Package 'AMAPVox'

August 24, 2022

Type Package

Title LiDAR Data Voxelisation

Version 0.12.0

Description Read, manipulate and write voxel spaces. Voxel spaces are read from text-based output files of the 'AMAPVox' software. 'AMAPVox' is a LiDAR point cloud voxelisation software that aims at estimating leaf area through several theoretical/numerical approaches. See more in the article Vincent et al. (2017) <doi:10.23708/1AJNMP> and the technical note Vincent et al. (2021) <doi:10.23708/1AJNMP>.

License CeCILL (>= 2)

URL https://amapvox.org

BugReports https://github.com/umr-amap/AMAPVox/issues

Depends R (>= 4.0.0)

Imports curl, data.table, dplyr, methods, rappdirs, rvest, stringr, utils

Suggests fields, ggplot2, graphics, grDevices, knitr, RANN, rgl, RefManageR, rmarkdown, sf, terra

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.1

SystemRequirements Java 1.8 64-Bit (Oracle or Corretto)

Collate 'AMAPVox.R' 'Butterfly.R' 'Canopy.R' 'Classes.R' 'ComputeG.R' 'Crop.R' 'FillNA.R' 'Generics.R' 'Getters.R' 'Ground.R' 'Operators.R' 'PlantArea.R' 'PlotVoxelSpace.R' 'ReadVoxelSpace.R' 'VersionManager.R' 'Run.R' 'Utils-voxel.R' 'WriteVoxelSpace.R'

NeedsCompilation no

Author Grégoire Vincent [aut], Julien Heurtebize [aut], Philippe Verley [aut, cre]

Maintainer Philippe Verley <philippe.verley@ird.fr>

Repository CRAN

Date/Publication 2022-08-24 11:32:36 UTC

R topics documented:

AMAPVox *AMAPVox package*

Description

The package provides a a set of R functions for working with voxel spaces (read, write, plot, etc.). Voxel spaces are read from text-based output files of the AMAPVox software.

References

- Research paper first describing AMAPVox:
Vincent, G., Antin, C., Laurans, M., Heurtebize, J., Durrieu, S., Lavalley, C., & Dauzat, J. (2017). Mapping plant area index of tropical evergreen forest by airborne laser scanning. A cross-validation study using LAI2200 optical sensor. Remote Sensing of Environment, 198, 254-266. doi:10.1016/j.rse.2017.05.034
- Up-to-date description of PAD/LAD estimators implemented in AMAPVox:
VINCENT, Gregoire; PIMONT, François; VERLEY, Philippe, 2021, "A note on PAD/LAD estimators implemented in AMAPVox 1.7", doi:10.23708/1AJNMP, DataSuds, V1

Contact

<contact@amapvox.org>

Author(s)

Philippe VERLEY <philippe.verley@ird.fr>

See Also

Useful links:

- https://amapvox.org
- Report bugs at https://github.com/umr-amap/AMAPVox/issues

`butterfly` *Identify butterflies from a VoxelSpace object.*

Description

Identify butterflies from a `VoxelSpace` object.

A butterfly refers to a non-empty isolated voxel. Non-empty means that there is one or more hits recorded in the voxel. Isolated means that voxels in the Moore neighborhood of rank 1 are empty (no hit).

Usage

```
butterfly(vxsp)
```

Arguments

`vxsp`	a `VoxelSpace` object

Value

a list of voxel index (i, j, k) identified as butterfly.

See Also

clear()

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# identify butterflies
btf <- butterfly(vxsp)
# clear butterflies
clear(vxsp, butterfly(vxsp))
```

---

canopy *Extract canopy from voxel space.*

Description

Extract canopy from VoxelSpace object. The canopy layer is the set of highest voxels with number of hits greater than a user-defined threshold.

Minimum number of hits/echos:
Minimum number of hits is set by default to one, meaning that a single echo in a voxel is enough to consider that there is some vegetation. Increasing this threshold will tend to lower the canopy level or introduce some gaps (i-j-cells with no vegetation). This `hit.min` filter is stronger than butterfly() since is does not discriminate isolated voxels. A reasonable value for `hit.min` cannot be suggested ad-hoc since it strongly depends on sampling intensity. Removing butterflies prior to extracting canopy is advisable.

Gaps:
For a VoxelSpace with fully defined ground level (see ground()), missing canopy cells can be interpreted as gaps. Conversely if both ground and canopy are missing for a i-j-cell, then it is inconclusive.

Above/below canopy:
Function `aboveCanopy` returns voxel index above canopy level (excluded). Function `belowCanopy` returns voxel index below canopy level (included).

Canopy Height Model:
Function `canopyHeight` returns ground distance at canopy level, including gaps.

Usage

```
canopy(vxsp, hit.min = 1)

belowCanopy(vxsp, ...)

aboveCanopy(vxsp, ...)

