls, sgt)
```

`stm.eigen.knn` *Stem denoising algorithm: KNN eigen decomposition + point normals intersections voting*

Description

This function is meant to be used inside `stemPoints`. It filters points based on their nearest neighborhood geometries (check `fastPointMetrics`) and assign them to stem patches if reaching a voxel with enough votes.

Usage

```
stm.eigen.knn(
  h_step = 0.5,
  max_curvature = 0.1,
  max_verticality = 10,
  voxel_spacing = 0.025,
  max_d = 0.5,
  votes_weight = 0.2,
  v3d = FALSE
)
```

Arguments

`h_step`	`numeric` - height interval to perform point filtering/assignment/classification.
`max_curvature`	`numeric` - maximum curvature (from 0 to 1) accepted when filtering a point neighborhood.
`max_verticality`	`numeric` - maximum deviation of a point neighborhood's orientation from an absolute vertical axis (Z = c(0,0,1)), in *degrees* (from 0 to 90).
`voxel_spacing`	`numeric` - voxel (or pixel) spacing for counting point normals intersections.
`max_d`	`numeric` - largest tree diameter expected in the point cloud.
`votes_weight`	`numeric` - fraction of votes a point neighborhood needs do reach in order to belong to a stem (applied for every *TreeID*), in relation to the voxel with most votes with same *TreeID*.
`v3d`	`logical` - count votes in 3D voxels (TRUE) or 2D pixels (FALSE).

Eigen Decomposition of Point Neighborhoods

Point filtering/classification methods that rely on eigen decomposition rely on shape indices calculated for point neighborhoods (knn or voxel). To derive these shape indices, eigen decomposition is performed on the XYZ columns of a point cloud patch. Metrics related to object curvature are calculated upon ratios of the resulting eigen values, and metrics related to object orientation are caltulated from approximate normals obtained from the eigen vectors.

For instance, a point neighborhood that belongs to a perfect flat surface will have all of its variance explained by the first two eigen values, and none explained by the third eigen value. The 'normal' of such surface, i.e. the vector oriented in the direction orthogonal to the surface, is therefore represented by the third eigenvector.

Methods for both tree mapping and stem segmentation use those metrics, so in order to speed up the workflow one might apply `fastPointMetrics` to the point cloud before other methods. The advantages of this approach are that users can parameterize the point neighborhoods themselves when calculating their metrics. Those calculations won't be performed again internally in the tree mapping or stem denoising methods, reducing the overall processing time.

Radius Estimation Through Normal Vectors Intersection

`stemPoints` methods that filter points based on eigen decomposition metrics (knn or voxel) provide a rough estimation of stem segments radii by splitting every stem segment into a local voxel space and counting the number of times that point normals intersect on every voxel (votes). Every stem point then has a radius assigned to it, corresponding to the distance between the point and the voxel with most votes its normal intersects. The average of all points' radii in a stem segment is the segment's radius. For approximately straight vertical stem segments, the voting can be done in 2D (pixels).

The point normals of this method are extracted from the eigen vectors calculated by `fastPointMetrics`. On top of the point metrics used for stem point filtering, the following fields are also added to the `LAS` object:

- `Votes`: number of normal vector intersections crossing the point's normal at its estimated center
- `VotesWeight`: ratio of (votes count) / (highest votes count) per *TreeID*
- `Radius`: estimated stem segment radius

This method was inspired by the denoising algorithm developed by Olofsson & Holmgren (2016), but it is not an exact reproduction of their work.

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Olofsson, K. & Holmgren, J., 2016. Single Tree Stem Profile Detection Using Terrestrial Laser Scanner Data, Flatness Saliency Features and Curvature Properties. Forests, 7, 207.

`stm.eigen.voxel` *Stem denoising algorithm: Voxel eigen decomposition + point normals intersections voting*

Description

This function is meant to be used inside `stemPoints`. It filters points based on their voxel geometries (check `fastPointMetrics`) and assign them to stem patches if reaching a voxel with enough votes.

Usage

