Package 'viewshed3d'

May 31, 2019

Title Compute Viewshed in 3D Terrestrial Laser Scanner Scenes of Ecosystems

Version 2.0.0

Description A set of tools to compute viewshed in 3D from Terrestrial Laser Scanner data and prepare the data prior to visibility calculation.

Depends R (>= 3.4.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports VoxR (>= 0.5.1), rgl (>= 0.99.16), doParallel (>= 1.0.14), foreach (>= 1.4.4), tcltk2 (>= 1.2.11), iterators (>= 1.0.10), tcltk (>= 3.4.4), utils (>= 3.4.4), data.table (>= 1.12.0), viridis (>= 0.5.1)

NeedsCompilation no

Author Bastien Lecigne [aut, cre], Jan Eitel [aut]

Maintainer Bastien Lecigne <lecignebastien@gmail.com>

Repository CRAN

Date/Publication 2019-05-30 22:30:04 UTC

R topics documented:

viewshed3d-package 2
class.ground 3
cum.view 4
ground.filter 5
param_1 6
param_10 6
param_2 6
param_3 6

param_4 6
param_5 7
param_6 7
param_7 7
param_8 7
param_9 7
reconstruct.ground 8
reconstruct.ground.par 9
reshape.scene 10
TLSclass 11
TLSrecons 11
TLSscene 11
view 11
view.param 12

Index **14**

`viewshed3d-package` *The* **viewshed3d** *package to compute visibility in 3D scenes of ecosystems acquired with Terrestrial Laser Scanner.*

Description

For many animals, the ability to visually assess the environment and detect approaching predators is an important part of anti-predator strategies. Because this can occur across spatial scales, estimation of the viewshed can help to quantify visibility as a continuous variable around animal locations and facilitate studies of habitat selection and predator-prey interactions.
The viewshed3d package provides tools to compute viewshed in 3D scenes of an ecosystem acquired with a Terrestrial Laser Scanner (TLS). The main function of the package, `view`, is designed to sample the point cloud in every direction of the 3D space from a single user-defined location and to record the distance to the nearest point in each direction. Each direction is thus considered as a slightline - of a user defined angle - that is assumed to end when an object - of undefined nature and size - is encountered. The `cum.view` function computes the overlap between viewsheds calculated from different locations. The other functions of the package are designed to properly prepare the data before the visibility calculation. First, the ground has to be reconstructed as occlusion might induce holes that would result in infinite slightlines to continue below the ground surface. To do so, the `class.ground` and `ground.filter` functions can be used to segment a TLS scene into ground and vegetation points. The `reconstruct.ground` and r`reconstruct.ground.par` functions can then be used to fill the holes by reconstructing the ground point cloud. Additionaly, the `reshape.scene` function can be used at any time during the process to reshape the point cloud and define its central location (that will be used as the starting point for slightlines) in order to speed up the subsequent processes if the original point cloud is very large. Finally, the `view.param` function is provided to build the parameter data required to run the `view` function. Considering the relatively long processing time required to run the *view.param* function, some predefined parameter data are provided for different slightline angles.

Author(s)

Bastien Lecigne and Jan Eitel

Maintainer : Bastien Lecigne <lecignebastien@gmail.com>

`class.ground` *Ground and vegetation segmentation in TLS scene*

Description

The **class.ground** function combines three filters to segment the soil and the vegetation in a TLS scene of a forest. In each pixel of a grid of *grid.res* resolution, the points with a z value between *min(z)* and *min(z) + ground.thickness* are first classified as ground. Then, in each pixel, the ground points with z above a *pix.percentile* percentile threshold are reclassified as vegetation. During this process, the z values of each point relative to the lower z value of all points within the pixel are recorded. This enables application of a last percentile filter apllied at the plot scale, such that all points with a z value higher than *global.percentile* are reclassified as vegetation.

Usage

```
class.ground(data, grid.res, ground.thickness, pix.percentile,
  global.percentile, plot3d)
```

Arguments

`data`	a data.frame containing xyz coordinates
`grid.res`	resolution of the grid's pixel. Default = 1.
`ground.thickness`	the thickness of the soil layer used for primary segmentation. Default = 0.2
`pix.percentile`	the percentile filter to use in each pixel. Default = 100 (i.e., no filter)
`global.percentile`	the percentile filter to use at the plot scale. Default = 100 (i.e., no filter)
`plot3d`	logical, if *TRUE* the result of the segmentation is plotted with the soil in brown and the vegetation in green

Details

The two percentile based filters were successfully tested in: Muir, J., Goodwin, N., Armston, J., Phinn, S., & Scarth, P. (2017). An accuracy assessment of derived digital elevation models from terrestrial laser scanning in a sub-tropical forested environment. Remote Sensing, 9(8), 843.

