Package: nat (via r-universe)

September 16, 2024

Type Package

Title NeuroAnatomy Toolbox for Analysis of 3D Image Data

Version 1.10.4

URL https://github.com/natverse/nat, https://natverse.org/

BugReports https://github.com/natverse/nat/issues

Description NeuroAnatomy Toolbox (nat) enables analysis and visualisation of 3D biological image data, especially traced neurons. Reads and writes 3D images in NRRD and 'Amira' AmiraMesh formats and reads surfaces in 'Amira' hxsurf format. Traced neurons can be imported from and written to SWC and 'Amira' LineSet and SkeletonGraph formats. These data can then be visualised in 3D via 'rgl', manipulated including applying calculated registrations, e.g. using the 'CMTK' registration suite, and analysed. There is also a simple representation for neurons that have been subjected to 3D skeletonisation but not formally traced; this allows morphological comparison between neurons including searches and clustering (via the 'nat.nblast' extension package).

Depends R (>= 2.15.1), rgl (>= 0.98.1)

Imports nabor, igraph (>= 0.7.1), methods, filehash (>= 2.3), digest, nat.utils (>= 0.4.2), plyr, yaml, progress, checkmate, stringr, pbapply, zip

Suggests spelling, Rvcg (>= 0.17), testthat, httr, XML, knitr, rmarkdown, MASS, alphashape3d, Morpho, plotly, readobj, brotli, qs, jsonlite

Enhances natcpp

License GPL-3

LazyData yes

Collate 'alphashape3d.R' 'amiralandmarks-io.R' 'amiramesh-io.R' 'brotli.R' 'cmtk-reformat.R' 'cmtk.R' 'cmtk_geometry.R' 'cmtk_io.R' 'cmtkreg.R' 'coordinates.R' 'dist3D_Segment_to_Segment.R' 'neuron.R' 'dotprops.R'

'geometry.R' 'graph-nodes.R' 'hxsurf.R' 'im3d.R'
'interactive.R' 'morphometry.R' 'nat-data.R' 'nat-package.R'
'ndigest.R' 'neuron-io-amira.R' 'neuron-io-fiji.R'
'neuron-io-neuroml.R' 'neuron-io.R' 'neuron-mesh.R'
'neuron-plot.R' 'neuronlist.R' 'neuronlist_interactive_3d.R'
'neuronlist_sets.R' 'neuronlistfh.R' 'neuronlistz.R' 'ngraph.R'
'nrrd-io.R' 'pop3d.R' 'potential_synapses.R' 'qs.R' 'reglist.R'
'seglist.R' 'summary.R' 'utils.R' 'vaa3draw-io.R' 'vtk-io.R'
'wire3d.R' 'xform.R' 'xformimage.R' 'xformpoints.R'
'xyzmatrix.R' 'zzz.R'

RoxygenNote 7.2.3

Encoding UTF-8

VignetteBuilder knitr

Language en-GB

Repository https://natverse.r-universe.dev

RemoteUrl https://github.com/natverse/nat

RemoteRef HEAD

RemoteSha 2538b4c5b8e2268da56765ed79f0b38206bbc82a

Contents

---

nat-package *Analyse 3D biological image data especially neurons*

Description

nat provides tools to read, analyse, plot, transform and convert neuroanatomical data, especially representations of neurons.

Neuron Objects

At present there are 2 main representations of neuronal data:

`neuron` objects contain one or more connected trees that make up a neuron

`dotprops` objects can contain one (or more) neurons represented as points and tangent vectors in which the connectivity information has been discarded

The `subset` function has both `subset.neuron` and `subset.dotprops` methods, which can be used to keep (or reject) specified vertices within a neuron e.g. by spatial constraints. `subset.neuron` will look after the tree structure of neurons in these circumstances.

`neuron` objects containing connected trees can be converted to `ngraph` objects, a lightweight wrapper around the `igraph` library's `graph` class that preserves 3D coordinate information. This allows neurons to be manipulated based on their graph structure, e.g. by finding all nodes upstream (closer to the root) or downstream of a given node. The `as.neuron` function can convert `ngraph` objects back to `neuron`s or selected vertex indices can be used to subset a neuron with `subset.neuron`.

Collections of Neurons

Neurons can be collected as `neuronlist` objects, which contain multiple `neuron` or `dotprops` objects along with an attached dataframe of metadata. The metadata can be accessed and manipulated using the `myneuronlist[i,j]` notation (see `neuronlist-dataframe-methods`).

Neurons can be read in to a neuronlist using `read.neurons` or written out using `write.neurons` with support for many of the most common formats including swc.

Metadata can be used to colour or subset the neurons during plotting (see `plot3d.neuronlist` and `subset.neuronlist`). Interactive 3D selection of neurons in a neuronlist is also possible using `find.neuron` (which makes use of `rgl`'s `select3d` function.

`neuronlist` objects also provide additional functionality to streamline arithmetic (e.g. scaling all the points in all neurons see `*.neuronlist`) and transformations (see **Transformations** section below and `xform`). Arbitrary functions can be applied to each individual neuron using the `nlapply` function, which also provides options for progress bars and simple parallelisation.

Transformations

neuron or dotprops objects can be transformed from e.g. sample to template brain space using affine or non-rigid registrations. **nat** has built in support for registrations calculated with the open source CMTK package available at www.nitrc.org/projects/cmtk. See ?cmtk for installation details. The function xform has methods to deal with a variety of types of interest.

3D Image Data

In addition to data types defined by unstructured collections of 3D vertices such as neuron, dotprops and hxsurf objects nat provides the im3d class to handle image/density data on a regular grid. I/O is handled by read.im3d and write.im3d, which are currently implemented for the AmiraMesh and NRRD file formats; there is also read only access to the vaa3d raw format.

Spatial information can be queried with voxdims, boundingbox and ijkpos, xyzpos methods. You can convert between voxel data and coordinate (vertex) -based representations using the following functions:

- as.im3d The as.im3d.matrix method converts XYZ coordinates to an im3d image volume
- ind2coord Find XYZ coordinates of specified voxels of an im3d image volume
- dotprops The dotprops.im3d method converts an im3d object to a dotprops format neuron, i.e. a cloud of unconnected segments.

Surface Data

nat can read, write, transform and subset surface (mesh) objects defined by Amira's HxSurface class. See read.hxsurf and links therein. In addition hxsurf objects can be converted to the mesh3d format, which provides a link to the rgl package and also to packages for morphometrics and sophisticated mesh manipulation such as Morpho and Rvcg.

rgl Package

nat uses the rgl package extensively for 3D visualisation. rgl's core function is to provide interactive visualisation (usually in an X11 window depending on OpenGL - and therefore on a graphics card or OpenGL software emulator) but recently significant functionality for static snapshots and embedding results in reports such as web pages has been added. With this in mind, Duncan Murdoch has added the rgl.useNULL option. As of nat 1.8.0, options(rgl.useNULL=TRUE) will be set before nat is loaded in non-interactive R sessions. If you want to use nat in interactive environments where X11 is not available, you may want to set options(rgl.useNULL=TRUE) manually before loading nat.

plotly Package

Since **nat** v1.9.2 there is support for plotly as an alternative 3D visualisation engine. You can set option(nat.plotengine='plotly') to make this the default. See the 3D graphics vignette for further details.

File Formats

nat supports multiple input and output data formats for the object classes. There is a registry-based mechanism which allows support for reading or writing specific file formats (see `fileformats`) to be plugged in to reasonably generic functions such as `read.neurons`. It is perfectly possible for other R packages or end users to extend the supported list of file types by registering new read/write or identification functions.

Package Options

The following options can be set to specify default behaviour.

`nat.cmtk.bindir` Location of CMTK binaries. See `cmtk.bindir`

`nat.default.neuronlist` A character string naming a `neuronlist` to use with the `plot3d.character` method

`nat.plotengine` A character string naming a plotengine to use either 'rgl' or 'plotly'. rgl is the default if unset, see `3D Graphics` vignette for details.

`nat.progress` The default progress reporter to use with `nlapply`. See `create_progress_bar` for possible values. When unset is equivalent to special value `'auto'`. To suppress altogether, use `nat.progress="none"`.

`nat.use_natcpp` Whether or not to use the natcpp package (if available) to accelerate some basic neuron processing functions. Set to `FALSE` if you don't want this to happen.

In addition there is one read-only option:

- `nat.cmtk.version` which is used to store the current cmtk version when there are repeated calls to `cmtk.version`.