```
stm.eigen.voxel(
  h_step = 0.5,
  max_curvature = 0.1,
  max_verticality = 10,
  voxel_spacing = 0.025,
  max_d = 0.5,
  votes_weight = 0.2,
  v3d = FALSE
)
```

Arguments

- `h_step` — `numeric` - height interval to perform point filtering/assignment/classification.
- `max_curvature` — `numeric` - maximum curvature (from 0 to 1) accepted when filtering a point neighborhood.
- `max_verticality` — `numeric` - maximum deviation of a point neighborhood's orientation from an absolute vertical axis (Z = c(0,0,1)), in *degrees* (from 0 to 90).
- `voxel_spacing` — `numeric` - voxel side length.
- `max_d` — `numeric` - largest tree diameter expected in the point cloud.
- `votes_weight` — `numeric` - fraction of votes a point neighborhood needs do reach in order to belong to a stem (applied for every *TreeID*), in relation to the voxel with most votes with same *TreeID*.
- `v3d` — `logical` - count votes in 3D voxels (TRUE) or 2D pixels (FALSE).

Eigen Decomposition of Point Neighborhoods

Point filtering/classification methods that rely on eigen decomposition rely on shape indices calculated for point neighborhoods (knn or voxel). To derive these shape indices, eigen decomposition is performed on the XYZ columns of a point cloud patch. Metrics related to object curvature are calculated upon ratios of the resulting eigen values, and metrics related to object orientation are caltulated from approximate normals obtained from the eigen vectors.

For instance, a point neighborhood that belongs to a perfect flat surface will have all of its variance explained by the first two eigen values, and none explained by the third eigen value. The 'normal' of such surface, i.e. the vector oriented in the direction orthogonal to the surface, is therefore represented by the third eigenvector.

Methods for both tree mapping and stem segmentation use those metrics, so in order to speed up the workflow one might apply `fastPointMetrics` to the point cloud before other methods. The advantages of this approach are that users can parameterize the point neighborhoods themselves when calculating their metrics. Those calculations won't be performed again internally in the tree mapping or stem denoising methods, reducing the overall processing time.

Radius Estimation Through Normal Vectors Intersection

`stemPoints` methods that filter points based on eigen decomposition metrics (knn or voxel) provide a rough estimation of stem segments radii by splitting every stem segment into a local voxel space and counting the number of times that point normals intersect on every voxel (votes). Every stem point then has a radius assigned to it, corresponding to the distance between the point and the voxel with most votes its normal intersects. The average of all points' radii in a stem segment is the segment's radius. For approximately straight vertical stem segments, the voting can be done in 2D (pixels).

The point normals of this method are extracted from the eigen vectors calculated by `fastPointMetrics`. On top of the point metrics used for stem point filtering, the following fields are also added to the `LAS` object:

- `Votes`: number of normal vector intersections crossing the point's normal at its estimated center
- `VotesWeight`: ratio of (votes count) / (highest votes count) per *TreeID*
- `Radius`: estimated stem segment radius

This method was inspired by the denoising algorithm developed by Olofsson & Holmgren (2016), but it is not an exact reproduction of their work.

References

Liang, X. et al., 2012. Automatic stem mapping using single-scan terrestrial laser scanning. IEEE Transactions on Geoscience and Remote Sensing, 50(2), pp.661–670.

Olofsson, K. & Holmgren, J., 2016. Single Tree Stem Profile Detection Using Terrestrial Laser Scanner Data, Flatness Saliency Features and Curvature Properties. Forests, 7, 207.

---

stm.hough *Stem denoising algorithm: Hough Transform*

Description

This function is meant to be used inside `stemPoints`. It applies an adapted version of the Hough Transform for circle search. Mode details are given in the sections below.

Usage