canopyHeight(vxsp, ...)
```

Arguments

vxsp	a VoxelSpace object.
hit.min	a positive integer, minimum number of hit/echo in a voxel to consider it contains vegetation.
...	additional parameters which will be passed to canopy function. So far only hit.min parameter.

Value

data.table::data.table object with voxel index either below canopy, canopy level or above canopy

See Also

butterfly(), ground()

Examples

```
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
cnp <- canopy(vxsp)
acnp <- aboveCanopy(vxsp)
bcnp <- belowCanopy(vxsp)
# canopy layer included in below canopy subset
all(bcnp[cnp, on=list(i, j, k)] == cnp) # TRUE expected
vxsp@data[cnp, list(i, j, ground_distance), on=list(i, j, k)]
```

---

clear *Clear voxel*

Description

Clear a set of voxels. Clearing means that the state variables of the selected voxels are altered as if they were *clear* of any vegetation. Namely:

- number of echo set to zero
- intercepted beam surface set to zero (if variable is outputted)
- plant area density set to zero (if variable is outputted)
- transmittance set to one (if variable is outputted)
- any attenuation variable set to zero

Other state variables such as sampling intensity, mean angle, entering beam surface, etc. are unaltered. A cleared voxel is not the same as an unsampled voxel (not "crossed" by any beam).

Usage

```
clear(vxsp, vx)

## S4 method for signature 'VoxelSpace,data.table'
clear(vxsp, vx)

## S4 method for signature 'VoxelSpace,vector'
clear(vxsp, vx)

## S4 method for signature 'VoxelSpace,matrix'
clear(vxsp, vx)
```

Arguments

vxsp	a VoxelSpace object.
vx	(i, j, k) voxel coordinates as a data.table::data.table with i, j, k columns, a vector (i, j, k) or a matrix with i, j, k columns.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# clear 1st voxel
clear(vxsp, c(0, 0, 0)) # clear 1st voxel
# clear butterflies
clear(vxsp, butterfly(vxsp))
# clear voxels with less than two hits
clear(vxsp, vxsp@data[nbEchos < 2])
```

---

computeG *Foliage projection ratio G(theta).*

Description

Compute the mean projection of unit leaf area on the plane perpendicular to beam direction, namely, G(theta) parameter. Assumption of symmetric distribution of leaf azimuth angle. When estimating G for large amount of theta values, it is advised to enable the lookup table for speeding up the calculation.

Usage

```
computeG(
  theta,
  pdf = "spherical",
  chi,
  mu,
```

```
  nu,
  with.lut = length(theta) > 100,
  lut.precision = 0.001
)
```

Arguments

`theta`	a numeric vector, theta, the incident beam inclination, in radian, ranging `[0, pi/2]`.
`pdf`	the name of the probability density function of the leaf angle distribution. One of "uniform", "spherical", "planophile", "erectophile", "plagiophile", "extremophile", "ellipsoidal", "twoParamBeta". Refer to section "Leaf Angle Distribution functions" for details.
`chi`	a float, parameter of the ellipsoidal leaf angle distribution. The ratio the ratio horizontal axis over vertical axis. See section "Leaf Angle Ditribution functions" for details.
`mu`	a float, parameter controlling the Beta distribution. See section "Leaf Angle Distribution functions" for details.
`nu`	a float, parameter controlling the Beta distribution. See section "Leaf Angle Distribution functions" for details.
`with.lut`	a Boolean, whether to estimate G with a lookup table (LUT). By default the lookup table is automatically generated when length of theta vector is greater than 100.
`lut.precision`	a float, the increment of the theta sequence ranging from 0 to pi/2 for computing the lookup table.

Details

Leaf Angle Distribution functions

- de Wit's leaf angle distribution functions:
  - **uniform**, proportion of leaf angle is the same at any angle
  - **spherical**, relative frequency of leaf angle is the same as for surface elements of a sphere
  - **planophile**, horizontal leaves most frequent
  - **erectophile**, vertical leaves most frequent
  - **plagiophile**, oblique leaves most frequent
  - **extremophile**, oblique leaves least frequent
- **ellipsoidal** distribution function, generalization of the spherical distribution over an ellipsoid. Relative frequency of leaf angle is the same as for surface elements of an ellipsoid. Takes one parameter `chi` the ratio horizontal axis over vertical axis. For `chi = 1` the distribution becomes spherical. For `chi < 1`, the ellipsoid is a prolate spheroid (like a rugby ball). For `chi > 1` the ellipsoid is an oblate spheroid (a sphere that bulges at the equator and is somewhat squashed at the poles).
- **two parameters Beta** distribution. Most generic approach from Goal and Strebel (1984) to represent large variety of leaf angle distribution. Takes two parameters `mu` and `nu` that control the shape of the Beta distribution.

References

Wang, W. M., Li, Z. L., & Su, H. B. (2007). Comparison of leaf angle distribution functions: effects on extinction coefficient and fraction of sunlit foliage. Agricultural and Forest Meteorology, 143(1), 106-122.