See Also

`neuron`, `dotprops`, `neuronlist`, `nlapply`, `plot3d`, `xform`, `im3d`, `read.hxsurf`, `rgl` which is used for visualisation, `fileformats`, `read.neurons`, `cmtk`.

`*.neuronlist` *Arithmetic for neuron coordinates applied to neuronlists*

Description

If x is one number or 3-vector, multiply coordinates by that If x is a 4-vector, multiply xyz and diameter TODO Figure out how to document arithmetic functions in one go

Usage

```
## S3 method for class 'neuronlist'
x * y

## S3 method for class 'neuronlist'
x + y

## S3 method for class 'neuronlist'
x - y

## S3 method for class 'neuronlist'
x / y
```

Arguments

x	a neuronlist
y	(a numeric vector to multiply coords in neuronlist members)

Value

modified neuronlist

See Also

Other neuronlist: `is.neuronlist()`, `neuronlist-dataframe-methods`, `neuronlistfh()`, `neuronlistz()`, `neuronlist()`, `nlapply()`, `read.neurons()`, `write.neurons()`

Examples

```
mn2<-Cell07PNs[1:10]*2
```

---

affmat2cmtkparams *Decompose homogeneous affine matrix to CMTK registration parameters*

Description

Decompose homogeneous affine matrix to CMTK registration parameters

Usage

```
affmat2cmtkparams(matrix, centre = c(0, 0, 0))
```

Arguments

matrix	4x4 homogeneous affine matrix
centre	Rotation centre

Details

The version attribute of the resultant matrix marks this as compliant with CMTK>v2.4 (~ Dec 2013) when a bug in affine matrix (de)composition was fixed.

Value

5x3 matrix of CMTK registration parameters with a version attribute

See Also

Other cmtk-geometry: `cmtk.dof2mat()`, `cmtk.mat2dof()`, `cmtkparams2affmat()`

`all.equal.dotprops` *all.equal method tailored to dotprops objects*

Description

all.equal method tailored to dotprops objects

Usage

```
## S3 method for class 'equal.dotprops'
all(target, current, check.attributes = FALSE, absoluteVectors = TRUE, ...)
```

Arguments

`target, current`	dotprops objects to compare
`check.attributes`	Whether to check attributes (false by default)
`absoluteVectors`	Whether to check only the absolute value of eigenvectors for equality (default TRUE, see details)
`...`	Additional arguments passed to base `all.equal`.

Details

This method is required because the direction vectors are computed using an eigenvector decomposition where the sign of the eigenvector is essentially random and subject to small numerical instabilities. Therefore it does not usually make sense to check the value of vect exactly.

Examples

```
# equal using default
kc1=kcs20[[1]]
kc1.recalc=dotprops(kc1)
# not equal due to differences in attributes and vectors
all.equal.default(kc1.recalc, kc1)
# still not equal because of tangent vector flipping
all.equal.default(kc1.recalc, kc1, check.attributes=FALSE)
# equal using appropriate method
stopifnot(isTRUE(all.equal(kc1.recalc, kc1)))
# NB identical when recalculated on same setup from same data
stopifnot(isTRUE(all.equal.default(kc1.recalc, dotprops(kc1))))
```

`all.equal.im3d` *Check equality on data and key attributes of im3d objects*

Description

Check equality on data and key attributes of im3d objects

Usage

```
## S3 method for class 'equal.im3d'
all(
  target,
  current,
  tolerance = 1e-06,
  attrsToCheck = c("BoundingBox"),
  attrsToCheckIfPresent = c("dim", "names", "dimnames", "x", "y", "z"),
  CheckSharedAttrsOnly = FALSE,
  ...
)
```

Arguments

`target`	R object.
`current`	other R object, to be compared with `target`.
`tolerance`	numeric ≥ 0. Differences smaller than `tolerance` are not reported. The default value is close to `1.5e-8`.
`attrsToCheck`	Which attributes in im3d should always be checked
`attrsToCheckIfPresent`	Which attributes in im3d should be checked if present
`CheckSharedAttrsOnly`	Logical whether to check shared attributes only (default: FALSE)
`...`	additional arguments passed to `all.equal`

See Also

all.equal

---

all.equal.neuron *Check equality on key fields of neuron object*

Description

Check equality on key fields of neuron object

Usage

```
## S3 method for class 'equal.neuron'
all(
  target,
  current,
  tolerance = 1e-06,
  check.attributes = FALSE,
 fieldsToCheck = c("NumPoints", "StartPoint", "BranchPoints", "EndPoints", "NumSegs",
    "SegList", "d"),
  fieldsToCheckIfPresent = c("NeuronName", "nTrees", "SubTrees"),
  fieldsToExclude = character(),
  CheckSharedFieldsOnly = FALSE,
  ...
)
```

Arguments

target	R object.
current	other R object, to be compared with target.
tolerance	numeric ≥ 0. Differences smaller than tolerance are not reported. The default value is close to 1.5e-8.
check.attributes	logical indicating if the attributes of target and current (other than the names) should be compared.
fieldsToCheck	Which fields in the neuron are always checked. The special value of NA indicates that **all** fields in the neurons will be compared.
fieldsToCheckIfPresent	These fields are only checked if they are present
fieldsToExclude	Character vector of fields to exclude from check
CheckSharedFieldsOnly	Logical whether to check shared fields only (default: FALSE)
...	additional arguments passed to all.equal

See Also

all.equal

Examples

```
x=Cell07PNs[[1]]
y=x
y$NeuronName='rhubarb'
# NOT TRUE
all.equal(x, y)
# TRUE
all.equal(x, y, fieldsToExclude='NeuronName')
```

---

amiratype *Return the type of an AmiraMesh file on disk or a parsed header*

Description

Return the type of an AmiraMesh file on disk or a parsed header

Usage

```
amiratype(x, bytes = NULL)
```

Arguments

x	Path to files on disk or a single pre-parsed parameter list
bytes	A raw vector containing at least 11 bytes from the start of the file.

Details

Note that when checking a file we first test if it is an AmiraMesh file (fast, especially when bytes!=NULL) before reading the header and determining content type (slow).

Value

character vector (NA_character_ when file invalid)

See Also

Other amira: is.amiramesh(), read.amiramesh(), read.hxsurf(), write.hxsurf()

`as.data.frame.neuronlist` *Get or set the attached data.frame of a neuronlist*

Description

For `as.data.frame`, when there is no attached data.frame the result will be a data.frame with 0 columns but an appropriate number of rows, named by the objects in the neuronlist.

`data.frame<-` methods set the data frame attached to an object. At present this is only used for neuronlist objects.

Usage

```
## S3 method for class 'neuronlist'
as.data.frame(x, row.names = names(x), optional = FALSE, ...)

data.frame(x) <- value

## S3 replacement method for class 'neuronlist'
data.frame(x) <- value
```

Arguments

`x`	neuronlist to convert
`row.names`	row names (defaults to names of objects in neuronlist, which is nearly always what you want.)
`optional`	ignored in this method
`...`	additional arguments passed to `data.frame` (see examples)
`value`	The new data.frame to be attached to `x`

Value

for `as.data.frame.neuronlist`, a `data.frame` with length(x) rows, named according to names(x) and containing the columns from the attached data.frame, when present.

for `data.frame<-.neuronlist`, a neuronlist with the attached data.frame.

See Also

`data.frame`, `neuronlist`

Examples

```
head(as.data.frame(kcs20))

# add additional variables
str(as.data.frame(kcs20, i=seq(kcs20), abc=LETTERS[seq(kcs20)]))
# stop character columns being turned into factors
newdf <- as.data.frame(kcs20, i=seq(kcs20), abc=LETTERS[seq(kcs20)],
  stringsAsFactors=FALSE)
str(newdf)
data.frame(kcs20)=newdf
```

---

as.hxsurf *Convert an object to a nat hxsurf object*

Description

Convert an object to a nat hxsurf object

Usage

```
as.hxsurf(x, ...)

## S3 method for class 'mesh3d'
as.hxsurf(x, region = "Interior", col = NULL, ...)
```

Arguments

x	A surface object
...	Additional arguments passed to methods
region	The default name for the surface region
col	The surface colour (default value of NULL implies the colour specified in mesh3d object or grey when the mesh3d object has no colour.)

Details

hxsurf objects are based on the format of Amira's surface objects (see read.hxsurf). They have the ability to include multiple distinct regions. However, at the moment the only method that we provide converts mesh3d objects, which can only include one region.