```
stm.hough(
  h_step = 0.5,
  max_d = 0.5,
  h_base = c(1, 2.5),
  pixel_size = 0.025,
  min_density = 0.1,
  min_votes = 3
)
```

Arguments

- `h_step` — `numeric` - height interval to perform point filtering/assignment/classification.
- `max_d` — `numeric` - largest tree diameter expected in the point cloud.
- `h_base` — `numeric` vector of length 2 - tree base height interval to initiate circle search.
- `pixel_size` — `numeric` - pixel side length to discretize the point cloud layers while performing the Hough Transform circle search.
- `min_density` — `numeric` - between 0 and 1 - minimum point density within a pixel evaluated on the Hough Transform - i.e. only *dense* point clousters will undergo circle search.
- `min_votes` — `integer` - Hough Transform parameter - minimum number of circle intersections over a pixel to assign it as a circle center candidate.

`LAS@data` Special Fields

Meaninful new fields in the output:

- `Stem`: `TRUE` for stem points
- `Segment`: stem segment number (from bottom to top and nested with TreeID)
- `Radius`: approximate radius of the point's stem segment estimated by the Hough Transform - always a multiple of the `pixel_size`
- `Votes`: votes received by the stem segment's center through the Hough Transform

Adapted Hough Transform

The Hough Transform circle search algorithm used in TreeLS applies a constrained circle search on discretized point cloud layers. Tree-wise, the circle search is recursive, in which the search for circle parameters of a stem section is constrained to the *feature space* of the stem section underneath it. Initial estimates of the stem's *feature space* are performed on a *baselise* stem segment - i.e. a low height interval where a tree's bole is expected to be clearly visible in the point cloud. The algorithm is described in detail by Conto et al. (2017).

This adapted version of the algorithm is very robust against outliers, but not against forked or leaning stems.

References

Olofsson, K., Holmgren, J. & Olsson, H., 2014. Tree stem and height measurements using terrestrial laser scanning and the RANSAC algorithm. Remote Sensing, 6(5), pp.4323–4344.

Conto, T. et al., 2017. Performance of stem denoising and stem modelling algorithms on single tree point clouds from terrestrial laser scanning. Computers and Electronics in Agriculture, v. 143, p. 165-176.

`tlsCrop` *Point cloud cropping*

Description

Returns a cropped point cloud of all points inside or outside specified boundaries of circle or square shapes.

Usage

```
tlsCrop(las, x, y, len, circle = TRUE, negative = FALSE)
```

Arguments

`las`	`LAS` object.
`x, y`	`numeric` - X and Y center coordinates of the crop region.
`len`	`numeric` - if `circle` = `TRUE`, `len` is the circle's radius, otherwise it is the side length of a square.
`circle`	`logical` - crops a circle (if `TRUE`) or a square.
`negative`	`logical` - if `TRUE`, returns all points **outside** the specified circle/square perimeter.

Value

`LAS` object.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)

tls = tlsCrop(tls, 2, 3, 1.5, TRUE, TRUE)
plot(tls)

tls = tlsCrop(tls, 5, 5, 5, FALSE, FALSE)
plot(tls)
```

tlsInventory *Extract forest inventory metrics from a point cloud*

Description

Estimation of diameter and height tree-wise for normalized point clouds with assigned stem points.

Usage

```
tlsInventory(
  las,
  dh = 1.3,
  dw = 0.5,
  hp = 1,
  d_method = shapeFit(shape = "circle", algorithm = "ransac", n = 15, n_best = 20)
)
```

Arguments

las	LAS object.
dh	numeric - height layer (above ground) to estimate stem diameters, in point cloud units.
dw	numeric - height layer width, in point cloud units.
hp	numeric - height percentile to extract per tree (0-1). Use 1 for top height, i.e. the highest point.
d_method	parameterized shapeFit function, i.e. method to use for diameter estimation.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

map = treeMap(tls, map.hough())
tls = treePoints(tls, map, trp.crop(circle=FALSE))
tls = stemPoints(tls, stm.hough())

dmt = shapeFit(shape = 'circle', algorithm='ransac', n=20)
inv = tlsInventory(tls, d_method = dmt)
tlsPlot(tls, inv)
```

tlsNormalize *Normalize a TLS point cloud*

Description

Fast normalization of TLS point clouds based on a Digital Terrain Model (DTM) of the ground points. If the input's ground points are not yet classified, the csf algorithm is applied internally.

Usage

```
tlsNormalize(las, min_res = 0.25, keep_ground = TRUE)
```

Arguments

las	LAS object.
min_res	numeric - minimum resolution of the DTM used for normalization, in point cloud units.
keep_ground	logical - if FALSE removes the ground points from the output.

Value

LAS object.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file)
plot(tls)
rgl::axes3d(col='white')

tls = tlsNormalize(tls, 0.5, FALSE)
plot(tls)
rgl::axes3d(col='white')
```

tlsPlot *Plot* TreeLS *outputs*

Description

Plot the outputs of *TreeLS* methods on the same scene using rgl.