See Also

`plotG()` for plotting G(theta) profiles

Examples

```
# G(theta) == 0.5 for spherical distribution
all(computeG(theta = runif(10, 0, pi/2)) == 0.5) # returns TRUE
# ellipsoidal distribution
computeG(theta = runif(10, 0, pi/2), pdf = "ellipsoidal", chi = 0.6)
```

---

crop *Crop voxel space*

Description

Crop `VoxelSpace` object based on voxel i, j, k, index. If cropping index are missing, the function will automatically crop the voxel space by discarding outermost unsampled slices of voxels. A *slice* designates a layer with constant i (i-slice), j (j-slice) or k (k-slice). *unsampled* means that no pulse went through.

One may want to crop the voxel space on coordinates rather than grid index. To do so the voxel space must be first converted to an `sf::sf` object and use the `sf::st_crop()` function.

```
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
vxsp@data[, c("x", "y"):=getPosition(vxsp)[, .(x, y)]]
library(sf)
vx.sf <- sf::st_as_sf(vxsp@data, coords=c("x", "y"))
vx.sf <- sf::st_crop(vx.sf, c(xmin = 4, ymin = 1, xmax = 5, ymax = 4))
sf::st_bbox(vx.sf)
vxsp@data <- sf::st_drop_geometry(vx.sf)
```

Usage

```
crop(vxsp, imin = 0, imax = Inf, jmin = 0, jmax = Inf, kmin = 0, kmax = Inf)
```

Arguments

vxsp	a `VoxelSpace` object.
imin	minimum i index of cropped area (inclusive)
imax	maximum i index of cropped area (inclusive)
jmin	minimum j index of cropped area (inclusive)

jmax	maximum j index of cropped area (inclusive)
kmin	minimum k index of cropped area (inclusive)
kmax	maximum k index of cropped area (inclusive)

Value

Cropped voxel space with updated i, j, k grid coordinates and updated header (min and max corner).

Examples

```
## Not run:
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
plot(crop(vxsp, imin = 1, imax = 5))
# introduce unsampled areas in voxel space
vxsp@data[i < 3, nbSampling:= 0]
# automatic cropping
plot(crop(vxsp))

## End(Not run)
```

Extract *Extract or Replace Parts of a VoxelSpace Object*

Description

Operators acting on VoxelSpace object. If user attempts

Usage

```
## S4 method for signature 'VoxelSpace'
x$name

## S4 method for signature 'VoxelSpace,ANY,missing'
x[[i, j, ...]]

## S4 replacement method for signature 'VoxelSpace'
x$name <- value

## S4 replacement method for signature 'VoxelSpace,ANY,missing'
x[[i, j]] <- value

## S4 replacement method for signature 'VoxelSpace'
x$name <- value
```

Arguments

x	a VoxelSpace object
name	A literal character string or a name (possibly backtick quoted).
i	string, name of elements to extract.
j	Unused.
...	Unused.
value	typically an array-like R object of a similar class as x.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))

# extract columns or header parameters
vxsp$nbSampling
vxsp[["i"]]
vxsp[["mincorner"]]

## Not run:
# add new column
vxsp[["pad_capped"]] <- ifelse(vxsp$PadBVTotal > 0.5, 0.5, vxsp$PadBVTotal)
# update header parameter
vxsp[["max_pad"]] <- 0.5

## End(Not run)
```

---

fillNA *Fill missing values (NA) with averaged neighboring data*

Description

Fill missing values of a given variable in a VoxelSpace object with averaged neighboring values.

Neighboring values are selected among voxels within a user-defined radius in meter and whose sampling rate (number of pulses that went through the voxel) is above a user-defined threshold. Distance between voxels is the euclidian distance between voxel centers. Fill-value may be capped by user-defined minimal and maximal values.

Default radius (if not defined by user) is set to largest dimension of voxel size `max(getVoxelSize(vxsp))`. It guarantees that default neighborhood is isotropic.

In some cases, for instance poorly sampled area, neighboring values may all be missing or discarded. A fallback value can be provided to "force fill" suche voxels. An other option is to run again the function with larger radius or lower sampling threshold.

Usage

```
fillNA(
  vxsp,
  variable.name,
  variable.min = -Inf,
  variable.max = Inf,
  variable.fallback,
  radius,
  pulse.min = 10
)
```

Arguments

vxsp	a VoxelSpace object.
variable.name	a character, the name of a variable in the VoxelSpace
variable.min	a numeric, minimal value for the fill values
variable.max	a numeric, maximal value for the fill values
variable.fallback	a numeric, optional fallback value in case no fill value can be estimated from neighboring voxels.
radius	a numeric, the radius in meter that defines the neighborhood of a voxel. The function looks for the voxels whose center is inside a sphere of radius radius centered at current voxel center. Default is set to max(getVoxelSize(vxsp))
pulse.min	a numeric, minimal sampling intensity (i.e. number of pulses that went through a voxel) to include neighboring voxel in the estimation of the averaged fill value.

Examples