Value

A new surface object of class hxsurf (see read.hxsurf) for details.

See Also

as.mesh3d

Other hxsurf: as.mesh3d(), materials(), plot3d.hxsurf(), read.hxsurf(), subset.hxsurf(), write.hxsurf()

Examples

```
tet=tetrahedron3d(col='red')
teth=as.hxsurf(tet)

plot3d(teth)
```

`as.im3d` *Convert a suitable object to an im3d object.*

Description

Convert a suitable object to an im3d object.

Usage

```
as.im3d(x, ...)

## S3 method for class 'im3d'
as.im3d(x, ...)

## S3 method for class 'matrix'
as.im3d(x, voxdims, origin = NULL, BoundingBox = NULL, ...)
```

Arguments

`x`	Object to turn into an im3d
`...`	Additional arguments to pass to methods.
`voxdims`	Numeric vector of length 3 *or* an `im3d` compatible object (see details) completely specifying the required space.
`origin`	the location (or centre) of the first voxel
`BoundingBox`	Physical extent of image. See the details section of `boundingbox`'s help for more.

Details

At present the only interesting method in `nat` is `as.im3d.matrix` which can be used to convert a matrix of 3D points into a 3D volume representation. `ind2coord` can be used to do the reverse: convert a set of 3D coords to an `im3d` volume.

Other than that, this is a largely a placeholder function with the expectation that other packages may wish to provide suitable methods.

`as.im3d.matrix` can accept any object that can be converted to an im3d object in the `voxdims` argument This will completely specify the dims, voxdims, origin etc. Any value passed to those parameters will be ignored. This can be useful for producing a new im3d to match a target image on disk or a `nat.templatebrains::templatebrain` object. See examples.

See Also

im3d, ind2coord

im3d, as.im3d

Other im3d: boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
## convert a list of neurons into an image volume
im=as.im3d(xyzmatrix(kcs20), voxdims=c(1, 1, 1),
  BoundingBox=c(250, 410, 0, 130, 0, 120))
## Not run:
write.im3d(im, 'kc20volume.nrrd')

## use image dimensions of an image on disk
# nb note use of ReadData = FALSE so that we just fetch the dimensions of
# the target image
diskim=read.im3d("/path/to/my/image.nrrd", ReadData = FALSE)
im=as.im3d(xyzmatrix(kcs20), diskim)

## use image dimensions of JFRC2 template brain to define the image space
library(nat.flybrains)
im=as.im3d(xyzmatrix(kcs20), JFRC2)

## End(Not run)
```

---

as.mesh3d *Convert an object to an rgl mesh3d*

Description

as.mesh3d.ashape3d converts an alphashape3d::ashape3d object into a nat/rgl compatible mesh3d surface

Note that this provides a link to the Rvcg package

as.mesh3d.boundingbox converts a nat boundingbox object into an rgl compatible mesh3d object.

Usage

```
## S3 method for class 'ashape3d'
as.mesh3d(x, tri_to_keep = 2L, ...)

## S3 method for class 'hxsurf'
as.mesh3d(x, Regions = NULL, material = NULL, drop = TRUE, ...)

## S3 method for class 'boundingbox'
as.mesh3d(x, ...)
```

Arguments

x	Object to convert to mesh3d
tri_to_keep	Which alphashape triangles to keep (expert use only - see triang entry in **Value** section of ashape3d docs for details.)
...	Additional arguments for methods
Regions	Character vector or regions to select from hxsurf object
material	rgl materials such as color
drop	Whether to drop unused vertices (default TRUE)

Details

An alpha shape is a generalisation of a convex hull enclosing a set of points. Unlike a convex hull, the resultant surface can be partly concave allowing the surface to more closely follow the set of points.

In this implementation, the parameter alpha is a scale factor with units of length that defines a spatial domain. When alpha is larger the alpha shape approaches the convex hull; when alpha is smaller the alpha shape has a greater number of faces / vertices i.e. it follows the points more closely.

Value

a mesh3d object which can be plotted and manipulated using rgl and nat packages.

See Also

ashape3d, mesh3d

as.mesh3d, tmesh3d, as.hxsurf, read.hxsurf

Other hxsurf: as.hxsurf(), materials(), plot3d.hxsurf(), read.hxsurf(), subset.hxsurf(), write.hxsurf()

Examples

```
library(alphashape3d)
kcs20.a=ashape3d(xyzmatrix(kcs20), alpha = 10)
plot(kcs20.a)

# convert to mesh3d
kcs20.mesh=as.mesh3d(kcs20.a)

# check that all points are inside mesh
all(pointsinside(kcs20, kcs20.mesh))
# and show that we can also use the alphashape directly
all(pointsinside(kcs20, kcs20.a))

nclear3d()
wire3d(kcs20.mesh)
plot3d(kcs20, col=type, lwd=2)

bb=boundingbox(kcs20)
```

```
mbb=as.mesh3d(bb)

plot3d(kcs20)
# simple plot
plot3d(bb)
shade3d(mbb, col='red', alpha=0.3)
```

`as.neuronlist` *Make a list of neurons that can be used for coordinate plotting/analysis*

Description

Make a list of neurons that can be used for coordinate plotting/analysis

Usage

```
as.neuronlist(l, ...)

## Default S3 method:
as.neuronlist(l, df = NULL, AddClassToNeurons = TRUE, ...)
```

Arguments

`l`	An existing list or a single neuron to start a list
`...`	Additional arguments passed to methods
`df`	the data.frame to attach with additional metadata.
`AddClassToNeurons`	Whether to ensure neurons have class `neuron` (see details).

Details

Note that `as.neuronlist` can cope with both `neurons` and `dotprops` objects but `AddClassToNeurons` will only apply to things that look like neurons but don't have a class of `neuron`.

See `neuronlist` details for more information.

Value

neuronlist with `attr('df')`

See Also

`is.neuronlist`,`is.neuron`,`is.dotprops`

as.neuronlist.neuronlistfh

convert neuronlistfh to a regular (in memory) neuronlist

Description

convert neuronlistfh to a regular (in memory) neuronlist

Usage

```
## S3 method for class 'neuronlistfh'
as.neuronlist(l, ...)
```

Arguments

l	An existing list or a single neuron to start a list
...	Additional arguments passed to methods

boundingbox

Get the bounding box of an image volume or object containing 3D vertices

Description

Set the bounding box of an im3d object

Usage

```
boundingbox(x, ...)

## S3 method for class 'im3d'
boundingbox(x, dims = dim(x), ...)

## S3 method for class 'character'
boundingbox(x, ...)

## Default S3 method:
boundingbox(x, na.rm = FALSE, ...)

boundingbox(x) <- value
```

Arguments

x	an im3d object or any object for which xyzmatrix can extract 3D points (e.g. neurons, surfaces etc), or, for boundingbox.character, a character vector specifying a file.
...	Additional arguments passed to methods, and eventually to makeboundingbox
dims	The number of voxels in each dimension when x is a BoundingBox matrix.
na.rm	Whether to ignore NA points (default FALSE)
value	The object which will provide the new boundingbox information. This can be be either an im3d object with a boundingbox or a vector or matrix defined according to boundingbox.default.

Details

The bounding box is defined as the position of the voxels at the two opposite corners of the cuboid encompassing an image, *when each voxel is assumed to have a single position (sometimes thought of as its centre)* **and no physical extent.** When written as a vector it should look like: c(x0,x1,y0,y1,z0,z1). When written as a matrix it should look like: rbind(c(x0,y0,z0),c(x1,y1,z1)) where x0,y0,z0 is the position of the origin.

Note that there are two competing definitions for the physical extent of an image that are discussed e.g. https://teem.sourceforge.net/nrrd/format.html. The definition that makes most sense depends largely on whether you think of a pixel as a little square with some defined area (and therefore a voxel as a cube with some defined volume) *or* you take the view that you can only define with certainty the grid points at which image data was acquired. The first view implies a physical extent which we call the bounds=dim(x) * c(dx,dy,dz); the second is defined as BoundingBox=dim(x)-1 * c(dx,dy,dz) and assumes that the extent of the image is defined by a cuboid including the sample points at the extreme corner of the grid. Amira takes this second view and this is the one we favour given our background in microscopy. If you wish to convert a bounds type definition into an im3d BoundingBox, you should pass the argument input='bounds'.

boundingbox.default is designed to be used on objects that contain 3D point information. This includes any object for which an xyzmatrix method is defined including matrix or data.frame objects describing 3D points as well as specialised classes such as neuron, neuronlist, rgl mesh3d objects.

Value

a matrix with 2 rows and 3 columns with class='boundingbox' or *NULL* when missing.

See Also

makeboundingbox, plot3d.boundingbox

Other im3d: as.im3d(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
# bounding box for a neuron
boundingbox(Cell07PNs[[1]])
```

c.hxsurf *Concatenate HyperSurface objects*

Description

Concatenate HyperSurface objects

Usage

```
## S3 method for class 'hxsurf'
c(...)
```

Arguments

...	multiple hxsurf objects

Value

new hxsurf

Examples

```
h1 = as.hxsurf(icosahedron3d(), 'a')
h2 = as.hxsurf(tetrahedron3d()+1, 'b')
h3 = as.hxsurf(icosahedron3d()+3, 'c')
hc = c(h1, h2, h3)
```

c.neuronlist *Combine multiple neuronlists into a single list*

Description

Combine multiple neuronlists into a single list

Usage

```
## S3 method for class 'neuronlist'
c(..., recursive = FALSE)
```

Arguments

...	neuronlists to combine
recursive	Presently ignored

Details

Uses rbind.fill to join any attached dataframes, so missing values are replaced with NAs.