Usage

```
tlsPlot(..., fast = FALSE, tree_id = NULL, segment = NULL)

add_segmentIDs(x, las, ...)

add_treeIDs(x, las, ...)

add_treeMap(x, las, ...)

add_treePoints(x, las, color_func = pastel.colors, ...)

add_stemPoints(x, las, ...)

add_stemSegments(x, stems_data_table, color = "white", fast = FALSE)

add_tlsInventory(x, inventory_data_table, color = "white", fast = FALSE)
```

Arguments

`...`	in `tlsPlot`: any object returned from a *TreeLS* method. In the `add_*` methods: parameters passed down to 3D plotting `rgl` functions.
`fast`	`logical`, use `TRUE` to plot spheres representing tree diameters or `FALSE` to plot detailed 3D cylinders.
`tree_id`	`numeric` - plot only the tree matching this tree id.
`segment`	`numeric` - plot only stem segments matching this segment id.
`x`	output from `plot` or `tlsPlot`
`las`	`LAS` object.
`color_func`	color palette function used in `add_treePoints`.
`stems_data_table, inventory_data_table`	`data.table` objects generated by `stemSegmentation` and `tlsInventory`.
`color`	color of 3D objects.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  stemPoints(stm.hough())

dmt = shapeFit(shape = 'circle', algorithm='ransac', n=20)
inv = tlsInventory(tls, d_method = dmt)

### quick plot
tlsPlot(tls, inv)

### customizable plots
x = plot(tls)
```

```
add_stemPoints(x, tls, color='red', size=3)
add_tlsInventory(x, inv, color='yellow')
add_segmentIDs(x, tls, color='white', cex=2, pos=4)
```

---

tlsRotate *Rotate point cloud to fit a horizontal ground plane*

Description

Check for ground points and rotates the point cloud aligning its ground surface to a horizontal plane (XY). This function is especially useful for point clouds not georeferenced or generated through mobile scanning, which might present a tilted coordinate system.

Usage

```
tlsRotate(las)
```

Arguments

las	LAS object.

Value

LAS object.

---

tlsSample *Resample a point cloud*

Description

Applies a sampling algorithm to reduce a point cloud's density. Sampling methods are prefixed by smp.

Usage

```
tlsSample(las, method = smp.voxelize())
```

Arguments

las	LAS object.
method	point sampling algorithm. Currently available: smp.voxelize and smp.randomize

Value

LAS object.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
nrow(tls@data)

### sample points systematically from a 3D voxel grid
vx = tlsSample(tls, smp.voxelize(0.05))
nrow(vx@data)

### sample half of the points randomly
rd = tlsSample(tls, smp.randomize(0.5))
nrow(rd@data)
```

---

tlsTransform — *Simple operations on point cloud objects*

Description

Apply transformations to the XYZ axes of a point cloud.

Usage

```
tlsTransform(
  las,
  xyz = c("X", "Y", "Z"),
  bring_to_origin = FALSE,
  rotate = FALSE
)
```

Arguments

las	LAS object.
xyz	character vector of length 3 - LAS' columns to be reassigned as XYZ, respectively. Use minus signs to mirror an axis' coordinates - more details in the sections below.
bring_to_origin	logical - force point cloud origin to match c(0,0,0)? If TRUE, clears the header of the LAS object.
rotate	logical - rotate the point cloud to align the ground points horizontally (as in tlsRotate)?

Value

LAS object.

XYZ Manipulation

The xyz argument can take a few different forms. It is useful for shifting axes positions in a point cloud or to mirror an axis' coordinates. All axes characters can be entered in lower or uppercase and also be preceded by a minus sign ('-') to reverse its coordinates.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file)
bbox(tls)
range(tls$Z)

### swap the Y and Z axes
zy = tlsTransform(tls, c('x', 'z', 'y'))
bbox(zy)
range(zy$Z)

### return an upside down point cloud
ud = tlsTransform(tls, c('x', 'y', '-z'))
bbox(ud)
range(ud$Z)
plot(zy)

### mirror all axes, then set the point cloud's starting point as the origin
rv = tlsTransform(tls, c('-x', '-y', '-z'), bring_to_origin=TRUE)
bbox(rv)
range(rv$Z)
```

---

treeMap — *Map tree occurrences from TLS data*

Description

Estimates tree occurrence regions from a **normalized** point cloud. Tree mapping methods are prefixed by map.

Usage

