Package 'lidR'

September 20, 2017

Type Package

Title Airborne LiDAR Data Manipulation and Visualization for Forestry Applications

Version 1.3.1

Date 2017-10-30

Description Airborne LiDAR (Light Detection and Ranging) interface for data manipulation and visualization. Read/write 'las' and 'laz' files, computation of metrics in area based approach, point filtering, artificial point reduction, classification from geographic data, normalization, individual tree segmentation and other manipulations.

URL https://github.com/Jean-Romain/lidR

BugReports https://github.com/Jean-Romain/lidR/issues

License GPL-3

Depends R (>= 3.1.0),magrittr,methods

Imports rlas (>= 1.1.0),rgl,stats,utils,grDevices,tools,parallel,raster,rgeos, data.table,Rcpp,lazyeval,geometry,gstat,sp,settings,gdalUtils,stringr, memoise,mapview,mapedit

Suggests rgdal,testthat,EBImage,hexbin

LazyData true

RoxygenNote 6.0.1

LinkingTo Rcpp,RcppProgress

Encoding UTF-8

biocViews

Collate 'RcppExports.R' 'catalog_apply.r' 'catalog_index.r' 'catalog_makecluster.r' 'catalog_query.r' 'catalog_reshape.r' 'catalog_select.r' 'class-lasheader.r' 'class-las.r' 'class-lascatalog.r' 'constant.R' 'deprecated.r' 'grid_canopy.r' 'grid_catalog.r' 'grid_density.r' 'grid_hexametrics.r' 'grid_metrics.r' 'grid_metrics3d.r'

'grid_terrain.r' 'grid_tincanopy.r' 'lasaggreagte.r'
'lascheck.r' 'lasclassify.r' 'lasclip.r' 'lascolor.r'
'lasdecimate.r' 'lasfilter.r' 'lasground.r' 'lasindentify.r'
'lasmetrics.r' 'lasnormalize.r' 'lasroi.r' 'lastrees.r'
'lidRError.r' 'metrics.r' 'metrics_canopy_roughness.r'
'mutatebyref.r' 'options.r' 'plot.catalog.r' 'plot.las.r'
'plot.lashexametrics.r' 'plot.lasmetrics.r'
'plot.lasmetrics3d.r' 'plot3d.r' 'readLAS.r' 'subcircled.r'
'tree_metrics.r' 'utils_colors.r' 'utils_geometry.r'
'utils_interpolations.r' 'utils_misc.r' 'utils_progress.r'
'utils_projection.r' 'utils_spatial.r' 'utils_typecast.r'
'writeLAS.r' 'zzz.r'

NeedsCompilation yes

Author Jean-Romain Roussel [aut, cre, cph],
David Auty [aut, ctb] (Reviews the documentation),
Florian De Boissieu [ctb] (Fixed bugs and improved catalog features),
Andrew Sánchez Meador [ctb] (Helped to review the documentation)

Maintainer Jean-Romain Roussel <jean-romain.roussel.1@ulaval.ca>

Repository CRAN

Date/Publication 2017-09-20 18:09:10 UTC

R topics documented:

`-,LAS,RasterLayer-method`

Convenient operator to lasnormalize

Description

Convenient operator to lasnormalize

Usage

```
## S4 method for signature 'LAS,RasterLayer'
e1 - e2
```

Arguments

e1	a LAS object
e2	a RasterLayer

---

area *Surface covered by a* LAS *object or by a* LAScatalog.

Description

The area is computed with the convex hull for LAS objects or x,y coordinates. If the data is not convex, the resulting area is only an approximation. The area is computed as the sum of the extents of each file for a LAScatalog.

Usage

```
area(x, y)
```

Arguments

x	An object of the class LAS or LAScatalog or numeric
y	If x is numeric, then provide also y.

Value

numeric. The area of the object computed in the same units as the coordinate reference system

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
las = readLAS(LASfile)
area(las)
```

as.lasmetrics *Set the class 'lasmetrics' to a data.frame or a data.table*

Description

Set the class lasmetrics to a data.frame. Useful when reading data from a file. In this case the data.frame does not have the class lasmetrics and cannot easily be plotted or transformed into a raster

Usage

```
as.lasmetrics(x, res)
```

Arguments

x	A data.frame or a data.table
res	numeric the original resolution

Value

Nothing. Object is updated in place by reference.

See Also

Other cast: as.raster.lasmetrics, as.spatial

as.raster.lasmetrics *Transform a* lasmetrics *object into a spatial* RasterLayer *object*

Description

Transform a lasmetrics object into a spatial RasterLayer object

Usage

```
## S3 method for class 'lasmetrics'
as.raster(x, z = NULL, fun.aggregate = mean, ...)
```

Arguments

x	a lasmetrics object
z	character. If 3 columns or more, the names of the field to extract. If NULL returns a RasterStack instead of a RasterLayer.
fun.aggregate	Should the data be aggregated before casting? If the table does not contain a single observation for each cell, then aggregation defaults to mean value with a message.
...	Internal use only.

Value

A RasterLayer object from package **raster** or a RasterStack if there are several layers.

See Also

grid_metrics grid_canopy grid_canopy raster

Other cast: `as.lasmetrics`, `as.spatial`

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

meanHeight = grid_metrics(lidar, mean(Z))
rmeanHeight = as.raster(meanHeight)
```

---

as.spatial *Transform a lidR object into sp object*

Description

LAS, LAScatalog, lasmetrics are transformed respectively into SpatialPointsDataFrame, SpatialPolygonsDataFrame, SpatialPixelsDataFrame

Usage

```
as.spatial(x)
```

Arguments

x	an object from the lidR package

Value

An object from sp

See Also

Other cast: `as.lasmetrics`, `as.raster.lasmetrics`

catalog *Build a catalog of las tiles/files*

Description

Build a LAScatalog object from a folder name. A catalog is the representation of a set of las files. A computer cannot load all the data at the same time. A catalog is a simple way to manage all the file sequentially reading only the headers.

Usage

```
catalog(folder, ...)
```

Arguments

folder	string. The path of a folder containing a set of .las files
...	Extra parameters to list.files. Typically 'recursive = TRUE'.

Value

A LAScatalog object

See Also

LAScatalog-class plot catalog_apply catalog_queries

catalog_apply *Apply a user-defined function to an entire catalog in a continuous way*

Description

This function enables application of a user-defined routine over an entire catalog using a multi-core process. When a user has a dataset organized into several files, it applies the user-defined function to the entire catalog by automatically splitting it into several clusters. The clustering pattern can be either split into a set of squared areas or split by file. The clustering pattern can be modified using the global catalog options with catalog_options. The "Examples" section describes the procedure for applying functions to the catalog, beginning with data loading (see example).

Warning: there is a mechanism to load buffered data and to avoid edge artifacts, but no mechanism to remove the buffer after applying user-defined functions, since this task is very specific to each process. See section "Edge artifacts".

lidR supports .lax files. Computation speed will be *signifcantly* improved with a spatial index.

Usage

```
catalog_apply(ctg, func, func_args = NULL, ...)
```

Arguments

ctg	A LAScatalog object.
func	A user-defined function for which the first input is a LAS object.
func_args	A list of extra arguments to pass in the function 'func'.
...	Any argument available in readLAS to reduce the amount of data loaded.

Edge artifacts

It is very important to take precautions to avoid 'edge artifacts' when processing LiDAR tiles. If the points from neighboring tiles are not included during certain processes, this could create 'edge artifacts' at the tile boundaries. For example, empty or incomplete pixels in a rasterization process. The lidR package provides internal tools to load buffered data. However, there is no mechanism to remove the results computed in the buffered area since this task depends on the output of the user-defined function. Therefore, depending on the metric being computed, some output results could appear several times.

The LAS object received by the user-defined function has a special column called 'buffer_side' which indicates, for each point, if it comes from a buffered area or not. Points from non-buffered areas have a 'buffer_side' value of zero, while points from buffered areas have a 'buffer_side' value of 1, 2, 3 or 4, where 1 is the bottom buffer and 2, 3 and 4 are the left, top and right buffers, respectively (see example).

Examples

```
# Visit http://jean-romain.github.io/lidR/wiki for an illustrated and commented
# version of this example.
# This is a dummy example. It is more efficient to load the entire file than
# splitting it into several pieces to process, even when using multiple cores.

# 1. Build a project (here, a single file catalog for the purposes of this example).
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
project = catalog(LASfile)
plot(project)

# 2. Set some global catalog options
# For this dummy example, the clustering size is 80 m and the buffer is 15 m using
# a single core (because this example is run on the CRAN server when the package is submitted).
catalog_options(buffer = 15, multicore = 1, tiling_size = 120)

# 3. Load the shapefile needed to filter your points.
folder <- system.file("extdata", "", package="lidR")
lake_shp = rgdal::readOGR(folder, "lake_polygons_UTM17")

# 4. Build the function that analyzes each cluster of the catalog.
# The function's first argument is a LAS object. The internal routine takes care of
```

```
# this part. The other arguments can be freely choosen by the user. See the following
# template:
tree_area = function(las, lake)
{
  # The las argument is a LAS object with each field loaded and an extra column 'buffer'

  # Associate geographic data with lidar points
  lasclassify(las, lake, field = "lake")

  # filter lakes, and low elevation points
  las %<>% lasfilter(lake == FALSE, Z > 4)

  if (is.null(las))
    return(NULL)

  # segment trees (in this example the low point density does not enable
  # accurate segmentation of trees. This is just a proof-of-concept)
  lastrees(las, algorithm = "li2012")

  # Here we used the function tree_metric to compute some metrics for each tree. This
  # function is defined later in the global environment.
  m = tree_metrics(las, myMetrics(X, Y, Z, buffer))

 # If min buffer is 0 it means the trees were at least partly in the non-buffered area, so we
  # want to keep these trees.
  # However, the trees that are on the edge of the buffered area will be counted
  # twice. So we must remove the trees on the right side and on the top side of the buffer
  # If max buffer is <= 2 it means that the trees belong inside the area of interest, on
  # the left side or the bottom side, or both.
  m = m[minbuff == 0 & maxbuff <= 2]

  # Remove buffering information that is no longer useful
  m[, c("minbuff","maxbuff") := NULL]

  return(m)
}

# This function enables users to extract, for a single tree, the position of the highest point
# and some information about the buffering position of the tree. The function tree_metrics takes
# care of mapping along each tree.
myMetrics <- function(x, y, z, buff)
{
  i = which.max(z)
  xcenter = x[i]
  ycenter = y[i]
  A = area(x,y)
  minbuff = min(buff)
  maxbuff = max(buff)

  return(
    list(
      x = xcenter,
      y = ycenter,
```

```
      area = A,
      minbuff = minbuff,
      maxbuff = maxbuff
    ))
}

# Everything is now well defined, so now we can process over an entire catalog with
# hundreds of files (but in this example we use just one file...)

# 4. Process the project. The arguments of the user-defined function must
# belong in a labelled list. We also pass extra arguments to the function readLAS
# to load only X, Y and Z coordinates. This way we save a huge amount of memory, which
# can be used for the current process.
fargs = list(lake = lake_shp)
output = catalog_apply(project, tree_area, fargs, XYZonly = TRUE)

# 5. Post-process the output result (depending on the output computed). Here, each value
# of the list is a data.table, so rbindlist does the job:
output = data.table::rbindlist(output)

output %$% plot(x,y, cex = sqrt(area/pi)/5, asp = 1)
```