See Also

c

Examples

```
stopifnot(all.equal(kcs20[1:2],c(kcs20[1],kcs20[2])))
```

Cell07PNs *Cell07PNs: 40 Sample Projection Neurons from Jefferis, Potter et al 2007*

Description

These R lists (which have additional class neuronlist) contain 40 traced olfactory projection neurons from Jefferis, Potter et al 2007 that have been transformed onto the IS2 template brain (Cachero, Ostrovsky et al 2010).

References

Jefferis G.S.X.E., Potter C.J., Chan A.M., Marin E.C., Rohlfing T., Maurer C.R.J., and Luo L. (2007). Comprehensive maps of Drosophila higher olfactory centers: spatially segregated fruit and pheromone representation. Cell 128 (6), 1187–1203. doi:10.1016/j.cell.2007.01.040

Cachero S., Ostrovsky A.D., Yu J.Y., Dickson B.J., and Jefferis G.S.X.E. (2010). Sexual dimorphism in the fly brain. Curr Biol 20 (18), 1589–601. doi:10.1016/j.cub.2010.07.045

See Also

head.neuronlist, with.neuronlist, dl1neuron

Other nat-data: MBL.surf, kcs20

Examples

```
head(Cell07PNs)
table(with(Cell07PNs,Glomerulus))
```

clampmax *Return function that finds maximum of its inputs within a clamping range*

Description

Return function that finds maximum of its inputs within a clamping range

Usage

```
clampmax(xmin, xmax, replace.infinite = NA_real_)
```

Arguments

xmin, xmax	clamping range. If xmax is missing xmin should be a vector of length 2.
replace.infinite	The value with which to replace non-finite values *in the input vector*. When codereplace.infinite=FALSE no action is taken. The default value of NA will result in e.g. Inf being mapped to NA.

Details

Note that by default infinite values in the input vector are converted to NAs before the being compared with the clampmax range.

Value

A function with signature f(x, ..., na.rm)

Examples

```
## Not run:
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd',package='nat'))
d=unmask(rnorm(sum(LHMask),mean=5,sd=5),LHMask)
op=par(mfrow=c(1,2))
rval=image(projection(d,projfun=max))
image(projection(d,projfun=clampmax(0,10)),zlim=rval$zlim)
par(op)

## End(Not run)
```

cmtk.bindir *Return path to directory containing CMTK binaries*

Description

The Computational Morphometry Toolkit (CMTK) is the default image registration toolkit supported by nat. An external CMTK installation is required in order to apply CMTK registrations. This function attempts to locate the full path to the CMTK executable files and can query and set an option.

Usage

```
cmtk.bindir(
  firstdir = getOption("nat.cmtk.bindir"),
 extradirs = c("~/bin", "/usr/local/lib/cmtk/bin", "/usr/local/bin", "/opt/local/bin",
    "/opt/local/lib/cmtk/bin/", "/Applications/IGSRegistrationTools/bin",
    "C:\\cygwin64\\usr\\local\\lib\\cmtk\\bin",
    "C:\\Program Files\\CMTK-3.3\\CMTK\\lib\\cmtk\\bin"),
  set = FALSE,
  check = FALSE,
  cmtktool = "gregxform"
)
```

Arguments

firstdir	Character vector specifying path containing CMTK binaries or NA (see details). This defaults to options('nat.cmtk.bindir').
extradirs	Where to look if CMTK is not in firstdir or the PATH
set	Whether to set options('nat.cmtk.bindir') with the found directory. Also check/sets cygwin path on Windows (see Installation section).
check	Whether to (re)check that a path that has been set appropriately in options(nat.cmtk.bindir='/some/path') or now found in the PATH or alternative directories. Will throw an error on failure.
cmtktool	Name of a specific cmtk tool which will be used to identify the location of all cmtk binaries.

Details

Queries options('nat.cmtk.bindir') if firstdir is not specified. If that does not contain the appropriate binaries, it will look in the system PATH for the cmtk wrapper script installed by most recent cmtk installations.

Failing that, it will look for the cmtk tool specified by cmtktool, first in the path and then a succession of plausible places until it finds something. Setting options(nat.cmtk.bindir=NA) or passing firstdir=NA will stop the function from trying to locate CMTK, always returning NULL unless check=TRUE, in which case it will error out.

Value

Character vector giving path to CMTK binary directory or NULL when this cannot be found.

Installation

It is recommended to install released CMTK versions available from the NITRC website. A bug in composition of affine transformations from CMTK parameters in the CMTK versions <2.4 series means that CMTK>=3.0 is strongly recommended. CMTK v3 registrations are not backwards compatible with CMTK v2, but CMTK v3 can correctly interpret and convert registrations from earlier versions.

On Windows, when `set=TRUE`, cmtk.bindir will also check that the cygwin bin directory is in the PATH. If it is not, then it is added for the current R session. This should solve issues with missing cygwin DLLs.

See Also

`options`

Examples

```
message(ifelse(is.null(d<-cmtk.bindir()), "CMTK not found!",
               paste("CMTK is at:",d)))
## Not run:
# set options('nat.cmtk.bindir') according to where cmtk was found
op=options(nat.cmtk.bindir=NULL)
cmtk.bindir(set=TRUE)
options(op)
## End(Not run)
```

---

cmtk.call *Utility function to create and run calls to CMTK command line tools*

Description

`cmtk.call` processes arguments into a form compatible with CMTK command line tools.

`cmtk.system2` actually calls a cmtk tool using a call list produced by `cmtk.call`

Usage

```
cmtk.call(
  tool,
  PROCESSED.ARGS = NULL,
  ...,
  FINAL.ARGS = NULL,
  RETURN.TYPE = c("string", "list")
)

cmtk.system2(cmtkcall, moreargs = NULL, ...)
```

Arguments

tool	Name of the CMTK tool
PROCESSED.ARGS	Character vector of arguments that have already been processed by the callee. Placed immediately after cmtk tool.
...	Additional named arguments to be processed by (cmtk.call, see details) or passed to system2 (cmtk.system2).
FINAL.ARGS	Character vector of arguments that have already been processed by the callee. Placed at the end of the call after optional arguments.
RETURN.TYPE	Sets return type to a character string or list (the latter is suitable for use with system2)
cmtkcall	A list containing processed arguments prepared by cmtk.call(RETURN.TYPE="list")
moreargs	Additional arguments to add to the processed call

Details

cmtk.call processes arguments in ... as follows:

argument names will be converted from arg.name to --arg-name

logical vectors (which must be of length 1) will be passed on as --arg-name

character vectors (which must be of length 1) will be passed on as --arg-name arg i.e. quoting is left up to callee.

numeric vectors will be collapsed with commas if of length greater than 1 and then passed on unquoted e.g. target.offset=c(1,2,3) will result in --target-offset 1,2,3

Value

Either a string of the form "<tool> <PROCESSED.ARGS> <...> <FINAL.ARGS>" *or* a list containing elements

- command A character vector of length 1 indicating the full path to the CMTK tool, shell quoted for protection.
- args A character vector of arguments of length 0 or greater.

See the help of system2 for details.