Value

a data frame containing xyz coordinates plus the point class: "ground" or "vegetation"

Examples

```
library(viewshed3d)
data(TLSscene)
segmented.scene=class.ground(TLSscene, plot3d = TRUE)
```

`cum.view` *Compute cumulated viewsheds in a 3D TLS scene from multiple location*

Description

Compute cumulated viewsheds in a 3D TLS scene from multiple location

Usage

```
cum.view(data, locations, param, vox_res)
```

Arguments

`data`	a data.frame containing the xyz coordinates of a TLS point cloud
`locations`	a data.frame with each row containing the xyz coordinates of one point representing the animal location at a single point in time.
`param`	a data.frame containing the directions that will be used to explore the point cloud
`vox_res`	the voxel resolution

Value

a data frame containing the x,y,z coordinates of a voxel cloud plus the number of locations from which each voxel could be seen

Examples

```
library(viewshed3d)
data(TLSrecons)
data(param_2)

locations=data.frame(c(12,12),c(58,60),c(0,0))

cum=cum.view(TLSrecons,param = param_2,location=locations,vox_res = 0.2)

library(viridis)
library(rgl)
col=cividis(max(cum[,4])+1)

plot3d(cum,col=col[cum[,4]+1],add=TRUE,size=8)
points3d(locations,col="red",size=5)
```

`ground.filter` *Additionnal filters after first ground and vegetation segmentation*

Description

The **ground.filter** function provides the ability to filter an output from the `class.ground` function. The first filter removes from the ground point cloud every point with a z value above *pix.percentile* threshold within a single pixel of *grid.res* resolution. The second filter is a global percentile filter that removes from the ground point cloud the points with a within-pixel z value above *global.percentile* threshold.

Usage

```
ground.filter(data, grid.res, pix.percentile, global.percentile)
```

Arguments

`data`	a data.frame containing the xyz and point class of ground points segmented with the `class.ground` function
`grid.res`	resolution of a grid's pixel. Default = 1
`pix.percentile`	the percentile filter to use in each pixel. Default = 50
`global.percentile`	the percentile filter to use at the plot scale. Default = 95

Value

a data.frame containing the xyz coordinates plus the point class: "ground" or "reclass to veg." if the point was classed as vegetation.

Examples

```
library(viewshed3d)
data(TLSclass)

ground=subset(TLSclass,TLSclass[,4]=='ground')
resegmented=ground.filter(ground)

ground = subset(resegmented,resegmented[,4]=='ground')
vegetation = subset(resegmented,resegmented[,4]=='reclass to veg.')

library(rgl)
open3d()
plot3d(ground,col="brown",add=TRUE)
plot3d(vegetation,col="darkgreen",add=TRUE)
```

param_1 *A parameter data set for the view function with* **angular.resolution = 1**

Description

A parameter data set for the view function with **angular.resolution = 1**

param_10 *A parameter data set for the view function with* **angular.resolution = 10**

Description

A parameter data set for the view function with **angular.resolution = 10**

Parameters for the `view` function with an angular resolution of 10 and no gap filling

param_2 *A parameter for the view function with* **angular.resolution = 2**

Description

A parameter for the view function with **angular.resolution = 2**

param_3 *A parameter for the view function with* **angular.resolution = 3**

Description

A parameter for the view function with **angular.resolution = 3**

param_4 *A parameter for the view function with* **angular.resolution = 4**

Description

A parameter for the view function with **angular.resolution = 4**

`param_5`	*A parameter for the view function with* **angular.resolution = 5**

Description

A parameter for the view function with **angular.resolution = 5**

`param_6`	*A parameter for the view function with* **angular.resolution = 6**

Description

A parameter for the view function with **angular.resolution = 6**

`param_7`	*A parameter for the view function with* **angular.resolution = 7**

Description

A parameter for the view function with **angular.resolution = 7**

`param_8`	*A parameter for the view function with* **angular.resolution = 8**

Description

A parameter for the view function with **angular.resolution = 8**

`param_9`	*A parameter for the view function with* **angular.resolution = 9**

Description

A parameter for the view function with **angular.resolution = 9**

`reconstruct.ground` *Reconstruct the missing portions of a TLS scene's ground*

Description

The **reconstruct.ground** function is used to reconstruct the missing portions of the ground after segmentation of the TLS scene with the `class.ground` function. It creates a grid of pixels of user defined resolution and attributes to each pixel an elevation value (i.e., a z value) calculated as the average value (weighted by distance) of the neighboring ground points. Two methods for neighboring points selection are provided: k nearest neighbors or absolute cartesian distance.