```
# read voxel space
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# Randomly add some NA in PAD variable
vx <- vxsp@data
ind <- sample(vx[PadBVTotal > 0, which = TRUE], 3)
# print initial values
vx[ind, .(i, j, k, PadBVTotal)]
vx[ind, PadBVTotal := NA]
# fill NA in PAD variable
fillNA(vxsp, "PadBVTotal", variable.max = 5)
# print filled values
vx[ind, .(i, j, k, PadBVTotal)]
```

getLocalVersions *List local AMAPVox versions.*

Description

List AMAPVox versions already installed on your computer by the package. AMAPVox versions are installed in the user-specific data directory, as specified by `rappdirs::user_data_dir()`.

Usage

```
getLocalVersions()
```

Value

a `data.frame` with 2 variables: `$version` that stores the version number and `$path` the local path of the AMAPVox directory.

See Also

`getRemoteVersions()`, `rappdirs::user_data_dir()`

getMaxCorner *Gets the x, y, z coordinates of the voxel space top right corner.*

Description

Gets the x, y, z coordinates of the voxel space top right corner.

Usage

```
getMaxCorner(vxsp)

## S4 method for signature 'VoxelSpace'
getMaxCorner(vxsp)
```

Arguments

`vxsp`	the `VoxelSpace` object.

Value

the x, y, z coordinates of the voxel space top right corner, as a numerical vector.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# retrieve 'max_corner' parameter
getMaxCorner(vxsp)
```

`getMinCorner` *Gets the x, y, z coordinates of the voxel space bottom left corner.*

Description

Gets the x, y, z coordinates of the voxel space bottom left corner.

Usage

```
getMinCorner(vxsp)

## S4 method for signature 'VoxelSpace'
getMinCorner(vxsp)
```

Arguments

`vxsp`	the `VoxelSpace` object.

Value

the x, y, z coordinates of the voxel space bottom left corner, as a numerical vector.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# retrieve 'min_corner' parameter
getMinCorner(vxsp)
```

`getParameter` *Gets a parameter from the VoxelSpace header.*

Description

Gets a parameter from the VoxelSpace header.

Usage

```
getParameter(vxsp, what)

## S4 method for signature 'VoxelSpace,character'
getParameter(vxsp, what)

## S4 method for signature 'VoxelSpace,missing'
getParameter(vxsp, what)
```

Arguments

vxsp	the VoxelSpace object
what	the name of the parameter. If missing returns all parameters.

Value

the parameter as a character

See Also

VoxelSpace

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# show parameters name
names(getParameter(vxsp))
# retrieve 'mincorner' parameter
getParameter(vxsp, "mincorner")
# all parameters
getParameter(vxsp)
```

---

getPosition *Gets the x, y, z coordinates of a given voxel.*

Description

Gets the x, y, z coordinates of the voxel center. If the voxel parameter is missing, it returns the positions of all the voxels in the voxel space.

Usage

```
getPosition(vxsp, vx)

## S4 method for signature 'VoxelSpace,vector'
getPosition(vxsp, vx)
```

```
## S4 method for signature 'VoxelSpace,matrix'
getPosition(vxsp, vx)

## S4 method for signature 'VoxelSpace,data.table'
getPosition(vxsp, vx)

## S4 method for signature 'VoxelSpace,missing'
getPosition(vxsp, vx)
```

Arguments

vxsp	a VoxelSpace object.
vx	(i, j, k) voxel coordinates as a data.table::data.table with i, j, k columns, a vector (i, j, k) or a matrix with i, j, k columns.

Value

the x, y, z coordinates of the voxel center.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))

# get position of voxel(i=0, j=0, k=0)
getPosition(vxsp, c(0, 0, 0))

# get position of voxels 1 to 10 in the data.table
getPosition(vxsp, vxsp@data[1:10,])

# get positions of every voxel
getPosition(vxsp)
```

---

getRemoteVersions *List remote AMAPVox versions.*

Description

List AMAPVox versions available for download from page https://amap-dev.cirad.fr/projects/amapvox/files

Usage

```
getRemoteVersions()
```

Value

a data.frame with 2 variables: $version that stores the version number and $url the URL of the associated ZIP file.

See Also

getLocalVersions()

---

getVoxelSize *Gets the elemental size of a voxel (dx, dy, dz) in meter.*

Description

Gets the elemental size of a voxel (dx, dy, dz) in meter.

Usage

```
getVoxelSize(vxsp)

## S4 method for signature 'VoxelSpace'
getVoxelSize(vxsp)
```

Arguments

vxsp	the VoxelSpace object.

Value

the size of the voxel in meter, as a numerical vector.

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# retrieve voxel size
getVoxelSize(vxsp)
```

---

ground *Extract ground from voxel space.*

Description

Extract ground layer from VoxelSpace object.