See Also

cmtk.bindir

Examples

```
## Not run:
cmtk.call("reformatx",'--outfile=out.nrrd', floating='floating.nrrd',
  mask=TRUE, target.offset=c(1,2,3), FINAL.ARGS=c('target.nrrd','reg.list'))
# get help for a cmtk tool
system(cmtk.call('reformatx', help=TRUE))
```

```
## End(Not run)
## Not run:
cmtk.system2(cmtk.call('mat2dof', help=TRUE, RETURN.TYPE="list"))
# capture response into an R variable
helptext=cmtk.system2(cmtk.call('mat2dof', help=TRUE, RETURN.TYPE="list"),
  stdout=TRUE)

## End(Not run)
```

`cmtk.dof2mat` *Convert CMTK registration to homogeneous affine matrix with dof2mat*

Description

Convert CMTK registration to homogeneous affine matrix with dof2mat

Usage

```
cmtk.dof2mat(reg, Transpose = TRUE, version = FALSE)
```

Arguments

`reg`	Path to input registration file or 5x3 matrix of CMTK parameters.
`Transpose`	output matrix so that form on disk matches R's convention.
`version`	Whether to return CMTK version string

Details

Transpose is true by default since this results in the orientation of cmtk output files matching the orientation in R. Do not change this unless you're sure you know what you're doing!

Value

4x4 transformation matrix

See Also

Other cmtk-commandline: `cmtk.mat2dof()`

Other cmtk-geometry: `affmat2cmtkparams()`, `cmtk.mat2dof()`, `cmtkparams2affmat()`

cmtk.extract_affine — *Extract affine registration from CMTK registration file or in-memory list*

Description

Extract affine registration from CMTK registration file or in-memory list

Usage

```
cmtk.extract_affine(r, outdir)
```

Arguments

r	A registration list or path to file on disk
outdir	Optional path to output file

Value

When `outdir` is missing a list containing the registration parameters Otherwise `NULL` invisibly.

See Also

`cmtkreglist`

Other cmtk-io: `read.cmtkreg()`, `read.cmtk()`, `write.cmtkreg()`, `write.cmtk()`

cmtk.mat2dof — *Use CMTK mat2dof to convert homogeneous affine matrix into CMTK registration*

Description

Use CMTK mat2dof to convert homogeneous affine matrix into CMTK registration

Usage

```
cmtk.mat2dof(m, f = NULL, centre = NULL, Transpose = TRUE, version = FALSE)
```

Arguments

m	Homogenous affine matrix (4x4) last row 0 0 0 1 etc
f	Output file (optional)
centre	Centre for rotation (optional 3-vector)
Transpose	the input matrix so that it is read in as it appears on disk
version	When TRUE, function returns CMTK version number of mat2dof tool

Details

If no output file is supplied, 5x3 params matrix will be returned directly. Otherwise a logical will be returned indicating success or failure at writing to disk.

Transpose is true by default since this results in an R matrix with the transpose in the fourth column being correctly interpreted by cmtk.

Value

5x3 matrix of CMTK registration parameters or logical

See Also

Other cmtk-commandline: `cmtk.dof2mat()`

Other cmtk-geometry: `affmat2cmtkparams()`, `cmtk.dof2mat()`, `cmtkparams2affmat()`

`cmtk.reformatx` *Reformat an image with a CMTK registration using the reformatx tool*

Description

Reformat an image with a CMTK registration using the reformatx tool

Usage

```
cmtk.reformatx(
  floating,
  registrations,
  output,
  target,
  mask = FALSE,
  direction = NULL,
  interpolation = c("linear", "nn", "cubic", "pv", "sinc-cosine", "sinc-hamming"),
  dryrun = FALSE,
  Verbose = TRUE,
  MakeLock = TRUE,
  OverWrite = c("no", "update", "yes"),
  filesToIgnoreModTimes = NULL,
  ...
)
```

Arguments

`floating`	The floating image to be reformatted
`registrations`	One or more CMTK format registrations on disk
`output`	The path to the output image (defaults to "<targetstem>_<floatingstem>.nrrd")

target	A character vector specifying an image file on disk, an im3d object (or an object that can be coerced to im3d) or a 6-or 9-vector defining a grid in the form Nx,Ny,Nz,dX,dY,dZ,[Ox,Oy,Oz].
mask	Whether to treat target as a binary mask (only reformatting positive voxels)
direction	Whether to transform image from sample space to reference space (called **forward** by CMTK) or from reference to sample space (called **inverse** by CMTK). Default (when NULL is forward).
interpolation	What interpolation scheme to use for output image (defaults to linear - see details)
dryrun	Just print command
Verbose	Whether to show cmtk status messages and be verbose about file update checks. Sets command line --verbose option.
MakeLock	Whether to use a lock file to allow simple parallelisation (see makelock)
OverWrite	Whether to OverWrite an existing output file. One of c("no","update","yes"). When OverWrite='update' RunCmdForNewerInput is used to determine if the output is older than any of the input files.
filesToIgnoreModTimes	Input files whose modification time should not be checked when determining if new output is required.
...	additional arguments passed to CMTK reformatx after processing by cmtk.call.

Details

Note that if you are reformatting a mask then you will need to change the interpolation to "nn", since interpolating between e.g. mask levels 72 and 74 with 73 may have unintended consequences. Presently we have no way of knowing whether an image should be treated as a mask, so the interpolation must be handled manually.

Value

the path to the output image (whether or not it was re-created afresh) or NA_character_ if no output was possible.

See Also

cmtk.bindir, cmtk.call, makelock, RunCmdForNewerInput

Examples

```
## Not run:
cmtk.reformatx('myimage.nrrd', target='template.nrrd',
  registrations='template_myimage.list')

# get full listing of command line options
system(cmtk.call('reformatx', help=TRUE))

## End(Not run)
```

`cmtk.statistics` *Calculate image statistics for a nrrd or other CMTK compatible file*

Description

Calculate image statistics for a nrrd or other CMTK compatible file

Usage

```
cmtk.statistics(
  f,
  mask,
  imagetype = c("greyscale", "label"),
  masktype = c("label", "binary"),
  ...,
  Verbose = FALSE
)
```

Arguments

`f`	Path to image file (any CMTK compatible format)
`mask`	Optional path to a mask file
`imagetype`	Whether image should be treated as greyscale (default) or label field.
`masktype`	Whether mask should be treated as label field or binary mask (default label)
`...`	Additional arguments for CMTK's `statistics` tool processed by `cmtk.call`.
`Verbose`	Whether to show cmtk status messages and be verbose about file update checks. Sets command line `--verbose` option.

Details

When given a label mask, returns a dataframe with a row for each level of the label field.

Note that the Entropy column (sometimes H, sometimes Entropy) will always be named Entropy in the returned dataframe.

Value

data.frame describing results with the following columns when image `f` is of `imagetype='greyscale'` (optionally with a mask):

- MaskLevel (only present when using a mask) the integer value of the label field for this region
- min The minimum voxel value within the current region
- max The maximum voxel value within the current region
- mean The mean voxel value within the current region
- sdev The standard deviation of voxel values within the current region

- n The count of **all** voxel within the region (irrespective of their value)
- Entropy Information theoretic entropy of voxel value distribution within region
- sum Sum of voxel values within the region

When image `f` is of `imagetype='label'`, the following results are returned:

- level The integer value of the label field for this region
- count The number of voxels in this region
- surface The surface area of this region
- volume The volume of this region
- X,Y,Z 3D coordinates of the centroid of this region

Examples

```
## Not run:
cmtk.statistics('someneuron.nrrd', mask='neuropilregionmask.nrrd')
cmtk.statistics('somelabelfield.nrrd', imagetype='label')

## End(Not run)
```

`cmtk.targetvolume` *Defines a target volume for a CMTK reformatx operation*

Description

`cmtk.targetvolume.list` is designed to cope with any user-defined class for which an as.im3d method exists. Presently the only example in the nat.* ecosystem is `nat.templatebrains::as.im3d.templatebrain`.

Usage

```
cmtk.targetvolume(target, ...)

## S3 method for class 'im3d'
cmtk.targetvolume(target, ...)

## S3 method for class 'list'
cmtk.targetvolume(target, ...)

## Default S3 method:
cmtk.targetvolume(target, ...)
```

Arguments

`target`	A character vector specifying an image file on disk, an `im3d` object (or an object that can be coerced to im3d) or a 6-or 9-vector defining a grid in the form Nx,Ny,Nz,dX,dY,dZ,[Ox,Oy,Oz].
`...`	additional arguments passed to methods

Details

if the character vector specifies an AmiraMesh file, it will be converted to a bare `im3d` object and then to an appropriate '–target-grid' specification.