`catalog_options` *Options Settings for the catalog tools*

Description

Allow the user to set and examine a variety of global options that affect the way in which lidR processes an entire catalog.

Usage

```
catalog_options(...)

catalog_reset()
```

Arguments

`...`	Option names to retrieve option values or `[key]=[value]` pairs to set options.

Supported options

The following options are supported:

- `progress` (`logical`) Display progress bar. Default is TRUE.
- `buffer` (numeric) - When applying a function to an entire catalog sequentially processing sub-areas (clusters) some algorithms (such as grid_terrain) require a buffer around the area to avoid edge effects. Default is 15 m.

- `multicore` (numeric) - For parallel processes, fix the number of cores to use. Default is the number of cores you have.
- `tiling_size` (numeric) - To process an entire catalog, the algorithm splits the dataset into several square sub-areas (clusters) to process them sequentially. This is the size of each square cluster. Default is 1000 (1 km^2).
- `by_file` (logical) - This option overwrites the option `tiling_size`. Instead of processing the catalog by arbitrary split areas, it forces processing by file. Buffering is still available.
- `return_virtual_raster` (logical) - Functions which return raster-like data such as grid_metrics, grid_terrain and other `grid_*` functions may return huge amounts of data for large catalogs or high resolution data (typically `grid_terrain` with a resolution of 1 meter). Switching this option to `TRUE` enables storage of the data on the hard disk and returns a lightweight virtual raster mosaic.
- `memory_limit_warning` (numeric) - When applying a function to an entire catalog, an internal function tries to estimate the size of the output before running the algorithm in an attempt to prevent memory overflow. This value (in bytes) is the threshold before a warning is given. Default is 5e8 (500 Mb). Set to `Inf` to disable.

Examples

```
catalog_options(multicore = 2)
catalog_options(buffer = 40)
catalog_options()

# Reset default options
catalog_reset()
```

---

catalog_queries *Extract LiDAR data based on a set of coordinates*

Description

From a set of (x, y) coordinates corresponding to the centers of regions of interest (ROIs), for example a ground inventory, the function automatically extracts the lidar data associated with the ROIs from a catalog. The algorithm will do this even for ROIs falling on the edges of one or more tiles. The extracted lidar data can be buffered. In this case the function adds a buffer area around the ROIs, and the LAS object returned has an extra column named 'buffer' which indicates, for each point, if the point is in the buffer or from the ROI (see more in the section Buffer).

`lidR` support .lax file. You will speed-up the computation *a lot* with a spatial index.

Usage

```
catalog_queries(obj, x, y, r, r2 = NULL, buffer = 0, roinames = NULL, ...)
```

Arguments

obj	A LAScatalog object
x	vector. A set of x coordinates corresponding to the centers of the ROIs
y	vector. A set of y coordinates corresponding to the centers of the ROIs
r	numeric or vector. A radius or a set of radii of the ROIs. If only r is provided (r2 = NULL) it will extract data falling onto a disc.
r2	numeric or vector. A radius or a set of radii of plots. If r2 is provided, the selection turns into a rectangular ROI (if r = r2 it is a square).
buffer	numeric. Adds a buffer area around the ROI. See relevant sections.
roinames	vector. A set of ROI names (the plot IDs, for example) to label the returned list.
...	Any argument available in readLAS to reduce the amount of data loaded.

Value

A list of LAS objects

Buffer

If the ROIs are buffered then the LAS objects returned by the function have extra points. The LAS objects received by the user contain a special column called 'buffer', which indicates, for each point, if it comes from a buffered area or not. Points from non-buffered areas (i.e. the ROI) have a 'buffer' value of 0, while those from buffered areas have a 'buffer' value greater than 0.

For a circular ROI, points in the buffered area have a buffer value of 1. For a rectangular ROI the points in the buffer area have a buffer value of 1, 2, 3 or 4, where 1 is the bottom buffer and 2, 3 and 4 are the left, top and right buffers, respectively.

Multicore computation

The process is done using several cores. To change the settings of how a catalog is processed use catalog_options.

See Also

readLAS catalog catalog_options

Examples

```
## Not run:
# Build a LAScatalog
catalog = catalog("<Path to a folder containing a set of .las or .laz files>")

# Get coordinates from an external file
X = runif(30, 690000, 800000)
Y = runif(30, 5010000, 5020000)
R = 25

# Return a List of 30 circular LAS objects of 25 m radius
```

```
catalog %>% catalog_queries(X, Y, R)

# Return a List of 30 square LAS objects of 50x50 m
catalog %>% catalog_queries(X, Y, R, R)

# Return a List of 30 circular LAS objects of 30 m radius. 25 m being the ROI and 5 m
# being a buffered area. The LAS objects have an extra column called 'buffer' to
# differentiate the points.
catalog %>% catalog_queries(X, Y, R, buffer = 5)

# Return a List of 30 circular LAS objects of 25 m radius for which only the fields X, Y and
# Z have been loaded and Z values < 0 were removed.
catalog %>% catalog_queries(X, Y, R, XYZonly = TRUE, filter = "-drop_z_below 0")

## End(Not run)
```

`catalog_reshape` *Reshape (retile) a catalog*

Description

This function splits or merges files to reshape the original catalog files (.las or .laz) into smaller or larger files. The new files are written in a dedicated folder. The function first displays the pattern of the new tiling and then asks the user to validate the command.

Usage

```
catalog_reshape(ctg, size, path, prefix, ext = c("las", "laz"))
```

Arguments

`ctg`	A LAScatalog object
`size`	scalar. The size of the new tiles.
`path`	string. The folder where the new files should be saved.
`prefix`	character. The initial part of the name of the written files.
`ext`	character. The format of the written files. Can be ".las" or ".laz".

Value

A new catalog object

See Also

catalog

Examples

```
## Not run:
ctg = catalog("path/to/catalog")

# Create a new set of .las files 500 by 500 wide in the folder
# path/to/new/catalog/ and iteratively named Forest_1.las, Forest_2.las
# Forest_3.las, and so on.
newctg = catalog_reshape(ctg, 500, "path/to/new/catalog", "Forest_")

## End(Not run)
```

`catalog_select` *Select LAS files interactively*

Description

Select a set of LAS tiles from a LAScatalog using the mouse interactively. This function enables the user to select a set of las files from a LAScatalog by clicking on the map of the file using the mouse. The selected files will be highlighted in red on the plot after selection is complete.

Usage

```
catalog_select(x)
```

Arguments

x	A LAScatalog object

Value

A LAScatalog object

See Also

LAScatalog

Examples

```
## Not run:
project = catalog("<Path to a folder containing a set of .las files>")
selectedFiles = catalog_select(project)

## End(Not run)
```

`entropy` *Normalized Shannon diversity index*

Description

A normalized Shannon vertical complexity index

Usage

```
entropy(z, by = 1, zmax = NULL)
```

Arguments

`z`	vector of positive z coordinates
`by`	numeric. The thickness of the layers used (height bin)
`zmax`	numeric. Used to turn the function entropy to the function VCI.

Details

The Shannon diversity index is a measure for quantifying diversity and is based on the number and frequency of species present. This index, developed by Shannon and Weaver for use in information theory, was successfully transferred to the description of species diversity in biological systems (Shannon 1948). Here it is applied to quantify the diversity and the evenness of an elevational distribution of LiDAR points. It makes bins between 0 and the maximum elevation.

Value

A number between 0 and 1

References

Pretzsch, H. (2008). Description and Analysis of Stand Structures. Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-88307-4 (pages 279-280) Shannon, Claude E. (1948), "A mathematical theory of communication," Bell System Tech. Journal 27, 379-423, 623-656.

See Also

VCI

Examples

```
z = runif(10000, 0, 10)

# expected to be close to 1. The highest diversity is given for a uniform distribution
entropy(z, by = 1)

 z = runif(10000, 9, 10)
```

```
# Must be 0. The lowest diversity is given for a unique possibility
entropy(z, by = 1)

z = abs(rnorm(10000, 10, 1))

# expected to be between 0 and 1.
entropy(z, by = 1)
```

---

extent,LAS-method *Extent*

Description

Returns an Extent object of a LAS or LAScatalog object.