Ground layer:

The ground layer is the set of voxels that are just above ground level. The bottom facet of the voxel must be above ground `ground_distance(voxel_center) >= dz/2` with dz the voxel size on z axis. Ground layer may be missing (the function returns an empty data.table) or incomplete (the function returns a data.table with `nrow(ground(vxsp)) < prod(dim(vxsp)[1:2])`) for some voxel space.

Above/below ground:

Function `aboveGround` returns voxel index above ground layer (included). Function `belowGround` returns voxel index below ground layer (excluded).

Ground energy:

Function `groundEnergy` estimates fraction of light reaching the ground. It is computed as the ratio of entering beam section on potential beam section (beams that would have crossed a voxel if there were no vegetation in the scene). It requires variables *bsEntering* and *bsPotential*.

Ground elevation:

Function `groundElevation` returns the elevation of the ground layer. It is provided as a check function, to make sure that AMAPVox *digital elevation model* is consistent with the one provided in input.

Usage

```
ground(vxsp)

belowGround(vxsp)

aboveGround(vxsp)

groundEnergy(vxsp)

groundElevation(vxsp)
```

Arguments

vxsp	a `VoxelSpace` object.

Value

`data.table::data.table` object with voxel index either below ground, ground level or above ground.

Examples

```
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
gr <- ground(vxsp)
ag <- aboveGround(vxsp)
bg <- belowGround(vxsp) # empty in test case
# ground layer included in above ground subset
all(ag[gr, on=list(i, j, k)] == gr) # TRUE expected
vxsp@data[ag, on=list(i, j, k)]
```

`installVersion` *Install specific AMAPVox version on local computer.*

Description

Install specific AMAPVox version on your computer. AMAPVox versions are installed in the user-specific data directory, as specified by `rappdirs::user_data_dir()`. You should not worry to call directly this function since local installations are automatically handled by the version manager when you launch AMAPVox GUI with `gui()` function.

Usage

```
installVersion(version, overwrite = FALSE)
```

Arguments

`version,`	a valid and existing AMAPVox remote version number (major.minor.build)
`overwrite,`	whether existing local installation should be re-installed.

Value

the path of the AMAPVox installation directory.

See Also

`getLocalVersions()`, `getRemoteVersions()`, `removeVersion()`

`rappdirs::user_data_dir()`

Examples

```
## Not run:
# install latest version
installVersion(tail(getRemoteVersions()$version, 1))

## End(Not run)
```

`merge.VoxelSpace` *Merge two voxel spaces*

Description

Merge of two `VoxelSpace` object. Voxel spaces must have same sptial extension and resolution, and some shared column names.

Merging modes:

Variables `i, j, k & ground_distance` are merged.

Variables `nbEchos, nbSampling, lgTotal, bsEntering, bsIntercepted, bsPotential, weightedEffectiveFreep` are summed-up.

Variables `sdLength, angleMean and distLaser` are weighted means with `nbSampling` (the number of pulses) as weights.

Attenuation FPL variables (`attenuation_FPL_biasedMLE, attenuation_FPL_biasCorrection, attenuation_FPL_ur`) are calculated analytically.

Transmittance and attenuation variables (except the FPL attenuation variables listed above) are weighted means with bsEntering as weights.

Any other variables will not be merged. In particular PAD variables are not merged and should be recalculated with `plantAreaDensity()` on the merged voxel space. E.g:`vxsp <- plantAreaDensity(merge(vxsp1, vxsp2))`

Merging multiple voxel spaces:

Merging several voxel spaces works as follow : vxsp1 and vxsp2 merged into vxsp12. vxsp12 & vxsp3 merged into vxsp123, etc. The process can be synthesized with the `Reduce()` function.

```
vxsp <- Reduce(merge, list(vxsp1, vxsp2, vxsp3))
```

Usage

```
## S3 method for class 'VoxelSpace'
merge(x, y, ...)
```

Arguments

`x, y`	`VoxelSpace` objects to be merged.
`...`	Not used

Value

A merged `VoxelSpace` object.

Examples

```
# merge same voxel space to confirm merging behavior
vxsp1 <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
vxsp2 <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
vxsp <- merge(vxsp1, vxsp2)
all(vxsp$nbSampling == vxsp1$nbSampling + vxsp2$nbSampling)

# with PAD
vxsp <- plantAreaDensity(merge(vxsp1, vxsp2), pulse.min = 1)
all((vxsp$pad_transmittance - vxsp1$PadBVTotal) < 1e-7) # equal at float precision
```

`plantAreaDensity` *Plant Area Density (PAD)*

Description

Computes Plant Area Density either from transmittance or attenuation coefficient estimates. Details of calculation and underlying assumptions can be found online at doi:10.23708/1AJNMP. PAD is defind as the plant area per unit volume (PAD plant area / voxel volume = m^2 / m^3).