Value

a character vector specifying the full cmtk reformatx '–target' or '–target-grid' argument

Examples

```
## Not run:
# see https://github.com/natverse/nat.flybrains
library(nat.flybrains)
cmtk.targetvolume(FCWB)

## End(Not run)
```

---

cmtk.version *Return cmtk version or test for presence of at least a specific version*

Description

Return cmtk version or test for presence of at least a specific version

Usage

```
cmtk.version(minimum = NULL)
```

Arguments

`minimum`	If specified checks that the cmtk version

Details

NB this function has the side effect of setting an option nat.cmtk.version the first time that it is run in the current R session.

Value

returns `numeric_version` representation of CMTK version or if minimum is not NULL, returns a logical indicating whether the installed version exceeds the current version. If CMTK is not installed returns NA.

See Also

`cmtk.bindir`, `cmtk.dof2mat`

Examples

```
## Not run:
cmtk.version()
cmtk.version('3.2.2')

## End(Not run)
```

`cmtkparams2affmat` *Compose homogeneous affine matrix from CMTK registration parameters*

Description

Compose homogeneous affine matrix from CMTK registration parameters

Usage

```
cmtkparams2affmat(
  params = NULL,
  tx = 0,
  ty = 0,
  tz = 0,
  rx = 0,
  ry = 0,
  rz = 0,
  sx = 1,
  sy = 1,
  sz = 1,
  shx = 0,
  shy = 0,
  shz = 0,
  cx = 0,
  cy = 0,
  cz = 0,
  legacy = NA
)
```

Arguments

`params`	5x3 matrix of CMTK registration parameters or list of length 5.
`tx`, `ty`, `tz`	Translation along x, y and z axes (default 0)
`rx`, `ry`, `rz`	Rotation about x, y and z axes (in degrees, default 0)
`sx`, `sy`, `sz`	Scale for x, y and z axes (default 1)
`shx`, `shy`, `shz`	Shear for x,y,z axes (default 0)
`cx`, `cy`, `cz`	Centre for rotation
`legacy`	Whether to assume that parameters are in the format used by CMTK <=2.4.0 (default value NA implies FALSE, see details).

Details

If the legacy parameter is not set explicitly, then it will be set to TRUE if params has a version attribute <2.4 or FALSE otherwise.

translation and centre components are assumed to be in physical coordinates.

Value

4x4 homogeneous affine transformation matrix

See Also

Other cmtk-geometry: affmat2cmtkparams(), cmtk.dof2mat(), cmtk.mat2dof()

---

cmtkreg *Create and test cmtkreg objects that specify path to a CMTK registration*

Description

cmtkreg creates an object of class cmtkreg that describes one (or more) CMTK registrations. This is simply a character vector that also has class cmtkreg.

as.cmtkreg converts objects to class cmtkreg, minimally just by adding an appropriate class attribute.

is.cmtkreg checks if an object is a cmtk registration either by checking class (default), or inspecting file. Supports CMTK parameter files as well as NRRD deformation fields.

Usage

```
cmtkreg(x, returnDir = NA)

as.cmtkreg(x, ...)

## S3 method for class 'matrix'
as.cmtkreg(x, ...)

## S3 method for class 'reglist'
as.cmtkreg(x, ...)

## Default S3 method:
as.cmtkreg(x, ...)

is.cmtkreg(x, filecheck = c("none", "exists", "magic"))
```

Arguments

x	Path to a cmtk registration (either plain character vector or cmtkreg object)
returnDir	Whether to return the registration directory or the actual file containing the registration. When returnDir=NA, the default, returns the input path x after validation.
...	Additional arguments passed to methods. Currently ignored.
filecheck	Whether to check object class only (default: 'none') or find and check if registration file **exists** or check **magic** value in first line of file.

---

cmtkreglist *Make in-memory CMTK registration list from affine matrix or CMTK parameters*

Description

Make in-memory CMTK registration list from affine matrix or CMTK parameters

Usage

```
cmtkreglist(x, centre = c(0, 0, 0), reference = "dummy", floating = "dummy")
```

Arguments

x	5x3 matrix of CMTK registration parameters OR 4x4 homogeneous affine matrix
centre	Optional centre of rotation passed to affmat2cmtkparams when decomposing 4x4 affine matrix
reference, floating	Path to reference and floating images.

Details

Note that this uses the modern CMTK notation of floating_study rather than model_study as used by IGSParamsToIGSRegistration (which results in an implicit inversion by CMTK tools).

Note that the reference and floating fields have no impact on the transformation encoded in the resultant .list folder and can be overridden on the command line of CMTK tools.

Value

list of class cmtkreg containing registration parameters suitable for write.cmtkreg

See Also

write.cmtkreg, affmat2cmtkparams, cmtkreg

coord2ind *Find 1D or 3D voxel indices into a 3D image given spatial coordinates*

Description

Find 1D or 3D voxel indices into a 3D image given spatial coordinates

Usage

```
coord2ind(coords, ...)

## Default S3 method:
coord2ind(
  coords,
  imdims,
  voxdims = NULL,
  origin = NULL,
  linear.indices = TRUE,
  aperm = NULL,
  Clamp = FALSE,
  CheckRanges = !Clamp,
  ...
)
```

Arguments

coords	spatial coordinates of image voxels.
...	extra arguments passed to methods.
imdims	array dimensions of 3D image *OR* an object for which a as.im3d object has been defined (see Details).
voxdims	vector of 3 voxels dimensions (width, height, depth).
origin	the origin of the 3D image.
linear.indices	Whether or not to convert the voxel indices into a linear 1D form (the default) or to keep as 3D indices.
aperm	permutation order for axes.
Clamp	Whether or not to map out of range coordinates to the nearest in range index (default FALSE)
CheckRanges	whether to check if coordinates are out of range.

Details

coord2ind is designed to cope with any user-defined class for which an as.im3d method exists. Presently the only example in the nat.* ecosystem is nat.templatebrains::as.im3d.templatebrain. The existence of an as.im3d method implies that voxdims,origin, and dim functions can be called. This is the necessary information required to convert i,j,k logical indices into x,y,z spatial indices.

See Also

ind2coord, sub2ind, ijkpos

Examples

```
coord2ind(cbind(1,2,3), imdims = c(1024,512,218),
  voxdims = c(0.622088, 0.622088, 0.622088), origin = c(0,0,0))
## Not run:
## repeat but using a templatebrain object to specify the coordinate system
library(nat.flybrains)
coord2ind(cbind(1,2,3), JFRC2)

## End(Not run)
```

---

correct_root *Interactively re-root neurons (usually to their soma)*

Description

Cycle through and manually re-root neurons using an rgl window. This will typically be used for manual identification of the soma of a neuron.

Usage

```
correct_root(someneuronlist, brain = NULL)
```

Arguments

someneuronlist	a neuron/neuronlist object
brain	for context, plot some other objects (e.g. a brain from the nat.templatebrains package such as FCWB, or any object that may be plotted using plot3d)

Value

a matrix of 3D points

Examples

```
## Not run:
## NB these neurons actually have their somata chopped off
correctedsomas = correct_root(Cell07PNs[1:3])
plot3d(correctedsomas, soma=TRUE)

## End(Not run)
```

distal_to *Return indices of points in a neuron distal to a given node*

Description

This function returns a list (containing the order of nodes) travelled using a depth first search starting from the given node.

Usage

```
distal_to(
  x,
  node.idx = NULL,
  node.pointno = NULL,
  root.idx = NULL,
  root.pointno = NULL
)
```

Arguments

x	A neuron
node.idx, node.pointno	The id(s) of node(s) from which distal points will be selected. node.idx defines the integer index (counting from 1) into the neuron's point array whereas node.pointno matches the PointNo column which will be the CATMAID id for a node.
root.idx, root.pointno	The root node of the neuron for the purpose of selection. You will rarely need to change this from the default value. See node argument for the difference between root.idx and root.pointno forms.