Usage

```
reconstruct.ground(data, grid.res, scene.radius, scene.center, shape,
  method, d, k, reconstruct.all)
```

Arguments

`data`	a data.frame containing the xyz and point class of ground points segmented with the `class.ground` and optionally with the `ground.filter` functions
`grid.res`	the resolution of a grid used to locate the new points to add. Default = 0.05
`scene.radius`	optional. Define scene radius relative to the 0,0,0 coordinate. Should be used after the scene was reshaped with the `reshape.scene function`
`scene.center`	a vector containing the xyz coordinates of the scene center for reshaping
`shape`	the shape of the scene radius reshaping (see `reshape.scene` function for more details)
`method`	the method to find the neighboring points. If = "distance" all the points located within a distance d are used to estimate z. If = "k-nearest" only the k nearest neighbors are used.
`d`	set the distance of neighboring points if *method* = "distance". Default = 1.
`k`	set the number of neighboring points if *method* = "k-nearest". Default = 5.
`reconstruct.all`	logical. If *TRUE* the entire ground surface is reconstructed. If FALSE only the ground portions with missing points are reconstructed

Value

a data.frame containing the xyz coordinates plus the class of the reconstructed points

See Also

`reconstruct.ground.par` for more effecient computation.

Examples

```
library(viewshed3d)
data(TLSclass)
ground=subset(TLSclass,TLSclass[,4]=='ground')
reconstructed=reconstruct.ground(ground,grid.res = 0.2)

library(rgl)
open3d()
plot3d(reconstructed,add=TRUE)
```

`reconstruct.ground.par` *Reconstruct the missing portions of a TLS scene's ground*

Description

The **reconstruct.ground.par** function is similar to the `reconstruct.ground` function with parallel computation capabilities.

Usage

```
reconstruct.ground.par(data, grid.res, scene.radius, scene.center, shape,
  method, d, k, reconstruct.all, ncores)
```

Arguments

`data`	a data.frame containing the xyz and point class of ground points segmented with the `class.ground` and optionally with the `ground.filter` functions
`grid.res`	the resolution of a grid used to locate the new points to add. Default = 0.05
`scene.radius`	optional. Define scene radius relative to the 0,0,0 coordinate. Should be used after the scene was reshaped with the `reshape.scene function`
`scene.center`	a vector containing the xyz coordinates of the scene center for reshaping
`shape`	the shape of the scene radius reshaping (see `reshape.scene` function for more details)
`method`	the method to find the neighboring points. If = "distance" all the points located within a distance *d* are used to estimate *z*. If = "k-nearest" only the *k* nearest neighbors are used.
`d`	set the distance of neighboring points if *method* = "distance". Default = 1.
`k`	set the number of neighboring points if *method* = "k-nearest". Default = 5.
`reconstruct.all`	logical. If *TRUE* the entire ground surface is reconstructed. If FALSE only the ground portions with missing points are reconstructed
`ncores`	the number of cores to use for parallel computation. Default = N detected cores-1

Value

a data.frame containing the xyz coordinates plus the class of the reconstructed points

---

`reshape.scene` *Reshape a scene by applying a cut off distance and recentering a TLS scene*

Description

The **reshape.scene** function enables reshaping a TLS scene at any time in the data preparation process. It enables application of a cut off distance and recentering a TLS scene by providing a user defined scene center, or by allowing users to interactively select a region of the point cloud to be used as the scene center.

Usage

```
reshape.scene(data, point, radius, shape, plot3d)
```

Arguments

`data`	a data.frame containing the xyz coordinates of a TLS scene
`point`	(optional) a vector containing the xyz coordinates of the scene center. If no point is provided the user had to interactively select a region of the point cloud
`radius`	the scene radius. Default = 5
`shape`	If = "2d" the cut off distance is applied on the x and y axes of the scene and the scene edge is a cylinder. If = "3d" the cut off distance is applied on the three dimensions of the scene and the scene edge is a sphere. Default = "2d"
`plot3d`	logical. If *TRUE* a 3d plot of the result is provided.