Usage

```
plantAreaDensity(
  vxsp,
  vx,
  lad = "spherical",
  angle.name = "angleMean",
  variable.name = c("transmittance", "attenuation_FPL_unbiasedMLE",
    "attenuation_PPL_MLE"),
  pad.max = 5,
  pulse.min = 5,
  ...
)
```

Arguments

`vxsp`	a `VoxelSpace` object.
`vx`	a subset of voxel index. A data.table with `i, j, k` columns. Missing parameter means whole voxel space.
`lad`	the name of the probability density function of the leaf angle distribution. One of `AMAPVox:::leafAngleDistribution`.
`angle.name`	the name of the mean angle variable in the VoxelSpace object.
`variable.name`	the name of the transmittance/attenuation variables in the VoxelSpace object. Transmittance variables are expected to start with "tra" and attenuation variables with "att".

pad.max	a float, the maximal PAD value
pulse.min	an integer, the minimal number of pulses in a voxel for computing the PAD. PAD set to NA otherwise.
...	additional parameters which will be passed to the leaf angle distribution functions. Details in computeG().

Value

A voxel space object with the requested PAD variables.

References

VINCENT, Gregoire; PIMONT, François; VERLEY, Philippe, 2021, "A note on PAD/LAD estimators implemented in AMAPVox 1.7", doi:10.23708/1AJNMP, DataSuds, V1

See Also

computeG()

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# compute PAD
pad <- plantAreaDensity(vxsp, variable.name = "attenuation_PPL_MLE")
# merge pad variables into voxel space
vxsp@data <- merge(vxsp@data, pad, on = list(i, j, k))
grep("^pad", names(vxsp), value = TRUE) # print PAD variables in vxsp
# PAD on a subset
pad.i2j3 <- plantAreaDensity(vxsp, vxsp@data[i ==2 & j==3, .(i, j, k)])
pad.i2j3[["ground_distance"]] <- vxsp@data[i ==2 & j==3]$ground_distance
## Not run:
# plot vertical profile
library(ggplot2)
# meld data.table (wide-to-long reshaping)
pad <- data.table::melt(pad.i2j3,
  id.vars = "ground_distance",
  measure.vars = c("pad_transmittance", "pad_attenuation_FPL_unbiasedMLE",
    "pad_attenuation_PPL_MLE"))
ggplot(data = pad, aes(x=value, y=ground_distance, color=variable)) +
  geom_path() + geom_point()

## End(Not run)
```

`plantAreaIndex` *Plant Area Index (PAI)*

Description

Computes Plant Area Index (PAI) from Plant Area Density (PAD). PAI is defined as the plant area per unit ground surface area (PAI = plant area / ground area = m^2 / m^2).

The function can estimate PAI on the whole voxel space or any region of interest (parameter vx subset of voxels). It can compute PAI from several perspectives : either an averaged PAI value, a two-dimensions (i, j) PAI array or vertical profiles either above ground or below canopy.

Usage

```
plantAreaIndex(
  vxsp,
  vx,
  type = c("av", "ag", "bc", "xy"),
  pattern.pad = "^pad_*"
)
```

Arguments

`vxsp`	a `VoxelSpace` object.
`vx`	a subset of voxel index. A data.table with `i, j, k` columns. Missing parameter means whole voxel space.
`type`	a character vector, the type of PAI profile. • `"av"` Averaged value on every voxel • `"ag"` Above ground vertical profile • `"bc"` Below canopy vertical profile • `"xy"` Spatial profile
`pattern.pad`	character string containing a regular expression to be matched in the voxel space variable names, for selecting PAD variables. Typing the name of a specific PAD variable works just fine.

Value

Returns a list of PAI profiles for requested PAD variables and PAI types.

`av` **Averaged PAI:**
Returns a single value. Calculated as the sum of PAD values multiplied by voxel volume and divided by ground surface with vegetation.

`ag` & `bc` **Above ground and below canopy PAI vertical profile:**
Returns a vertical profile of PAI values either from ground distance or canopy depth. Calculated as the averaged PAD values per layer (a layer being defined by either the distance to ground or canopy level) multiplied by voxel size along z (equivalent to multiplying PAD by voxel volume and dividing by voxel ground surface).

xy **Spatial PAI profile:**
Returns a list a PAI values by i, j index. Calculated as the sum of PAD on (i, j) column multiplied by voxel size along z (equivalent to multiplying PAD by voxel volume and dividing by voxel ground surface).

See Also

plantAreaDensity()

Examples

```
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
vxsp@data <- merge(vxsp@data, plantAreaDensity(vxsp), on = list(i, j, k))
## Not run:
lai <- plantAreaIndex(vxsp)
names(lai)
library(ggplot2)
ggplot(data = lai[["pad_transmittance.pai.ag" ]], aes(x=pai, y=ground_distance)) +
  geom_path() + geom_point()

## End(Not run)
# PAI on a subset
ni <- round(dim(vxsp)[1]/2)
vx <- vxsp@data[i < ni, .(i, j, k)]
lai <- plantAreaIndex(vxsp, vx)
```

---

plot *Plot an object of class VoxelSpace*

Description

plot a VoxelSpace object.

Usage

```
plot(x, y, ...)