Usage

```
## S4 method for signature 'LAS'
extent(x, ...)

## S4 method for signature 'LAScatalog'
extent(x, ...)
```

Arguments

x	An object of the class LAS or LAScatalog
...	Unused

Value

Extent object from **raster**

See Also

raster::extent

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
las = readLAS(LASfile)
extent(las)
```

gap_fraction_profile *Gap fraction profile*

Description

Computes the gap fraction profile using the method of Bouvier et al. (see reference)

Usage

```
gap_fraction_profile(z, dz = 1, z0 = 2)
```

Arguments

z	vector of positive z coordinates
dz	numeric. The thickness of the layers used (height bin)
z0	numeric. The bottom limit of the profile

Details

The function assesses the number of laser points that actually reached the layer z+dz and those that passed through the layer [z, z+dz]. By definition the layer 0 will always return 0 because no returns pass through the ground. Therefore, the layer 0 is removed from the returned results.

Value

A data.frame containing the bin elevations (z) and the gap fraction for each bin (gf)

References

Bouvier, M., Durrieu, S., Fournier, R. a, & Renaud, J. (2015). Generalizing predictive models of forest inventory attributes using an area-based approach with airborne LiDAR data. Remote Sensing of Environment, 156, 322-334. http://doi.org/10.1016/j.rse.2014.10.004

See Also

LAD

Examples

```
z = c(rnorm(1e4, 25, 6), rgamma(1e3, 1, 8)*6, rgamma(5e2, 5,5)*10)
z = z[z<45 & z>0]

hist(z, n=50)

gapFraction = gap_fraction_profile(z)

plot(gapFraction, type="l", xlab="Elevation", ylab="Gap fraction")
```

grid_canopy *Canopy surface model*

Description

Creates a canopy surface model using a LiDAR point cloud. For each pixel the function returns the highest point found (point-to-raster). This basic method could be improved by replacing each LiDAR return with a small disk. An interpolation for empty pixels is also available.

Usage

```
grid_canopy(x, res = 2, subcircle = 0, na.fill = "none", ...,
  filter = "")
```

Arguments

x	An object of class LAS or a catalog (see section "Use with a LAScatalog")
res	numeric. The size of a grid cell in LiDAR data coordinates units. Default is 2 meters i.e. 4 square meters.
subcircle	numeric. radius of the circles. To obtain fewer empty pixels the algorithm can replace each return with a circle composed of 8 points (see details).
na.fill	character. name of the algorithm used to interpolate the data and fill the empty pixels. Can be "knnidw", "delaunay" or "kriging" (see details).
...	extra parameters for the algorithm used to interpolate the empty pixels (see details)
filter	character. Streaming filter while reading the files (see readLAS). If x is a LAScatalog the function readLAS is called internally. The user cannot manipulate the lidar data directly but can use streaming filters instead.

Details

The algorithm relies on a point-to-raster approach. For each pixel the elevation of the highest point is found and attributed to this pixel. This method implies that the resulting surface model can contain empty pixels. Those 'holes' can be filled by interpolation. Internally, the interpolation is based on the same method used in the function grid_terrain. Therefore the documentation for grid_terrain is also applicable to this function (see also examples).

The 'subcircle' tweak replaces each point with 8 points around the original one. This allows for virtual 'emulation' of the fact that a lidar point is not a point as such, but more realistically a disc. This tweak densifies the point cloud and the resulting canopy model is smoother and contains fewer 'pits' and empty pixels.

Value

Returns a data.table of class lasmetrics, which enables easier plotting and RasterLayer casting.

Use with a LAScatalog

When the parameter x is a LAScatalog the function processes the entire dataset in a continuous way using a multicore process. Parallel computing is set by default to the number of core available in the computer. The user can modify the global options using the function catalog_options.

lidR support .lax file. Computation speed will be *significantly* improved with a spatial index.

See Also

grid_metrics as.raster

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Local maximum algorithm with a resolution of 2 meters
lidar %>% grid_canopy(2) %>% plot

# Local maximum algorithm with a resolution of 1 meter replacing each
# point by a 20 cm radius circle of 8 points
lidar %>% grid_canopy(1, 0.2) %>% plot

# Local maximum algorithm with a resolution of 1 meter replacing each
# point by a 10 cm radius circle of 8 points and interpolating the empty
# pixels using the 3-nearest neighbours and an inverse-distance weighting.
grid_canopy (lidar, 1, subcircle = 0.1, na.fill = "knnidw", k = 3) %>% plot

## Not run:
grid_canopy(lidar, 1, na.fill = "knnidw", k = 3) %>% plot
grid_canopy(lidar, 1, subcircle = 0.1, na.fill = "delaunay") %>% plot

## End(Not run)
```

---

grid_density *Map the pulse or point density*

Description

Creates a pulse density map using a LiDAR point cloud. This function is an alias for `grid_metrics(obj, f, res)` with `f = length(unique(pulseID))/res^2)`

Usage

```
grid_density(x, res = 4, filter = "")
```

Arguments

x	An object of class LAS or a catalog (see section "Use with a LAScatalog")
res	numeric. The size of a grid cell in LiDAR data coordinates units. Default is 4 = 16 square meters.
filter	character. Streaming filter while reading the files (see readLAS). If x is a LAScatalog the function readLAS is called internally. The user cannot manipulate the lidar data directly but can use streaming filters instead.

Value

Returns a data.table of class lasmetrics which enables easier plotting and RasterLayer casting.

Use with a LAScatalog

When the parameter x is a LAScatalog the function processes the entire dataset in a continuous way using a multicore process. Parallel computing is set by default to the number of core available in the computer. The user can modify the global options using the function catalog_options.

lidR support .lax files. Computation speed will be *signifcantly* improved with a spatial index.

See Also

grid_metrics

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

lidar %>% grid_density(5) %>% plot
lidar %>% grid_density(10) %>% plot
```

---

grid_hexametrics *Compute metrics for hexagonal cells*

Description

Computes a series of descriptive statistics for a LiDAR dataset within hexagonal cells from a hexagonal grid pattern. This function is identical to grid_metrics or grid_metrics3d or tree_metrics but with hexagonal cells instead of classical square pixels. Please refer to grid_metrics for more information.

Usage

```
grid_hexametrics(.las, func, res = 20, splitlines = FALSE, debug = FALSE)
```

Arguments

.las	An object of class LAS
func	the function to be applied to each hexagonal cell
res	numeric. The inscribed circle radius of a hexagon. Default = 20.
splitlines	logical. If TRUE the algorithm will compute the metrics for each flightline individually. It returns the same cells several times in overlaps.
debug	logical. If you encounter a non trivial error try debug = TRUE.

Value

It returns a data.table containing the metrics for each hexagonal cell. The table has the class "lashexametrics" enabling easy plotting.

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Maximum elevation with a resolution of 4 m
grid_hexametrics(lidar, max(Z), 4) %>% plot

# Mean height with a resolution of 20 m
grid_hexametrics(lidar, mean(Z)) %>% plot

# Define your own new metrics
myMetrics = function(z, i)
{
  metrics = list(
     zwimean = sum(z*i)/sum(i), # Mean elevation weighted by intensities
     zimean  = mean(z*i),       # Mean products of z by intensity
     zsqmean = sqrt(mean(z^2))  # Quadratic mean
   )

   return(metrics)
}

metrics = grid_hexametrics(lidar, myMetrics(Z, Intensity), 10)

plot(metrics, "zwimean")
plot(metrics, "zimean")
plot(metrics, "zsqmean")
#etc.
```

---

grid_metrics *Rasterize the space and compute metrics for each cell*

Description

Computes a series of user-defined descriptive statistics for a LiDAR dataset within each pixel of a raster. Output is a data.table in which each line is a pixel (single grid cell), and each column is a metric. Works both with LAS or catalog objects. `grid_metrics` is similar to lasmetrics or grid_hexametrics except it computes metrics within each cell in a predefined grid. The grid cell coordinates are pre-determined for a given resolution.

Usage

```
grid_metrics(x, func, res = 20, start = c(0, 0), splitlines = FALSE,
  filter = "")
```

Arguments

`x`	An object of class LAS or a catalog (see section "Use with a LAScatalog")
`func`	the function to be applied to each cell (see section "Parameter func")
`res`	numeric. The size of the cells. Default 20.
`start`	vector x and y coordinates for the reference raster. Default is (0,0) (see section "Parameter start").
`splitlines`	logical. If TRUE the algorithm will compute the metrics for each flightline individually. It returns the same cells several times in overlap.
`filter`	character. Streaming filter while reading the files (see readLAS). If the input is a `LAScatalog` the function readLAS is called internally. The user cannot manipulate the lidar data directly but can use streaming filters instead.

Details

`grid_metrics` is similar to lasmetrics or grid_hexametrics except it computes metrics within each cell in a predefined grid. The grid cell coordinates are pre-determined for a given resolution, so the algorithm will always provide the same coordinates independently of the dataset. When start = (0,0) and res = 20 grid_metrics will produce the following raster centers: (10,10), (10,30), (30,10) etc.. When start = (-10, -10) and res = 20 grid_metrics will produce the following raster centers: (0,0), (0,20), (20,0) etc.. In Quebec (Canada) the reference is (-831600, 117980) in the NAD83 coordinate system.

Value

Returns a `data.table` containing the metrics for each cell. The table has the class "lasmetrics" enabling easy plotting.

Parameter `func`

The function to be applied to each cell is a classical function (see examples) that returns a labelled list of metrics. The following existing functions allows the user to compute some metrics:

- stdmetrics
- entropy

- VCI
- LAD

Users must write their own functions to create metrics. `grid_metrics` will dispatch the LiDAR data for each cell in the user's function. The user writes their function without considering grid cells, only a point cloud (see example).

Parameter `start`

The algorithm will always provide the same coordinates independently of the dataset. When start = (0,0) and res = 20 grid_metrics will produce the following raster centers: (10,10), (10,30), (30,10) etc.. When start = (-10, -10) and res = 20 grid_metrics will produce the following raster centers: (0,0), (0,20), (20,0) etc.. In Quebec (Canada) reference is (-831600, 117980) in the NAD83 coordinate system.

Use with a `LAScatalog`

When the parameter `x` is a LAScatalog the function processes the entire dataset in a continuous way using a multicore process. Parallel computing is set by default to the number of core available in the computer. The user can modify the global options using the function catalog_options.

`lidR` support .lax files. Computation speed will be *significantly* improved with a spatial index.