```
treeMap(las, method = map.hough(), merge = 0.2, positions_only = FALSE)
```

Arguments

las	LAS object.
method	tree mapping algorithm. Currently available: map.hough, map.eigen.knn, map.eigen.voxel and map.pick.
merge	numeric - parameter passed down to treeMap.merge (if merge > 0).
positions_only	logical - if TRUE returns only a 2D tree map as a data.table.

Value

signed LAS or data.table.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

x = plot(tls)

map = treeMap(tls, map.hough(h_step = 1, max_h = 4))
add_treeMap(x, map, color='red')

xymap = treeMap.positions(map)
```

---

treeMap.merge *Merge tree coordinates too close on* treeMap *outputs.*

Description

Check all tree neighborhoods and merge TreeIDs which are too close in a treeMap's object.

Usage

```
treeMap.merge(map, d = 0.2)
```

Arguments

map	object generated by treeMap.
d	numeric - distance threshold.

Details

The d parameter is a relative measure of close neighbors. Sorting all possible pairs by distance from a tree map, the merge criterion is that none of the closest pairs should be distant less than the next closest pair's distance minus d. This method is useful when merging forked stems or point clusters from plots with too much understory, especially if those are from forest stands with regularly spaced trees.

treeMap.positions *Convert a tree map to a 2D* data.table

Description

Extracts the tree XY positions from a *treeMap* output.

Usage

```
treeMap.positions(map, plot = TRUE)
```

Arguments

map	object generated by treeMap.
plot	logical - plot the tree map?

Value

signed data.table of tree IDs and XY coordinates.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

x = plot(tls)

map = treeMap(tls, map.hough(h_step = 1, max_h = 4))
add_treeMap(x, map, color='red')

xymap = treeMap.positions(map)
```

treePoints *Classify individual tree regions in a point cloud*

Description

Assigns TreeIDs to a LAS object based on coordinates extracted from a treeMap object. Tree region segmentation methods are prefixed by trp.

Usage

```
treePoints(las, map, method = trp.voronoi())
```

Arguments

`las`	`LAS` object.
`map`	object generated by `treeMap`.
`method`	tree region algorithm. Currently available: `trp.voronoi` and `trp.crop`.

Value

`LAS` object.

Examples

```
file = system.file("extdata", "pine_plot.laz", package="TreeLS")
tls = readTLS(file) %>%
  tlsNormalize %>%
  tlsSample

map = treeMap(tls, map.hough())
tls = treePoints(tls, map, trp.crop(circle=FALSE))

x = plot(tls, size=1)
add_treePoints(x, tls, size=2)
add_treeIDs(x, tls, color='yellow', cex=2)
```

`trp.crop` *Tree points algorithm: fixed size patches.*

Description

This function is meant to be used inside `treePoints`. Assign points to a `TreeID` inside circles/squares of fixed area around `treeMap` coordinates.

Usage

```
trp.crop(l = 1, circle = TRUE)
```

Arguments

`l`	`numeric` - circle radius or square side length.
`circle`	`logical` - assign `TreeID`s to circular (`TRUE`) or squared (`FALSE`) patches.

`trp.voronoi` *Tree points algorithm: voronoi polygons.*

Description

This function is meant to be used inside `treePoints`. Assign **all** points to a `TreeID` based on their closest `treeMap` coordinate.

Usage

```
trp.voronoi()
```

`writeTLS` *Export* TreeLS *point clouds to las/laz files*

Description

Wrapper to `writeLAS`. This function automatically adds new data columns as extra bytes to the written las/laz file using `add_lasattribute` internally.

Usage

```
writeTLS(las, file, col_names = NULL, index = FALSE)
```

Arguments

`las`	`LAS` object.
`file`	file path.
`col_names`	column names from *las* that you wish to export. If left empty, all columns not listed among `the standard LAS attributes` are added to the file.
`index`	`logical` - write lax file also.

Examples

```
file = system.file("extdata", "pine.laz", package="TreeLS")
tls = readTLS(file) %>% fastPointMetrics#'
tls_file = tempfile(fileext = '.laz')
writeTLS(tls, tls_file)

up_tls = readTLS(tls_file)
summary(up_tls)
```

Index