## S4 method for signature 'VoxelSpace,missing'
plot(
  x,
  y,
  variable.name = "nbSampling",
  palette = "viridis",
  bg.color = "lightgrey",
  width = 640,
  voxel.size = 5,
  unsampled.discard = TRUE,
```

```
  empty.discard = TRUE,
  ...
)

## S4 method for signature 'VoxelSpace,data.table'
plot(
  x,
  y,
  variable.name = "nbSampling",
  palette = "viridis",
  bg.color = "lightgrey",
  width = 640,
  voxel.size = 5,
  unsampled.discard = TRUE,
  empty.discard = TRUE,
  ...
)
```

Arguments

x	the object of class VoxelSpace to plot
y	a subset of voxel index. A data.table with `i, j, k` columns. Missing parameter means whole voxel space.
...	additional parameters which will be passed to `rgl::plot3d()`.
variable.name	character, the name of the variable to plot
palette	character, a valid palette name (one of hcl.pals())
bg.color	character, a valid background color name (one of colors())
width	numeric, the width of the windows
voxel.size	numeric, the size of voxel in pixels
unsampled.discard	logical, whether to discard unsampled voxel
empty.discard	logical, whether to discard empty voxel (no hit)

Details

Plot an object of class VoxelSpace in a 3d device. By default it plots the sampling intensity but the user can choose any variable available in the voxel file.

See Also

`rgl::plot3d()`

Examples

```
## Not run:
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
```

```
# plot sampling intensity by default
plot(vxsp)
# plot PAD
plot(vxsp, variable.name = "PadBVTotal", palette = "YlOrRd")
# plot a subset
plot(vxsp, vxsp@data[k > 4, .(i, j, k)])

## End(Not run)
```

---

plotG *Plot G(theta) profiles for one or several leaf angle distribution functions*

Description

Plot G(theta) profiles for one or several leaf angle distribution functions with `theta in [0, pi/2]`. Requires ggplot2 package.

Usage

```
plotG(pdf = leafAngleDistribution, chi = 0.6, mu = 1.1, nu = 1.3)
```

Arguments

pdf	the name of the leaf angle distribution functions. One of "uniform", "spherical", "planophile", "erectophile", "plagiophile", "extremophile", "ellipsoidal", "twoParamBeta".
chi	a float, parameter of the ellipsoidal leaf angle distribution. The ratio the ratio horizontal axis over vertical axis. See section "Leaf Angle Ditribution functions" for details.
mu	a float, parameter controlling the Beta distribution. See section "Leaf Angle Distribution functions" for details.
nu	a float, parameter controlling the Beta distribution. See section "Leaf Angle Distribution functions" for details.

Examples

```
## Not run:
# plot G(theta) for planophile leaf angle distribution function
AMAPVox::plotG(pdf = "planophile")
# plot G(theta) for every distributions
AMAPVox::plotG()

## End(Not run)
```

readVoxelSpace *Read a voxel file*

Description

read a voxel file and cast it into a VoxelSpace object.

Zipped voxel file is accepted. AMAPVox uses user cache directory to unzip the file (rappdirs::user_cache_dir()).

Usage

```
readVoxelSpace(f)
```

Arguments

f	The path of the voxel file.

See Also

writeVoxelSpace()

Examples

```
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
```

removeVersion *Remove specific AMAPVox version from local computer.*

Description

Uninstall specific AMAPVox version from your computer.

Usage

```
removeVersion(version)
```

Arguments

version,	a valid and existing AMAPVox local version number (major.minor.build)

See Also

getLocalVersions(), installVersion()

Examples

```
## Not run:
# uninstall oldest version from your computer
removeVersion(head(getLocalVersions()$version, 1))

## End(Not run)
```

`run` *Run AMAPVox*

Description

Run AMAPVox either in batch mode or with Graphical User Interface (GUI). The function embeds a version manager for installing locally any version available remotely. AMAPVox relies on Java 64 bit. It has been compiled with JDK 8 64-Bit Oracle. For running AMAPVox in batch mode, any version of Java 64-bit >= 8 should work. AMAPVox GUI relies additionally on JavaFX. Refer to following section for installing a suitable Java/JavaFX.

`gui` function as been kept for background compatibility. It is an alias of the `run` function.

Usage

```
run(
  version = "latest",
  xml,
  java = "java",
  jvm.options = "-Xms2048m",
  nt = 1,
  ntt = 1,
  stdout = ""
)

gui(version = "latest", java = "java", jvm.options = "-Xms2048m", stdout = "")
```

Arguments

`version,`	either "latest" or a valid version number major.minor(.build) if `version="latest"` the function looks for latest remote version. If there is no internet connection it runs latest local version.
`xml`	path(s) to AMAPVox XML configuration files. If missing or `NULL` AMAPVox launches the GUI.
`java`	path to the java executable. Default 'java' value assumes that java is correctly defined on the $PATH variable.
`jvm.options`	JVM (Java Virtual Machine) options. By default it allocates 2Go of heap memory to AMAPVox.
`nt`	maximum number of threads for running tasks. `nt=1` means sequential execution. `nt=0` means as many threads as available.

ntt	maximum number of threads per task. ntt=0 means as many threads as available.
stdout	where output from both stdout/stderr should be sent. Same as stdout & stderr options from function system2().

Java 1.8 64-Bit with JavaFX

AMAPVox GUI relies on Java 1.8 64-Bit and JavaFX. In practice it requires either Java 1.8 64-Bit Oracle or Java 1.8 64-Bit Corretto. OpenJDK 8 will not work since JavaFX is not included. You may check beforehand if java is installed on your system and which version.