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Canopy surface model with 4 m^2 cells
grid_metrics(lidar, max(Z), 2) %>% plot

# Mean height with 400 m^2 cells
grid_metrics(lidar, mean(Z), 20) %>% plot

# Define your own new metrics
myMetrics = function(z, i)
{
  metrics = list(
     zwimean = sum(z*i)/sum(i), # Mean elevation weighted by intensities
     zimean  = mean(z*i),       # Mean products of z by intensity
     zsqmean = sqrt(mean(z^2))  # Quadratic mean
   )

   return(metrics)
}

metrics = grid_metrics(lidar, myMetrics(Z, Intensity))

plot(metrics)
plot(metrics, "zwimean")
plot(metrics, "zimean")
```

```
plot(metrics, "zsqmean")
#etc.
```

`grid_metrics3d` *Voxelize the space and compute metrics for each voxel*

Description

Voxelize the cloud of points and compute a series of descriptive statistics for each voxel.

Usage

```
grid_metrics3d(.las, func, res = 1, debug = FALSE)
```

Arguments

`.las`	An object of class `LAS`
`func`	the function to be apply to each voxel.
`res`	numeric. The size of the voxels
`debug`	logical. If you encounter a non trivial error try `debug = TRUE`.

Details

Voxelize creates a 3D matrix of voxels with a given resolution. It creates a voxel from the cloud of points if there is at least one point in the voxel. For each voxel the function allows computation of one or several derived metrics in the same way as the grid_metrics functions. Basically there are no predefined metrics. Users must write their own function to create metrics. Voxelize will dispatch the LiDAR data for each voxel in the user's function. The user writes their function without considering voxels, only a cloud of points (see example).

Value

It returns a `data.table` containing the metrics for each voxel. The table has the class `lasmetrics3d` enabling easier plotting.

See Also

grid_metrics

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Cloud of points is voxelized with a 1-meter resolution and in each voxel
# the number of points is computed.
grid_metrics3d(lidar, length(Z))
```

```
# Cloud of points is voxelized with a 1-meter resolution and in each voxel
# the mean scan angle of points is computed.
grid_metrics3d(lidar, mean(ScanAngle))

# Define your own metric function
myMetrics = function(i, angle, pulseID)
{
  ret = list(
     npulse  = length(unique(pulseID)),
     angle   = mean(angle),
     imean   = mean(i)
   )

   return(ret)
}

voxels = grid_metrics3d(lidar, myMetrics(Intensity, ScanAngle, pulseID))

plot(voxels, "angle")
plot(voxels, "imean")
#etc.
```

`grid_terrain` *Digital Terrain Model*

Description

Interpolates ground points and creates a rasterized digital terrain model. The interpolation can be done using 3 methods: `"knnidw"`, `"delaunay"` or `"kriging"` (see details). The algorithm uses the points classified as "ground" to compute the interpolation.
Depending on the interpolation method, the edges of the dataset can be more, or less poorly interpolated. A buffer around the region of interest is always recommended to avoid edge effects.

Usage

```
grid_terrain(x, res = 1, method, k = 10L, model = gstat::vgm(0.59, "Sph",
  874), keep_lowest = FALSE)
```

Arguments

`x`	An object of class LAS or a catalog (see section "Use with a LAScatalog")
`res`	numeric. resolution.
`method`	character. can be `"knnidw"`, `"delaunay"` or `"kriging"` (see details)
`k`	numeric. number of k-nearest neighbours when the selected method is either `"knnidw"` or `"kriging"`
`model`	a variogram model computed with vgm when the selected method is `"kriging"`. If null, it performs an ordinary or weighted least squares prediction.
`keep_lowest`	logical. This function forces the original lowest ground point of each pixel (if it exists) to be chosen instead of the interpolated values.

Details

`knnidw` Interpolation is done using a k-nearest neighbour (KNN) approach with an inverse distance weighting (IDW). This is a fast but basic method for spatial data interpolation.

`delaunay` Interpolation based on Delaunay triangulation. It makes a linear interpolation within each triangle. There are usually few cells outside the convex hull, determined by the ground points at the very edge of the dataset that cannot be interpolated with a triangulation. Extrapolation is done using knnidw.

`kriging` Interpolation is done by universal kriging using the krige function. This method combines the KNN approach with the kriging approach. For each point of interest the terrain is kriged using the k-nearest neighbour ground points. This method is more difficult to manipulate but it is also the most advanced method for interpolating spatial data.

Value

A `lasmetrics` data.table.

Use with a `LAScatalog`

When the parameter `x` is a LAScatalog the function processes the entire dataset in a continuous way using a multicore process. Parallel computing is set by default to the number of core available in the computer. A buffer is required to avoid edge artifacts. The user can modify the global options using the function catalog_options.

`lidR` support .lax files. Computation speed will be *significantly* improved with a spatial index.

See Also

grid_terrain lasnormalize vgm krige lasnormalize RasterLayer

Examples

```
LASfile <- system.file("extdata", "Topography.laz", package="lidR")
lidar = readLAS(LASfile)
plot(lidar)

dtm1 = grid_terrain(lidar, method = "knnidw", k = 6)
dtm2 = grid_terrain(lidar, method = "delaunay")
dtm3 = grid_terrain(lidar, method = "kriging", k = 10)

## Not run:
plot(dtm1)
plot(dtm2)
plot(dtm3)
plot3d(dtm1)
plot3d(dtm2)
plot3d(dtm3)

## End(Not run)
```

grid_tincanopy *Canopy height model based on a triangulation.*

Description

Canopy height model based on a triangulation of first returns. Depending on the inputs this function computes a simple Delaunay triangulation of the first returns with a linear interpolation within each triangle. This function also enables the use of the pit-free algorithm developed by Khosravipour et al. (2014), which is based on the computation of a set of classical triangulations at different heights (see reference).

Usage

```
grid_tincanopy(x, res = 0.5, thresholds = c(0, 2, 5, 10, 15),
  max_edge = c(0, 1), subcircle = 0, filter = "-keep_first")
```

Arguments

x	A LAS object
res	numeric. Resolution of the canopy height model.
thresholds	numeric. Set of height thresholds. If `thresholds = 0` the algorithm is a strict rasterization of the triangulation of the first returns. However, if an array is passed to the function it becomes the Khosravipour et al. pit-free algorithm.
max_edge	numeric. Maximum edge-length of a triangle in the Delaunay triangulation. If a triangle has an edge greater than this value it will be removed. It is used to drive the pit-free algorithm (see reference) and to trim dummy interpolation on non-convex areas. The first number is the value for the classical triangulation (threshold = 0), the second number is the value for the pit-free algorithm for (thresholds > 0). If `max_edge = 0` no trimming will be done.
subcircle	numeric. Radius of the circles. To obtain fewer pits the algorithm can replace each return with a circle composed of 8 points before computing the triangulation (see also grid_canopy).
filter	character. Streaming filter while reading the files (see readLAS). If the input is a `LAScatalog` the function readLAS is called internally. The user cannot manipulate the lidar data directly but can use streaming filters instead.

Value

Returns a `data.table` of class `lasmetrics`, which enables easier plotting and RasterLayer casting.

Use with a `LAScatalog`

When the parameter `x` is a LAScatalog the function processes the entire dataset in a continuous way using a multicore process. Parallel computing is set by default to the number of core available in the computer. A buffer is required. The user can modify the global options using the function catalog_options.

lidR support .lax files. Computation speed will be *significantly* improved with a spatial index.

References

Khosravipour, A., Skidmore, A. K., Isenburg, M., Wang, T., & Hussin, Y. A. (2014). Generating pit-free canopy height models from airborne lidar. Photogrammetric Engineering & Remote Sensing, 80(9), 863-872.

Examples

```
LASfile = system.file("extdata", "Tree.laz", package="lidR")
las = readLAS(LASfile, Classification = FALSE, Intensity = FALSE, filter = "-drop_z_below 0")

# Basic triangulation and rasterization
chm1 = grid_tincanopy(las, thresholds = 0, max_edge = 0)

# Khosravipour et al. pitfree algorithm
chm2 = grid_tincanopy(las, thresholds = c(0,2,5,10,15), max_edge = c(0, 1.5))

plot(chm1)
plot(chm2)
```

LAD *Leaf area density*

Description

Computes a leaf area density profile based on the method of Bouvier et al. (see reference)

Usage

```
LAD(z, dz = 1, k = 0.5, z0 = 2)
```

Arguments

z	vector of positive z coordinates
dz	numeric. The thickness of the layers used (height bin)
k	numeric. is the extinction coefficient
z0	numeric. The bottom limit of the profile

Details

The function assesses the number of laser points that actually reached the layer z+dz and those that passed through the layer [z, z+dz] (see gap_fraction_profile). Then it computes the log of this quantity and divides it by the extinction coefficient k as described in Bouvier et al. By definition the layer 0 will always return infinity because no returns pass through the ground. Therefore, the layer 0 is removed from the returned results.

Value

A data.frame containing the bin elevations (z) and leaf area density for each bin (lad)

References

Bouvier, M., Durrieu, S., Fournier, R. a, & Renaud, J. (2015). Generalizing predictive models of forest inventory attributes using an area-based approach with airborne LiDAR data. Remote Sensing of Environment, 156, 322-334. http://doi.org/10.1016/j.rse.2014.10.004

See Also

gap_fraction_profile

Examples

```
z = c(rnorm(1e4, 25, 6), rgamma(1e3, 1, 8)*6, rgamma(5e2, 5,5)*10)
z = z[z<45 & z>0]

lad = LAD(z)

plot(lad, type="l", xlab="Elevation", ylab="Leaf area density")
```

LAS *Create a* LAS *object*

Description

Create a LAS object

Usage

```
LAS(data, header = list(), check = TRUE)
```

Arguments

data	a data.table containing the LiDAR data.
header	a list containing the data from the header of a las file.
check	logical. consistency tests while building the object.

Value

An object of class LAS

See Also

Class LAS

LAS-class *An S4 class to represent the data read in a .las or .laz file*

Description

A LAS object is the representation of a las/laz files.