Value

a dataframe containing the reshaped TLS scene

Examples

```
library(viewshed3d)
data(TLSclass)

# when a point is provided, the scene is automatically reshaped
center=c(mean(TLSclass[,1]),mean(TLSclass[,2]),mean(TLSclass[,3]))
reshaped=reshape.scene(TLSclass,point = center,radius = 2, plot3d = TRUE)

# if no point is provided, the user has to select a region (draw a squared region with the
# right click) of the point cloud that will be used as the scene center
reshaped=reshape.scene(TLSclass,radius = 2, plot3d = TRUE)
```

TLSclass — *TLS scene of a plantation plot after ground and vegetation segmentation*

Description

TLS scene of a plantation plot after ground and vegetation segmentation

TLSrecons — *TLS scene of a plantation plot after ground reconstruction*

Description

TLS scene of a plantation plot after ground reconstruction

TLSscene — *TLS scene of a plantation plot*

Description

TLS scene of a plantation plot

view — *Compute visibility in 3D TLS scene*

Description

The **view** function explores a TLS point cloud in all directions of the 3D space and records the nearest point in each direction. A single direction is thus assumed to be a sligthline that ends as soon as an object is encountered (see package description for more details). The view function requires a parameter data.frame produced with the `view.param` function, and the angle of a single slightline is thus defined at this step. The visibility is computed from a user defiend location.

Usage

```
view(data, param, location, plot3d, plot.result)
```

Arguments

data	a data.frame containing the xyz coordinates of a TLS point cloud
param	a data.frame containing the directions that will be used to explore the point cloud
location	a vector containing the xyz coordinates of the point representing the animal location. Default is 0,0,0.
plot3d	logical. If *TRUE* a 3D view of the point cloud exploration is plotted
plot.result	logical. If *TRUE* the % of visibility vs. distance is ploted

Value

a list containing the remaining visibility as function of the distance from the scene center ($visibility) and the 3D point cloud of the portion of data seen from the scene center ($points).

Note

some existing parameters are already provided with the viewshed3d package with parameters for an angular resolution ranging from 1 to 10 : param_1, param_2, param_3, param_4, param_5, param_6, param_7, param_8, param_9, param_10.

Examples

```
library(viewshed3d)
data(TLSrecons)
data(param_10)

center=c(mean(TLSrecons[,1]),mean(TLSrecons[,2]),mean(TLSrecons[,3]))

view.data=view(TLSrecons,param = param_1,location = center, plot3d = TRUE)
```

---

view.param *Computing parameters for the view function*

Description

The **view.param** function computes the parameters for the view function. Each line of the output is a direction corresponding to a single slightline. The calibration is done by optimizing the filling of the surface of a sphere with circles, each direction being recorded as the 3D polar coordinates of the circle center.

Usage

```
view.param(angular.res, method)
```

Arguments

angular.res	the angular resolution of slightline that will be used in the view funtion. Default = 10.
method	in some cases, when no space is available to insert a new circle, a hole might remain after the sphere surface filling. Two options are available to correct this issue. If *method = 'fill'* the gap is filled with a portion of circle and a new diection is added but its angle with the neighboring direction is lower than *angular.res*. If *method = 'regularize* (the default value) the directions are corrected so that they all have a similar angle between them. If *method = 'n'* no action is taken.

Value

a data.frame containing the parameters to use as input in the view function.

Note

some existing parameters are already provided with the viewshed3d package with parameters for an angular resolution ranging from 1 to 10 : param_1, param_2, param_3, param_4, param_5, param_6, param_7, param_8, param_9, param_10.

Examples

```
library(viewshed3d)

#- anglura.res = 40 is way to big. Just good for the example
param=view.param(angular.res = 40)

head(param)
```

Index

∗Topic **3d point cloud**
viewshed3d-package, 2
∗Topic **T-LiDAR**
viewshed3d-package, 2
∗Topic **Terrestrial Laser Scanner**
viewshed3d-package, 2
∗Topic **Viewshed**
viewshed3d-package, 2
∗Topic **data**
param_1, 6
param_10, 6
param_2, 6
param_3, 6
param_4, 6
param_5, 7
param_6, 7
param_7, 7
param_8, 7
param_9, 7
TLSclass, 11
TLSrecons, 11
TLSscene, 11
∗Topic **ground vegetation segmentation**
viewshed3d-package, 2
∗Topic **package**
viewshed3d-package, 2

class.ground, *2*, 3, *5*, *8*, *9*
cum.view, *2*, 4

ground.filter, *2*, 5, *8*, *9*

param_1, 6, *12*, *13*
param_10, 6, *12*, *13*
param_2, 6, *12*, *13*
param_3, 6, *12*, *13*
param_4, 6, *12*, *13*
param_5, 7, *12*, *13*
param_6, 7, *12*, *13*
param_7, 7, *12*, *13*
param_8, 7, *12*, *13*
param_9, 7, *12*, *13*

reconstruct.ground, *2*, 8, *9*
reconstruct.ground.par, *2*, *8*, 9
reshape.scene, *2*, *8*, *9*, 10

TLSclass, 11
TLSrecons, 11
TLSscene, 11

view, *2*, *6*, *7*, 11, *12*, *13*
view.param, *2*, *11*, 12
viewshed3d *(*viewshed3d-package*)*, 2
viewshed3d-package, 2