```
system2("java", args = "-version")
```

If AMAPVox::gui keeps throwing errors after you have installed suitable Java 1.8 64-Bit, it means that Java 1.8 may not be properly detected by your system. In such case you may have to check and set the JAVA_HOME environment variable.

```
Sys.getenv("JAVA_HOME")
Sys.setenv(JAVA_HOME="path/to/java/1.8/bin")
system2("java", args = "-version")
```

As a last resort you may change the java parameter of this function and set the full path to Java 1.8 binary.

```
AMAPVox::run(java = "/path/to/java/1.8/bin/java")
```

See Also

getLocalVersions(), getRemoteVersions(), installVersion() and removeVersion()

Examples

```
## Not run:
# (install and )run latest AMAPVox version with GUI
AMAPVox::run()
# (install and )run version 1.6.4 with GUI
AMAPVox::run(version="1.6.4")
# run latest AMAPVox version with XML configuration
AMAPVox::run(xml="/path/to/cfg.xml")
# run multiple configurations
AMAPVox::run(xml=c("cfg1.xml", "cfg2.xml"), nt=2)

## End(Not run)
```

tools *Tools inherited from base R for VoxelSpace object.*

Description

Tools inherited from base R for VoxelSpace objects.

Usage

```
## S4 method for signature 'VoxelSpace'
show(object)

## S3 method for class 'VoxelSpace'
print(x, ...)

## S3 method for class 'VoxelSpace'
length(x)

## S3 method for class 'VoxelSpace'
dim(x)

is.VoxelSpace(x)

## S4 method for signature 'VoxelSpace'
ncol(x)

## S4 method for signature 'VoxelSpace'
nrow(x)

## S3 method for class 'VoxelSpace'
names(x)
```

Arguments

object	a VoxelSpace object.
x	a VoxelSpace object.
...	further arguments passed to print function.

Note on length.VoxelSpace

AMAPVox allows to discard empty voxels in the voxel file. In such case length.VoxelSpace will return the expected number of voxels as if none were missing. As a consequence the number of voxels stored in the VoxelSpace object may be inferior to the returned value, namely nrow(x) <= length(x)

toRaster *Voxel layer to raster*

Description

Converts a voxel space (i, j) layer into a terra::SpatRaster object.

Usage

```
toRaster(vxsp, vx)
```

Arguments

vxsp	a VoxelSpace object.
vx	a voxel space horizontal slice. A data.table with i, j columns and least one additional variable, the value of the raster layer. Every column beside i and j will be converted into a raster layer.

Value

a terra::SpatRaster object.

Examples

```
## Not run:
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
library(terra)

# CHM, DEM and PAI as raster
plot(toRaster(vxsp, merge(canopyHeight(vxsp), groundElevation(vxsp), all = T)))

# PAI
vxsp <- plantAreaDensity(vxsp)
pai <- plantAreaIndex(vxsp, type = "xy", pattern.pad = "pad_transmittance")
plot(toRaster(vxsp, pai))

# sampling intensity at 2 meters
plot(toRaster(vxsp, vxsp@data[ground_distance == 2.25, .(i, j, nbSampling)]))

## End(Not run)
```

VoxelSpace-class *VoxelSpace*

Description

Class that holds the state variables of every voxel of the voxel space in a `data.table::data.table` object, plus metadata from the voxel space header.

Value

An object of class VoxelSpace.

Slots

`file` the path of the voxel file (.vox).

`data` the voxels hold in a data.table.

`header` a list of parameters associated to this voxel file.

See Also

`readVoxelSpace()`

writeVoxelSpace *Write a voxel file*

Description

write a voxel file out of a `VoxelSpace` object.

Usage

```
writeVoxelSpace(vxsp, f)
```

Arguments

`vxsp`	the object of class VoxelSpace to write
`f`	a character string naming a file.

See Also

`readVoxelSpace()`

Examples

```
## Not run:
# load a voxel file
vxsp <- readVoxelSpace(system.file("extdata", "tls_sample.vox", package = "AMAPVox"))
# set max PAD to 5
vxsp@data[, PadBVTotal:=sapply(PadBVTotal, min, 5)]
# write updated voxel file in temporary file
writeVoxelSpace(vxsp, tempfile("pattern"="amapvox_", fileext=".vox"))

## End(Not run)
```

Index