Details

A LAS object contains a data.table in the slot @data with the data read from a las/laz file, a LASheader in the slot @header (see the ASPRS documentation for the LAS file format for more information) and a CRS (coordinates reference system) in the slot @crs. In the slot @data The fields read from the las/laz file are named:

- X Y Z
- Intensity
- ReturnNumber
- NumberOfReturns
- ScanDirectionFlag
- EdgeOfFlightline
- Classification
- ScanAngle
- UserData
- PointSourceID

3 other fields can be computed is desired: slot @data:

- pulseID: a unique identifying number for each pulse so the beam origin of each point is known (see laspulse)
- flightlineID: a unique identifying number for the flightline so the flightline origin of each point is known (see lasflightline)
- color: the hexadecimal name of the color of the point if R, G and B fields exist (see lascolor)

Slots

data data.table. a table representing the LAS data

header list. A list of information contained in the las file header.

crs A CRS object.

See Also

LAS readLAS

LAScatalog-class *An S4 class to represent a set of a .las or .laz files*

Description

A LAScatalog object is a representation of a set of las/laz files

Slots

`data` data.table. A table representing the header of each file.

`crs` A CRS object.

See Also

LAS LAS readLAS

lasclassify *Classify LiDAR points from source data*

Description

Classify LAS points based on geographic data from external sources. It adds an attribute to each point based on a value found in the external data. External sources can be an ESRI Shapefile (SpatialPolygonsDataFrame) or a Raster (RasterLayer).

Usage

```
lasclassify(.las, source, field = NULL)
```

Arguments

`.las`	An object of the class `LAS`
`source`	An object of class `SpatialPolygonsDataFrame` or `RasterLayer`
`field`	characters. The name of a field in the table of attributes of the shapefile or the name of the new column in the LAS object (see details)

Details

The function recognizes several type of sources:

- `SpatialPolygonsDataFrame`: Polygons can be simple, one-part shapes, multi-part polygons, or polygons with holes. It checks if the LiDAR points are in polygons given in the shapefile. If the parameter `field` is the name of a field in the table of attributes of the shapefile it assigns to the points the values of that field. Otherwise it classifies the points as boolean. TRUE if the points are in a polygon, FALSE otherwise. This function allows filtering of lakes, for example.

- `RasterLayer`: It attributes to each point the value found in each pixel of the RasterLayer. Use the parameter `field` to force the name of the new column added in the LAS object. This function is used internally to normalize the lidar dataset and is exported because some users may find it useful (for example to colorize the point cloud using a georeferenced RGB image.

More examples available on lidR wiki.

Value

Nothing. The new field is added by reference to the original data.

See Also

`readOGR` `SpatialPolygonsDataFrame`

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
shapefile_dir <- system.file("extdata", package = "lidR")

lidar = readLAS(LASfile)
lakes = rgdal::readOGR(shapefile_dir, "lake_polygons_UTM17")

# The field "inlake" does not exist in the shapefile. Points are classified as TRUE if in a polygon
lasclassify(lidar, lakes, "inlakes") # New column 'inlakes' is added.
forest = lasfilter(lidar, inlakes == FALSE)
plot(lidar)
plot(forest)

# The field "LAKENAME_1" exists in the shapefile.
# Points are classified with the values of the polygons
lasclassify(lidar, lakes, "LAKENAME_1") # New column 'LAKENAME_1' is added.
```

---

lasclip *Clip LiDAR points*

Description

Clip LiDAR points within a given geometry and convenient wrappers most common geometries

Usage

```
lasclip(.las, geometry, coord, inside = TRUE)

lasclipRectangle(.las, xleft, ybottom, xright, ytop, inside = TRUE)

lasclipPolygon(.las, x, y, inside = TRUE)

lasclipCircle(.las, xcenter, ycenter, radius, inside = TRUE)
```

Arguments

.las	An object of class LAS
geometry	characters. name of a geometry. Can be "circle", "rectangle", "polygon", "cuboid" or "sphere"
coord	matrix or data.frame. The coordinates of the minimum points required to fully describe the geometry. For circle a 1-by-3 matrix, for rectangle a 2-by-2 matrix, for polygon n-by-2 matrix, for cuboid 2-by-3 matrix and for sphere a 1-by-4 matrix
inside	logical. Keep data inside or outside the shape
xleft	scalar. of left x position.
ybottom	scalar. of bottom y position.
xright	scalar. of right x position.
ytop	scalar. of top y position.
x	numerical array. x-coordinates of polygon
y	numerical array. y-coordinates of polygon
xcenter	scalar. x disc center
ycenter	scalar. y disc center
radius	scalar. Disc radius

Value

An object of class LAS

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

subset = lidar %>% lasclipRectangle(xleft  = 684850, ybottom = 5017850,
                                    xright = 684900, ytop    = 5017900)
plot(subset)

msphere = matrix(c(684850, 5017850, 10, 10), ncol = 4)
subset = lidar %>% lasclip("sphere", msphere)
plot(subset)

mrect = matrix(c(684850, 684900, 5017850, 5017900), ncol = 2)
subset = lidar %>% lasclip("rectangle", mrect)
```

lascolor *Compute the color from RGB fields*

Description

Compute the hexadecimal name of a color from RGB fields. A column 'color' is added in the slot @data

Usage

```
lascolor(.las, nbits = 16)
```

Arguments

.las	A LAS object
nbits	the number of bits used to store the R G and B field. Default is 16 bits i.e. each channel ranged between 0 and 65535 as defined in LAS specifications. But sometime this rule is not respected. You can force the default behavior. A 8 bit color ranged between 0 and 255.

Value

Return nothing. The original object is modified in place by reference.

lasdecimate *Thin LiDAR data*

Description

Thin LIDAR data randomly removes a given proportion of pulses to reach specific pulse densities

Usage

```
lasdecimate(.las, density, homogenize = TRUE, res = 5)
```

Arguments

.las	An object of the class LAS
density	numeric. The expected density
homogenize	logical. If TRUE, the algorithm tries to homogenize the pulse density to provide a uniform dataset. If FALSE the algorithm will reach the pulse density over the whole area.
res	numeric. Cell size to compute the pulse density.

Details

lasdecimate is designed to produce output datasets that have uniform pulse densities throughout the coverage area. For each cell, the proportion of pulses that will be retained is computed using the actual pulse density and the desired pulse density. If the required pulse density is greater than the actual pulse density it returns an unchanged set of points (it cannot increase the pulse density). If homogenize = FALSE is selected, it randomly removes pulses to reach the required pulse density over the whole area (see area). The cell size must be large enough to compute a coherent local pulse density i.e., in a 2 pulse/m^2 dataset, 25 square meters would be feasible; however, an extent too small to thin (e.g. <1 square meter) would not be feasible because pulse density does not have meaning at this scale.

Value

It returns a LAS object.

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile, select = "xyztP")

# By default the method is homogenize = TRUE
thinned = lidar %>% lasdecimate(1, res = 5)
lidar   %>% grid_density %>% plot
thinned %>% grid_density %>% plot

# Method homogenize = FALSE enables a global pulse density to be reached
thinned = lidar %>% lasdecimate(1, homogenize = FALSE)
thinned %>% summary
thinned %>% grid_density %>% plot
```

---

lasfilter *Return points with matching conditions*

Description

Return points with matching conditions.

Usage

```
lasfilter(.las, ...)
```

Arguments

.las	An object of class LAS
...	Logical predicates. Multiple conditions are combined with & or ,

Value

An object of class LAS

See Also

Other lasfilters: lasfilters

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Select the first returns classified as ground
firstground = lidar %>% lasfilter(Classification == 1 & ReturnNumber == 1)

# Multiple arguments are equivalent to &
firstground = lidar %>% lasfilter(Classification == 1, ReturnNumber == 1)

# Multiple criteria
first_or_ground = lidar %>% lasfilter(Classification == 1 | ReturnNumber == 1)
```

---

lasfilters *Predefined filters*

Description

Select only some returns

Usage

```
lasfilterfirst(.las)

lasfilterfirstlast(.las)

lasfilterfirstofmany(.las)

lasfilterground(.las)

lasfilterlast(.las)

lasfilternth(.las, n)

lasfiltersingle(.las)

lasfilterfirstofmany(.las)
```

Arguments

.las	An object of class LAS
n	the position in the return sequence

Details

- lasfilterfirst Select only the first returns.
- lasfilterfirstlast Select only the first and last returns.
- lasfilterground Select only the returns classified as ground according to LAS specification v1.3.
- lasfilterlast Select only the last returns i.e. the last returns and the single returns.
- lasfilternth Select the returns from their position in the return sequence.
- lasfilterfirstofmany Select only the first returns from pulses which returned multiple points.
- lasfiltersingle Select only the returns which return only one point.