Value

Integer 1-based indices into the point array of points that are distal to the specified node(s) when traversing the neuron from the root to that node. Will be a vector if only one node is specified, otherwise a list is returned

See Also

subset.neuron, prune

Examples

```
## Use EM  finished DL1 projection neuron

## subset to part of neuron distal to a tag "SCHLEGEL_LH"
# nb distal_to can accept either the PointNo vertex id or raw index as a
# pivot point
```

```
dl1.lh=subset(dl1neuron, distal_to(dl1neuron,
  node.pointno = dl1neuron$tags$SCHLEGEL_LH))
plot(dl1neuron,col='blue', WithNodes = FALSE)
plot(dl1.lh, col='red', WithNodes = FALSE, add=TRUE)
```

`dl1neuron` *Olfactory Projection Neuron reconstructed from EM data*

Description

A DL1 olfactory projection neuron object traced in CATMAID from the FAFB whole brain EM volume. This has a complex morphology that makes a good test for pruning and simplification strategies.

Usage

```
dl1neuron
```

Format

A `neuron` object with additional class `catmaidneuron`

See Also

`Cell07PNs`

`dotprops` *dotprops: Neurons as point clouds with tangent vectors (but no connectivity)*

Description

`dotprops` makes dotprops representation from raw 3D points (extracting vertices from S3 objects that have them)

`dotprops.character` makes dotprops objects from one or more files on disk (typically binary segmentations saved as NRRDs). `x` can a vector of paths or be a directory (in which case `pattern` can be used to restrict the files to read). The `...` argument is passed first to `nlapply` (if there is more than one file) and then to `dotprops.default`.

`dotprops.dotprops` will default to the original vale of `k` and copy over all attributes that are not set by `dotprops.default`.

`dotprops.neuronlist` will run for every object in the neuronlist using `nlapply`. `...` arguments will be passed to `nlapply` in addition to the named argument `OmitFailures`.

Usage

```
is.dotprops(x)

as.dotprops(x, ...)

dotprops(x, ...)

## S3 method for class 'character'
dotprops(x, pattern = NULL, OmitFailures = NA, ...)

## S3 method for class 'dotprops'
dotprops(x, k = attr(x, "k"), ...)

## S3 method for class 'im3d'
dotprops(x, ...)

## S3 method for class 'neuronlist'
dotprops(x, ..., OmitFailures = NA)

## S3 method for class 'neuron'
dotprops(x, Labels = NULL, resample = NA, topo = FALSE, ...)

## Default S3 method:
dotprops(x, k = NULL, Labels = NULL, na.rm = FALSE, topo_features = NULL, ...)
```

Arguments

x	Object to be tested/converted
...	Additional arguments passed to methods
pattern	an optional regular expression. Only file names which match the regular expression will be returned.
OmitFailures	Whether to omit neurons for which FUN gives an error. The default value (NA) will result in nlapply stopping with an error message the moment there is an error. For other values, see details.
k	Number of nearest neighbours to use for tangent vector calculation (set to k=20 when passed NULL)
Labels	Vector of labels for each point e.g. identifying axon vs dendrite. The default value NULL will produce class-specific default behaviour for different classes of input object, TRUE always uses labels when an incoming object has them and FALSE never uses labels.
resample	When finite, a new length to which all segmented edges will be resampled. See resample.neuron.
topo	flag that says whether or not to add topological features (reversed Strahler Order and distance from soma)
na.rm	Whether to remove NA points (default FALSE)
topo_features	topological features of each dotprops

Details

k will default to 20 nearest neighbours when unset (i.e. when it has default value of NA) unless x is a dotprops object (when the original value of k is reused).

References

The dotprops format is essentially identical to that developed in:

Masse N.Y., Cachero S., Ostrovsky A., and Jefferis G.S.X.E. (2012). A mutual information approach to automate identification of neuronal clusters in *Drosophila* brain images. Frontiers in Neuroinformatics 6 (00021). doi:10.3389/fninf.2012.00021

See Also

nlapply

Examples

```
## Not run:
# process a single file on disk
dp=dotprops.character('~/skeleton-nrrds/file01.nrrd', k=5)
# process a whole directory of files
dps=dotprops.character('~/skeleton-nrrds/', OmitFailures=T, k=5)

## End(Not run)
```

---

fileformats *Set or return list of registered file formats that we can read*

Description

fileformats returns format names, a format definition list or a table of information about the formats that match the given filter conditions.

registerformat registers a format in the io registry

getformatreader gets the function to read a file

getformatwriter gets the function to write a file

Usage

```
fileformats(
  format = NULL,
  ext = NULL,
  read = NULL,
  write = NULL,
  class = NULL,
  rval = c("names", "info", "all")
)
```

```
registerformat(
  format = NULL,
  ext = format,
  read = NULL,
  write = NULL,
  magic = NULL,
  magiclen = NA_integer_,
  class = NULL
)

getformatreader(file, class = NULL)

getformatwriter(format = NULL, file = NULL, ext = NULL, class = NULL)
```

Arguments

format	Character vector naming the format
ext	Character vector of file extensions (including periods)
read, write	Functions to read and write this format
class	The S3 class for the format (character vector e.g. 'neuron')
rval	Character vector choosing what kind of return value fileformats will give.
magic	Function to test whether a file is of this format
magiclen	Optional integer specifying maximum number of bytes required from file header to determine file's type.
file	Path to a file

Details

if a format argument is passed to fileformats it will be matched with partial string matching and iif a unique match exists that will be returned.

getformatreader starts by reading a set number of bytes from the start off the current file and then checks using file extension and magic functions to see if it can identify the file. Presently formats are in a queue in alphabetical order, dispatching on the first match.

Value

- fileformats returns a character vector, matrix or list according to the value of rval.
- getformatreader returns a list. The reader can be accessed with $read and the format can be accessed by $format.
- getformatwriter returns a list. The writer can be accessed with $write.

getformatwriter output file

If getformatwriter is passed a file argument, it will be processed based on the registered fileformats information and the ext argument to give a final output path in the $file element of the returned list.

If ext='.someext' getformatwriter will use the specified extension to overwrite the default value returned by fileformats.

If ext=NULL, the default, and file='somefilename.someext' then file will be untouched and ext will be set to 'someext' (overriding the value returned by fileformats).

If file='somefile_without_extension' then the supplied or calculated extension will be appended to file.

If ext=NA then the input file name will not be touched (even if it has no extension at all).

Note that if ext=NULL or ext=NA, then only the specified format or, failing that, the file extension will be used to query the fileformats database for a match.

See write.neuron for code to make this discussion more concrete.

See Also

write.neuron

Examples

```
# information about the currently registered file formats
fileformats(rval='info')
## Not run:
registerformat("swc",read=read.swc,write=read.swc,magic=is.swc,magiclen=10,
  class='neuron')

## End(Not run)
swc=tempfile(fileext = '.swc')
write.neuron(Cell07PNs[[1]], swc)
stopifnot(isTRUE(getformatreader(swc)$format=='swc'))
unlink(swc)
```

---

find.neuron *Find neurons within a 3D selection box (usually drawn in rgl window)*

Description

Find neurons within a 3D selection box (usually drawn in rgl window)

Usage

```
find.neuron(
  sel3dfun = select3d(),
  indices = names(db),
  db = getOption("nat.default.neuronlist"),
  threshold = 0,
  invert = FALSE,
  rval = c("names", "data.frame", "neuronlist")
)
```

Arguments

sel3dfun	A select3d style function to indicate if points are within region
indices	Names of neurons to search (defaults to all neurons in list)
db	neuronlist to search. Can also be a character vector naming the neuronlist. Defaults to options('nat.default.neuronlist').
threshold	More than this many points must be present in region
invert	Whether to return neurons outside the selection box (default FALSE)
rval	What to return (character vector, default='names')

Details

Uses subset.neuronlist, so can work on dotprops or neuron lists.

Value

Character vector of names of selected neurons, neuronlist, or data.frame of attached metadata according to the value of rval.

See Also

select3d, find.soma, subset.neuronlist

Examples

```
## Not run:
plot3d(kcs20)
# draw a 3D selection e.g. around tip of vertical lobe when ready
find.neuron(db=kcs20)
# would return 9 neurons
# make a standalone selection function
vertical_lobe=select3d()
find.neuron(vertical_lobe, db=kcs20)
# use base::Negate function to invert the selection function
# i.e. choose neurons that do not overlap the selection region
find.neuron(Negate(vertical_lobe), db=kcs20)

## End(Not run)
```

---

find.soma *Find neurons with soma inside 3D selection box (usually drawn in rgl window)*

Description

Find neurons with soma inside 3D selection box (usually drawn in rgl window)

Usage

```
find.soma(
  sel3dfun = select3d(),
  indices = names(db),
  db = getOption("nat.default.neuronlist"),
  invert = FALSE,
  rval = c("names", "neuronlist", "data.frame")
)
```

Arguments

sel3dfun	A select3d style function to indicate if points are within region
indices	Names of neurons to search (defaults to all neurons in list)
db	neuronlist to search. Can also be a character vector naming the neuronlist. Defaults to options('nat.default.neuronlist').
invert	Whether to return neurons outside the selection box (default FALSE)
rval	What to return (character vector, default='names')

Details

Can work on neuronlists containing neuron objects *or* neuronlists whose attached data.frame contains soma positions specified in columns called X,Y,Z .