Value

An object of class LAS

See Also

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Other lasfilters: lasfilter

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

firstReturns  = lidar %>% lasfilterfirst
groundReturns = lidar %>% lasfilterground
```

lasflightline *Retrieve individual flightlines*

Description

Retrieve each individual flightline by attributing a number to each point. The function depends on the GPS time to retrieve each individual flightline. In a continuous dataset, once points are ordered by GPS time, the time between two consecutive points does not exceed a few milliseconds. If the time between two consecutive points is too long it means that the second point is from a different flightline. The default threshold is 30 seconds. A column 'flightlineID' is added in the slot @data

Usage

```
lasflightline(.las, dt = 30)
```

Arguments

.las	A LAS object
dt	numeric. The threshold time lag used to retrieve flightlines

Value

Return nothing. The original object is modified in place by reference.

lasground *Classify points as ground or not ground*

Description

Implements a Progressive Morphological Filter for segmentation of ground points. The function updates the field Classification of the input LAS object. The points classified as 'ground' are assigned a value of 2 according to las specifications (See the ASPRS documentation for the LAS file format). This function is an implementation of the Zhang et al. (2003) algorithm (see reference)

Usage

```
lasground(.las, MaxWinSize = 20, Slope = 1, InitDist = 0.5, MaxDist = 3,
  CellSize = 1, ...)
```

Arguments

.las	a LAS object
MaxWinSize	numeric. Maximum window size to be used in filtering ground returns (see references)
Slope	numeric. Slope value to be used in computing the height thresholds (see references)
InitDist	numeric. Initial height above the parameterized ground surface to be considered a ground return (see references)
MaxDist	numeric. Maximum height above the parameterized ground surface to be considered a ground return (see references)
CellSize	numeric. Cell size
...	Any additional specific parameters to be passed to the progressive morphological filter. These include: - exponential logical. Default is TRUE. - base numeric. Default is 2

Value

Nothing. The original LAS object is updated by reference. In the 'Classification' column a value of 2 denotes ground according to LAS specifications.

References

Zhang, K., Chen, S. C., Whitman, D., Shyu, M. L., Yan, J., & Zhang, C. (2003). A progressive morphological filter for removing nonground measurements from airborne LIDAR data. IEEE Transactions on Geoscience and Remote Sensing, 41(4 PART I), 872–882. http:#doi.org/10.1109/TGRS.2003.810682

Examples

```
LASfile <- system.file("extdata", "Topography.laz", package="lidR")
las = readLAS(LASfile, XYZonly = TRUE)

lasground(las, MaxWinSize = 40, Slope = 1, MaxDist = 5, InitDist = 0.01, CellSize = 7)

plot(las, color = "Classification")
```

LASheader *Create a* LASheader *object*

Description

Create a LASheader object

Usage

```
LASheader(data = list())
```

Arguments

data	a list containing the data from the header of a las file.

Value

An object of class LASheader

See Also

Class LASheader Class LAS

LASheader-class *An S4 class to represent the header read in a .las or .laz file*

Description

A LASheader object contains a list in the slot @PHB with the data read from the Public Header Block and list in the slot @VLR with the data read from the Variable Lenght Records

Slots

PHB list. Represents the Public Header Block

VLR list. Represents the Variable Length Records

See Also

LAS readLAS

---

lasmetrics *Compute metrics for a cloud of points*

Description

lasmetrics computes a series of user-defined descriptive statistics for a LiDAR dataset. See grid_metrics to compute metrics on a grid. Basically there are no predefined metrics. Users must write their own functions to create metrics (see example). The following existing functions can serve as a guide to help users compute their own metrics:

- stdmetrics
- entropy
- VCI
- LAD

Usage

```
lasmetrics(obj, func)
```

Arguments

obj	An object of class LAS
func	The function to be applied to a cloud of points. Function must return a list (see example)

Value

It returns a list containing the metrics

See Also

grid_metrics stdmetrics entropy VCI LAD

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

lasmetrics(lidar, max(Z))
lasmetrics(lidar, mean(ScanAngle))

# Define your own new metrics
myMetrics = function(z, i)
{
  metrics = list(
     zwimean = sum(z*i)/sum(i), # Mean elevation weighted by intensities
     zimean  = mean(z*i),       # Mean products of z by intensity
     zsqmean = sqrt(mean(z^2))  # Quadratic mean
   )

   return(metrics)
}

metrics = lasmetrics(lidar, myMetrics(Z, Intensity))

# Predefined metrics
lasmetrics(lidar, .stdmetrics)
```

---

lasnormalize *Subtract digital terrain model*

Description

Subtract digital terrain model (DTM) from LiDAR data to create a dataset normalized with the ground at 0. The DTM can originate from several sources e.g. from an external file or computed by the user. It can also be computed on the fly. In this case the algorithm does not use rasterized data and each point is interpolated. There is no inaccuracy due to the discretization of the terrain the resolution of the terrain is virtually infinite (but it is slower).
Depending on the interpolation method, the edges of the dataset can be more or less poorly interpolated. A buffer around the region of interest is always recommended to avoid edge effects.

Usage

```
lasnormalize(.las, dtm = NULL, method, k = 10L, model = gstat::vgm(0.59,
  "Sph", 874), copy = FALSE)
```

Arguments

.las	a LAS object
dtm	a RasterLayer or a lasmetrics object computed with grid_terrain.
method	character. Used if dtm = NULL. Can be "knnidw", "delaunay" or "kriging" (see grid_terrain for more details)
k	numeric. Used if dtm = NULL. Number of k-nearest neighbours when the selected method is either "knnidw" or "kriging"
model	Used if dtm = NULL. A variogram model computed with vgm when the selected method is "kriging". If NULL it performs an ordinary or weighted least squares prediction.
copy	By default the point cloud is updated in place by reference. User can force the function to return a new point cloud. Set TRUE to get a compatibility with versions < 1.3.0

Details

knnidw Interpolation is done using a k-nearest neighbour (KNN) approach with an inverse distance weighting (IDW). This is a fast but basic method for spatial data interpolation.

delaunay Interpolation based on Delaunay triangulation. It makes a linear interpolation within each triangle. There are usually few points outside the convex hull, determined by the ground points at the very edge of the dataset which cannot be interpolated with a triangulation. Extrapolation is done using knnidw.

kriging Interpolation is done by universal kriging using the krige function. This method combines the KNN approach with the kriging approach. For each point of interest it kriges the terrain using the k-nearest neighbour ground points. This method is more difficult to manipulate but it is also the most advanced method for interpolating spatial data.

Value

The function returns NULL. The LAS object is updated by reference. Z is now the normalized elevation, A new column 'Zref' records the former elevations values. This is a way to save memory avoiding copies of the point cloud. But if copy = TRUE, a new LAS object is returned and the original one is not modified.

See Also

raster grid_terrain

Examples

```
LASfile <- system.file("extdata", "Topography.laz", package="lidR")
las = readLAS(LASfile)

plot(las)

# --- First option: compute a raster DTM with grid_terrain ---
# (or read it from a file)

dtm = grid_terrain(las, method = "kriging", k = 10L)
lasnormalize(las, dtm)

plot(dtm)
plot(las)

# --- Second option: interpolate each point (no discretization) ---
las = readLAS(LASfile)

lasnormalize(las, method = "kriging", k = 10L, model = gstat::vgm(0.59, "Sph", 874))
plot(las)
```

---

laspulse *Retrieve individual pulses*

Description

Retrieve each individual pulse by attributing a number to each point. The function depends on the GPS time to retrieve each individual beam. A column 'pulseID' is added in the slot @data

Usage

```
laspulse(.las)
```

Arguments

.las	A LAS object

Value

Return nothing. The original object is modified in place by reference.

lasroi *Select a region of interest interactively*

Description

Select a rectangular region of interest interactively using the mouse

Usage

```
lasroi(.las, ...)
```

Arguments

.las	An object of class LAS
...	Optional parameters for the plot function

Details

lasroi enables the user to select a region of interest (ROI) by drawing a rectangle with the mouse

Value

An object of class LAS

Examples

```
## Not run:
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

subset = lasroi(lidar)

## End(Not run)
```

lasscanline *Retrieve individual scanline*

Description

Retrieve each individual scanline by attributing to each point a number. When data are sampled according to a saw-tooth pattern (oscillating mirror) a scanline is one line, or row of data. The function relies on the GPS field time to order the data. Then, the 'ScanDirectionFlag' field (when available) is used to retrieve each scanline A column 'scanline' is added in the slot @data

Usage

```
lasscanline(.las)
```

Arguments

.las	A LAS object

Value

Return nothing. The original object is modified in place by reference.

---

lastrees *Individual tree segmentation*

Description

Individual tree segmentation with several possible algorithms (see details). The function attributes to each point of the point cloud a number identifying the detected tree the point comes from (treeID column). By default the classification is done at the point cloud level. However, with some algorithms it is possible to return a raster image of the classification. There are currently 3 algorithms implemented. See relevant sections.

Usage

```
lastrees(.las, algorithm, image = NULL, ..., extra = FALSE)
```

Arguments

.las	An object of the class LAS
algorithm	character. The name of an algorithm. Can be "dalponte2016", "watershed" or "li2012" (see sections relevant to each algorithm).
image	RasterLayer. Image of the canopy if the algorithm works on a canopy surface model. But some algorithms work on the raw point cloud (see relevant sections). You can compute it with grid_canopy or grid_tincanopy or read it from external file.
...	parameters for the algorithms. These depend on the algorithm used (see details about the algorithms)
extra	logical. By default the function works at the point cloud level and returns nothing. If extra = TRUE the function can return a RasterLayer or a list of 2 RasterLayers with the positions of the local maxima and a map of the crowns, depending on the algorithm used.

Value

Nothing, the point cloud is updated by reference. If extra = TRUE, "dalponte2012" returns two RasterLayers, "watershed" returns one RasterLayer and "li2012" does not support the extra parameter.

Dalponte 2016

This is the algorithm developed by Dalponte and Coomes (see references). This algorithm exists in the package **itcSegment**. This version is identical to the original but with superfluous code removed and rewritten in C++. Consequently it is 6 times faster. Note that this algorithm strictly performs a segmentation while the original method as implemented in `itcSegment` and described in the manuscript also performs a pre- and post-process when these tasks are expected to be done by the user. The names of the parameters are the same as those in Dalponte's `itcSegment` package. Dalponte's algorithm is a canopy surface model-based method. An image of the canopy is expected.

`searchWinSize` Size (in pixels) of the moving window used to the detect the local maxima. It should be an odd number larger than 3. Default 3

`TRESHSeed` Growing threshold 1. It should be between 0 and 1. Default 0.45

`TRESHCrown` Growing threshold 2. It should be between 0 and 1. Default 0.55

`DIST` Maximum value of the crown diameter of a detected tree (in meters). Default 10

`th` Digital number value below which a pixel cannot be a local maxima. Default 2

Watershed

This method relies on the watershed segmentation method. It is based on the bioconductor package **EBImage**. You need to install this package to run this method (see its github page). The Watershed algorithm is a canopy surface model-based method. An image of the canopy is expected.

`th` Numeric. Number value below which a pixel cannot be a crown. Default 2

`tolerance` Numeric. see ?EBImage::watershed

`ext` Numeric. see ?EBImage::watershed

Li 2012

This method is an implementation of the Li et al. (see references) algorithm made by **lidR** author. It may have some differences compared with the original method due to potential mis-interpretation of the Li et al. manuscript. This method works at the point cloud level. An image of the canopy is *not* expected.

`dt1` Numeric. Threshold number 1. See reference page 79. Default is 1.5

`dt2` Numeric. Threshold number 2. See reference page 79. Default is 2

`R` Numeric. Maximum radius of a crown. Any value greater than a crown is good because this parameter does not affect the result. However, it greatly affects the computation speed. The lower the value, the faster the method. Default is 10.

The current implementation is known to be slow. Improvements are possible in future package versions.

References

Dalponte, M. and Coomes, D. A. (2016), Tree-centric mapping of forest carbon density from airborne laser scanning and hyperspectral data. Methods Ecol Evol, 7: 1236–1245. doi:10.1111/2041-210X.12575

Li, W., Guo, Q., Jakubowski, M. K., & Kelly, M. (2012). A new method for segmenting individual trees from the lidar point cloud. Photogrammetric Engineering & Remote Sensing, 78(1), 75-84.

Examples

```
LASfile <- system.file("extdata", "Tree.laz", package="lidR")
las = readLAS(LASfile, XYZonly = TRUE, filter = "-drop_z_below 0")

# compute a canopy image
chm = grid_canopy(las, res = 0.5, subcircle = 0.1, na.fill = "knnidw", k = 4)
chm = as.raster(chm)

# smoothing post-process (e.g. 2x mean)
kernel = matrix(1,3,3)
chm = raster::focal(chm, w = kernel, fun = mean)
chm = raster::focal(chm, w = kernel, fun = mean)
raster::plot(chm, col = height.colors(50)) # check the image

# segmentation
lastrees(las, "dalponte2016", chm, th = 5)

# plot points that actually are trees
trees = lasfilter(las, !is.na(treeID))
plot(trees, color = "treeID", colorPalette = random.colors(100))
```

---

lidR-deprecated *Deprecated function(s) in the lidR package*

Description

These functions are provided for compatibility with older version of the lidR package. They may eventually be completely removed.

Usage

```
cloud_metrics(...)

lasarea(...)
```

Arguments

...	Parameters to be passed to the modern version of the function

Details

- `lasmetrics` replace old `cloud_metrics`
- `area` replace old `lasarea`

lidrpalettes *Palettes*

Description

Create a vector of n contiguous (or not) colors

Usage

```
height.colors(n)

forest.colors(n)

random.colors(n)

pastel.colors(n)
```

Arguments

n	The number of colors (> 1) to be in the palette

See Also

grDevices::Palettes

lidr_options *Options Settings for the lidR package*

Description

Allows the user to set and examine a variety of global options that affect the way in which lidR computes and displays its results.

Usage

```
lidr_options(...)

lidr_reset()
```

Arguments

...	Option names to retrieve option values or [key]=[value] pairs to set options.

Supported options

The following options are supported:

- verbose (logical) Make the package "talkative".
- progress (logical) Display progress bar when available.
- debug (logical) Switch the package to debug mode when available.

See Also

catalog_options

Examples

```
lidr_options(verbose = TRUE)
lidr_options(progress = TRUE)
lidr_options()

# Reset default options
lidr_reset()
```

---

plot.LAS *Plot LiDAR data*

Description

This functions implements a 3D plot method for LAS objects

Usage

```
## S3 method for class 'LAS'
plot(x, y, color = "Z", colorPalette = height.colors(50),
  bg = "black", trim = 1, ...)
```

Arguments

x	An object of the class LAS
y	Unused (inherited from R base)
color	characters. The field used to color the points. Default is Z coordinates. Or a vector of colors.
colorPalette	characters. a list of colors such as that generated by heat.colors, topo.colors, terrain.colors or similar functions. Default is height.colors(50) provided by the package lidR
bg	The color for the background. Default is black.

trim	numeric. Enables trimming of values when outliers break the color palette range. Default is 1, meaning that the whole range of values is used for the color palette. 0.9 means that 10% of the highest values are not used to define the color palette. In this case values higher than the 90th percentile are set to the highest color. They are not removed.
...	Supplementary parameters for points3d

See Also

points3d height.colors forest.colors heat.colors colorRampPalette Class LAS

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

plot(lidar)

# Outliers of intensity breaks the color range. Use the trim parameter.
plot(lidar, color = "Intensity", colorPalette = heat.colors(50))
plot(lidar, color = "Intensity", colorPalette = heat.colors(50), trim = 0.99)
```

---

plot.LAScatalog *Plot a LAScatalog object*

Description

This functions implements a plot method for LAScatalog objects

Usage

```
## S3 method for class 'LAScatalog'
plot(x, y = TRUE, ...)
```

Arguments

x	A LAScatalog object
y	logical. Disable interactive display
...	inherited from base plot

Examples

```
## Not run:
ctg = catalog("<Path to a folder containing a set of .las files>")
plot(catalog)

# Exemple with a single file
proj4   <- "+proj=utm +zone=17"
```

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")

ctg = catalog(LASfile)
plot(ctg, proj4)

## End(Not run)
```

`plot.lashexametrics` *Plot an object of class lashexametrics in 2D*

Description

This functions implements a plot method for `lashexametrics` data.table

Usage

```
## S3 method for class 'lashexametrics'
plot(x, z = NULL, colorPalette = height.colors(50),
  ...)
```

Arguments

`x`	A data.table of class `lashexametrics`.
`z`	character. The field to plot. If NULL, autodetect.
`colorPalette`	characters. A list of colors such as that generated by heat.colors, topo.colors, terrain.colors or similar functions. Default is `height.colors(50)` provided by the package `lidR`
`...`	inherited from base

See Also

grid_metrics heat.colors

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

# Max Z within
grid_hexametrics(lidar, max(Z), 5) %>% plot
```

`plot.lasmetrics` *Plot an object of class lasmetrics in 2D*

Description

This functions implements a plot method for a `lasmetrics` data.frame

Usage

```
## S3 method for class 'lasmetrics'
plot(x, z = NULL, colorPalette = height.colors(50),
  trim = Inf, ...)
```

Arguments

`x`	A data.frame or data.table of class `lasmetrics`.
`z`	character. The field to plot. If NULL, autodetect.
`colorPalette`	characters. a list of colors such as that generated by heat.colors, topo.colors, terrain.colors or similar functions. Default is `height.colors(50)` provided by the package `lidR`
`trim`	numeric. Enables trimming of values when outliers break the color palette range. Default is +Inf (no trimming), meaning that the whole range of values is used for the color palette. 10 means that the values greater than 10 are all assigned to the highest color.
`...`	Supplementary parameters for plot

Details

The ... param provides additional arguments to plot.

See Also

grid_metrics grid_canopy height.colors forest.colors heat.colors colorRampPalette

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
las = readLAS(LASfile)

# Canopy surface model with 4 m^2 cells
grid_canopy(las) %>% plot

# Mean height with 400 m^2 cells
grid_metrics(las, mean(Z)) %>% plot

# With multiple metrics
metrics = grid_metrics(las, .stdmetrics_z)
```

```
plot(metrics)
plot(metrics, "zmean")
plot(metrics, "zmax")
```

`plot.lasmetrics3d` *Plot voxelized LiDAR data*

Description

This function implements a 3D plot method for 'lasmetrics3d' objects

Usage

```
## S3 method for class 'lasmetrics3d'
plot(x, y, color = "Z",
  colorPalette = height.colors(50), bg = "black", trim = 1, ...)
```

Arguments

`x`	An object of the class `'lasmetrics3d'`
`y`	Unused (inherited from R base)
`color`	characters. The field used to color the points. Default is Z coordinates. Or a vector of colors.
`colorPalette`	characters. A color palette name. Default is `height.colors` provided by the package lidR
`bg`	The color for the background. Default is black.
`trim`	numeric. Enables trimming of values when outliers break the color palette range. Default is 1 meaning that the whole range of the values is used for the color palette. 0.9 means that 10 In this case the values higher than the 90th percentile are set to the highest color. They are not removed
`...`	Supplementary parameters for points3d if display method is "points"

See Also

grid_metrics3d points3d height.colors forest.colors heat.colors colorRampPalette

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile)

voxels = grid_metrics3d(lidar, list(Imean = mean(Intensity)))
plot(voxels, color = "Imean", colorPalette = heat.colors(50), trim=0.99)
```

plot3d *Plot a wireframe of a* RasterLayer *or a* lasmetrics *object*

Description

Plot a wireframe of a RasterLayer or a lasmetrics object

Usage

```
plot3d(x, y, add = FALSE, bg = "black", ...)
```

Arguments

x	An object of the class RasterLayer or lasmetrics
y	Unused (inherited from R base)
add	logical. if TRUE, add to current 3D plot.
bg	The color for the background. Default is black.
...	Supplementary parameters for surface3d

readLAS *Read .las or .laz files*

Description

Reads .las or .laz files in format 1 to 3 according to LAS specification and returns an object of class LAS. If several files are given the returned LAS object is considered as one LAS file. The information retained in the header will be read from the first file in the list. The optional parameters enable the user to save a substantial amount of memory by choosing to load only the fields or points required. These internal options are much more memory efficient than any other R code.

Usage

```
readLAS(files, select = "xyztinrcaRGBP", filter = "", ...)
```

Arguments

files	array of characters or a LAScatalog object
select	character. select only columns of interest to save memory (see details)
filter	character. streaming filters - filter data while reading the file (see details)
...	compatibility with former arguments from lidR (<= 1.2.1)

Details

The 'select' argument specifies which data will actually be loaded. For example, 'xyzia' means that the x, y, and z coordinates, the intensity and the scan angle will be loaded. The supported entries are t - gpstime, a - scan angle, i - intensity, n - number of returns, r - return number, c - classification, u - user data, p - point source ID, e - edge of flight line flag, d - direction of scan flag, R - red channel of RGB color, G - green channel of RGB color, B - blue channel of RGB color, * - is the wildcard and enables everything from the LAS file.
x, y, z are implicit and always loaded. 'xyzia' is equivalent to 'ia' and an empty string is equivalent to 'xyz' but `select = "xyz"` is more readable and explicit than `select = ""`.

Three extra metrics can be computed on the fly with the following flags: P - pulse id, F - flightline id and C - color string (see Class LAS. The symbol + is a shortcut for 'PFC'.

The 'filter' argument allows filtering of the point cloud while reading files. This is much more efficient than lasfilter in many ways. If the desired filters are known before reading the file, the internal filters should always be preferred. The available filters are those from `LASlib` and can be found by running the following command: rlas:::lasfilterusage()

Value

A LAS object

See Also

Class LAS LAScatalog

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
las = readLAS(LASfile)
las = readLAS(LASfile, select = "xyz")
las = readLAS(LASfile, select = "xyzi", filter = "-keep_first")
las = readLAS(LASfile, select = "xyziar", filter = "-keep_first -drop_z_below 0")
las = readLAS(LASfile, select = "*+")
```

---

rumple_index *Rumple index of roughness*

Description

Computes the roughness of a surface as the ratio between its area and its projected area on the ground. If the input is a gridded object (lasmetric or raster) the function computes the surfaces using Jenness's algorithm (see references). If the input is a point cloud the function uses a Delaunay triangulation of the points and computes the area of each triangle.

Usage

```
rumple_index(x, y = NULL, z = NULL, ...)
```

Arguments

x	A 'RasterLayer' or a 'lasmetrics' object, or a vector of x point coordinates.
y	numeric. If x is a vector of coordinates: the associated y coordinates.
z	numeric. If x is a vector of coordinates: the associated z coordinates.
...	unused

Value

numeric. The computed Rumple index.

References

Jenness, J. S. (2004). Calculating landscape surface area from digital elevation models. Wildlife Society Bulletin, 32(3), 829–839.

Examples

```
x = runif(20, 0, 100)
y = runif(20, 0, 100)

# Perfectly flat surface, rumple_index = 1
z = rep(10, 20)
rumple_index(x, y, z)

# Rough surface, rumple_index > 1
z = runif(20, 0, 10)
rumple_index(x, y, z)

# Rougher surface, rumple_index increases
z = runif(20, 0, 50)
rumple_index(x, y, z)

# Measure of roughness is scale-dependent
rumple_index(x, y, z)
rumple_index(x/10, y/10, z)

# Use with a canopy height model
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
las = readLAS(LASfile)
chm = las %>% grid_canopy
rumple_index(chm)
```

set.colors *Automatic colorization*

Description

Attribute a color to each element of a vector

Usage

```
set.colors(x, palette, trim = 1)
```

Arguments

x	A vector
palette	function. A color palette function. Default is height.colors provided by the package lidR
trim	numeric.

stdmetrics *Predefined standard metrics functions*

Description

Predefined functions usable in grid_metrics, grid_hexametrics, lasmetrics, tree_metrics, and their convenient shortcuts. The philosophy of the lidR package is to provide an easy way to compute user-defined metrics rather than to provide them. However, for efficiency and to save time, a set of standard metrics has been predefined. To use these functions please read the Details and Examples sections.

Usage

```
stdmetrics(x, y, z, i, a, rn, class, dz = 1)

.stdmetrics

stdmetrics_z(z, dz = 1)

.stdmetrics_z

stdmetrics_i(i, z = NULL, class = NULL, rn = NULL)

.stdmetrics_i

stdmetrics_rn(rn, class = NULL)
```

```
.stdmetrics_rn

stdmetrics_pulse(pulseID, rn)

.stdmetrics_pulse

stdmetrics_ctrl(x, y, z, a)

.stdmetrics_ctrl

stdtreemetrics(x, y, z)

.stdtreemetrics
```

Arguments

`x, y, z, i, a`	Coordinates of the points, Intensity and ScanAngle
`rn, class`	ReturnNumber, Classification
`dz`	Layer thickness for metrics requiring this data, such as entropy
`pulseID`	The number referencing each pulse

Format

An object of class `expression` of length 1.

Details

The function names, their parameters and the output names of the metrics rely on a nomenclature chosen for brevity:

- `z`: refers to the elevation
- `i`: refers to the intensity
- `rn`: refers to the return number
- `q`: refers to quantile
- `a`: refers to the ScanAngle
- `n`: refers to a number (a count)
- `p`: refers to a percentage

For example the metric named `zq60` refers to the elevation, quantile, 60 i.e. the 60th percentile of elevations. The metric `pground` refers to a percentage. It is the percentage of points classified as ground. The function `stdmetric_i` refers to metrics of intensity. A description of each existing metric can be found on the lidR wiki page.

Some functions have optional parameters. If these parameters are not provided the function computes only a subset of existing metrics. For example `stdmetrics_i` requires the intensity values, but if the elevation values are provided it can compute additional metrics such as cumulative intensity at a given percentile of height.

Each function has a convenient associated variable. It is the name of the function, with a dot before the name. This enables the function to be used without writing parameters. The cost of such a feature is inflexibility. It corresponds to a predefined behaviour (see examples)

`stdtreemetrics` is a special function which works with tree_metrics. Actually it won't fail with other functions but the output make sense more sense if computed at the individual tree level.

See Also

grid_metrics lasmetrics grid_hexametrics grid_metrics3d tree_metrics

Examples

```
LASfile <- system.file("extdata", "Megaplot.laz", package="lidR")
lidar = readLAS(LASfile, all = TRUE)

# All the predefined functions
lidar %>% grid_metrics(stdmetrics(X,Y,Z,Intensity, ScanAngle,
                                  ReturnNumber, Classification,
                                  dz = 1))

# Convenient shortcut
lidar %>% grid_metrics(.stdmetrics)

# Basic metrics from intensities
lidar %>% grid_metrics(stdmetrics_i(Intensity))

# All the metrics from intensities
lidar %>% grid_metrics(stdmetrics_i(Intensity, Z, Classification, ReturnNumber))

# Convenient shortcut for the previous example
lidar %>% grid_metrics(.stdmetrics_i)

# Compute the metrics only on first return
lidar %>% lasfilterfirst %>% grid_metrics(.stdmetrics_z)

# Compute the metrics with a threshold at 2 meters
lidar %>% lasfilter(Z > 2) %>% grid_metrics(.stdmetrics_z)

# Works also with lasmetrics and grid_hexametrics
lidar %>% lasmetrics(.stdmetrics)
lidar %>% grid_hexametrics(.stdmetrics)

# Combine some predefined function with your own new metrics
# Here convenient shortcuts are no longer usable.
myMetrics = function(z, i)
{
  metrics = list(
     zwimean = sum(z*i)/sum(i), # Mean elevation weighted by intensities
     zimean  = mean(z*i),       # Mean products of z by intensity
     zsqmean = sqrt(mean(z^2))  # Quadratic mean
```

```
   )

  return( c(metrics, stdmetrics_z(z)) )
}

lidar %>% grid_metrics(myMetrics(Z, Intensity))

# You can write your own convenient shorcuts like this:
.myMetrics = expression(myMetrics(Z,Intensity))

lidar %>% grid_metrics(.myMetrics)
```

---

summary.LAS *LiDAR data summary*

Description

Print a summary of a LAS object

Usage

```
## S3 method for class 'LAS'
summary(object, ...)
```

Arguments

object	A LAS object
...	unused

---

tree_metrics *Compute metrics for each tree*

Description

Once the trees are segmented with lastrees, computes a series of descriptive statistics defined by the user for each individual tree. The output is a table in which each line is a tree, and each column is a metric. tree_metrics is similar to lasmetrics or grid_metrics or grid_metrics3d or grid_hexametrics, except it computes metrics for each segmented tree.

Usage

```
tree_metrics(.las, func, debug = FALSE)
```

Arguments

.las	An object of class LAS.
func	The function to be applied to each tree.
debug	logical. When facing a non trivial error, try debug = TRUE.

Details

The following existing functions contain a small set of pre-defined metrics:

- stdmetrics_tree

Users must write their own functions to create their own metrics. tree_metrics will dispatch the LiDAR data for each segmented tree in the user-defined function. Functions are defined without the need to considering each segmented tree i.e. only the point cloud (see examples).

Value

Returns a data.table containing the metrics for each segmented tree.

Examples

```
LASfile <- system.file("extdata", "Tree.laz", package="lidR")
las = readLAS(LASfile, filter = "-drop_z_below 0")

# segment trees (see lastrees)
lastrees(las, algorithm = "li2012")

# Max height for each tree
tree_metrics(las, mean(Z))

# Define your own new metrics
myMetrics = function(z, i)
{
  metrics = list(
     imean = mean(i),
     imax  = max(i),
     npoint = length(z)
   )

   return(metrics)
}

metrics = tree_metrics(las, myMetrics(Z, Intensity))

# predefined metrics (see ?stdmetrics)
metrics = tree_metrics(las, .stdtreemetrics)
```

VCI *Vertical Complexity Index*

Description

A fixed normalization of the entropy function (see references)

Usage

```
VCI(z, zmax, by = 1)
```

Arguments

z	vector of z coordinates
zmax	numeric. Used to turn the function entropy to the function vci.
by	numeric. The thickness of the layers used (height bin)

Value

A number between 0 and 1

References

van Ewijk, K. Y., Treitz, P. M., & Scott, N. A. (2011). Characterizing Forest Succession in Central Ontario using LAS-derived Indices. Photogrammetric Engineering and Remote Sensing, 77(3), 261-269. Retrieved from <Go to ISI>://WOS:000288052100009

See Also

entropy

Examples

```
z = runif(10000, 0, 10)

VCI(z, by = 1, zmax = 20)

z = abs(rnorm(10000, 10, 1))

# expected to be closer to 0.
VCI(z, by = 1, zmax = 20)
```

`writeLAS` *Write a las or laz file*

Description

Write a LAS object into a binary file (.las or .laz specified in filename)

Usage

```
writeLAS(.las, file)
```

Arguments

`.las`	an object of class LAS
`file`	character. A character string naming an output file

Value

void

Index