Value

Character vector of names of selected neurons

See Also

select3d, subset.neuronlist, find.neuron

---

flip *Flip an array, matrix or vector about an axis*

Description

Flip an array, matrix or vector about an axis

Usage

```
flip(x, ...)

## S3 method for class 'array'
flip(x, flipdim = "X", ...)
```

Arguments

x	Object to flip
...	Additional arguments for methods
flipdim	Character vector or 1-indexed integer indicating array dimension along which flip will occur. Characters X, Y, Z map onto dimensions 1, 2, 3.

Details

Note that dimensions 1 and 2 for R matrices will be rows and columns, respectively, which does not map easily onto the intuition of a 2D image matrix where the X axis would typically be thought of as running from left to right on the page and the Y axis would run from top to bottom.

---

get_topo_features *Get topological features per each node*

Description

Assigns to each node a distance from cell body.

Usage

```
get_topo_features(n)

get_distance_to_soma(n)
```

Arguments

n	neuron object with soma

Value

list with distance and Reversed Strahler order features per node.

vector with distances from soma

See Also

dotprops, ngraph

Examples

```
get_topo_features(Cell07PNs[[1]])
get_distance_to_soma(Cell07PNs[[1]])
```

graph.nodes *Return root, end, or branchpoints of an igraph object*

Description

Return root, end, or branchpoints of an igraph object

Usage

```
graph.nodes(
  x,
  type = c("root", "end", "branch"),
  original.ids = "name",
  exclude.isolated = TRUE
)
```

Arguments

x	An ngraph or raw igraph object
type	one of root, end (which includes root) or branch
original.ids	Use named attribute to return original vertex ids (when available). Set to FALSE when this is not desired.
exclude.isolated	Do not count isolated vertices as root/end points (default)

Details

This function underlies rootpoints.igraph methods and friends. It is conceived of as slightly lower level and end users would normally use the rootpoints methods.

graph.nodes should work for any igraph object (including ngraph objects, which inherit from igraph). However the graph must be directed in order to return a root point

See Also

rootpoints, ngraph

Examples

```
ng=as.ngraph(Cell07PNs[[1]])
# set some arbitrary vertex identifiers
igraph::vertex_attr(ng, 'name') <-sample(500, nvertices(ng))
# return those identifiers
graph.nodes(ng, type = 'end')
# ... or raw vertex indices
graph.nodes(ng,type = 'end', original.ids = FALSE)
```

im3d *Construct an im3d object representing 3D image data, densities etc*

Description

im3d objects consist of a data array with attributes defining the spatial positions at which the voxels are located. There should always be a BoundingBox attribute which defines the physical extent of the volume in the same manner as the Amira 3D visualisation and analysis software. This corresponds to the **node** centers option in the NRRD format.

Usage

```
im3d(
  x = numeric(0),
  dims = NULL,
  voxdims = NULL,
  origin = NULL,
  BoundingBox = NULL,
  bounds = NULL,
  ...
)
```

Arguments

x	The object to turn into an im3d
dims	The dimensions of the image array either as an integer vector *or* as an im3d object, whose attributes will provide defaults for dims, origin, BoundingBox, bounds arguments. The default (dims=NULL) will result in dims being set to x if x is an im3d object or dim(x) otherwise.
voxdims	The voxel dimensions
origin	the location (or centre) of the first voxel
BoundingBox, bounds	Physical extent of image. See the details section of boundingbox's help for the distinction.
...	Additional attributes such as units or materials

Details

We follow Amira's convention of setting the bounding box equal to voxel dimension (rather than 0) for any dimension with only 1 voxel.

Value

An array with additional class im3d

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

---

im3d-coords *Interconvert pixel and physical coordinates*

Description

xyzpos converts pixel coordinates to physical coordinates

ijkpos converts physical coordinates to pixel coordinates

Usage

```
xyzpos(d, ijk)

ijkpos(d, xyz, roundToNearestPixel = TRUE)
```

Arguments

d	An im3d object defining a physical space
ijk	an Nx3 matrix of pixel coordinates (1-indexed)
xyz	Nx3 matrix of physical coordinates
roundToNearestPixel	Whether to round calculated pixel coordinates to nearest integer value (i.e. nearest pixel). default: TRUE

Value

Nx3 matrix of physical or pixel coordinates

See Also

ind2coord

Other im3d: as.im3d(), boundingbox(), im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
# make an emty im3d
d=im3d(,dim=c(20,30,40),origin=c(10,20,30),voxdims=c(1,2,3))
# check round trip for origin
stopifnot(all.equal(ijkpos(d,xyzpos(d,c(1,1,1))), c(1,1,1)))
```

im3d-io *Read/Write calibrated 3D blocks of image data*

Description

Read/Write calibrated 3D blocks of image data

Usage

```
read.im3d(
  file,
  ReadData = TRUE,
  SimplifyAttributes = FALSE,
  ReadByteAsRaw = FALSE,
  ...
)

write.im3d(x, file, format = NULL, ...)
```

Arguments

file	Character vector describing a single file
ReadData	Whether to read the data itself or return metadata only. Default: TRUE
SimplifyAttributes	When TRUE leave only core im3d attributes.
ReadByteAsRaw	Whether to read byte values as R raw arrays. These occupy 1/4 memory but arithmetic is less convenient. (default: FALSE)
...	Arguments passed to methods
x	The image data to write (an im3d, or capable of being interpreted as such)
format	Character vector specifying an image format (e.g. "nrrd", "amiramesh"). Optional, since the format will normally be inferred from the file extension. See getformatwriter for details.

Details

Currently only nrrd and amira formats are implemented. Furthermore implementing a registry to allow extension to arbitrary formats remains a TODO item.

The core attributes of an im3d object are BoundingBox, origin, x, y , z where x, y, z are the locations of samples in the x, y and z image axes (which are assumed to be orthogonal).

Value

For read.im3d an objecting inheriting from base array and im3d classes.

See Also

read.nrrd, read.amiramesh

write.nrrd, getformatwriter

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
## Not run:
# read attributes of vaa3d raw file
read.im3d("L1DS1_crop_straight.raw", ReadData = F, chan=2)

## End(Not run)
```

---

image.im3d *Method to plot spatially calibrated image arrays*

Description

Method to plot spatially calibrated image arrays

Usage

```
## S3 method for class 'im3d'
image(
  x,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  plotdims = NULL,
  flipdims = "y",
  filled.contour = FALSE,
  asp = 1,
  axes = FALSE,
  xlab = NULL,
  ylab = NULL,
  nlevels = 20,
  levels = pretty(zlim, nlevels + 1),
  color.palette = colorRampPalette(c("navy", "cyan", "yellow", "red")),
  col = color.palette(length(levels) - 1),
  useRaster = NULL,
  ...
)
```

Arguments

x	The im3d object containing the data to be plotted (NAs are allowed).
xlim, ylim	ranges for the plotted x and y values, defaulting to the BoundingBox of x.
zlim	the minimum and maximum z values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The *midpoints* of the intervals cover the range, so that values just outside the range will be plotted.
plotdims	Which dimensions of 3D im3d object to plot (character vector). Defaults to c('x','y')
flipdims	Which dimensions to flip (character vector). Defaults to flipping y.
filled.contour	Whether to use a filled.contour plot instead of a regular image plot.
asp	Whether to have a a square aspect ratio (logical, default: FALSE)
axes	Whether to plot axes (default: FALSE)
xlab, ylab	each a character string giving the labels for the x and y axis. Default to the 'call names' of x or y, or to "" if these were unspecified.
nlevels	The number of colour levels in z
levels	The levels at which to break z values
color.palette	The colour palette from which col will be selected.
col	a list of colors such as that generated by rainbow, heat.colors, topo.colors, terrain.colors or similar functions.
useRaster	Whether to use rasterImage to plot images as a bitmap (much faster for large images). default useRaster=NULL checks dev.capabilities to see if raster images are supported.
...	graphical parameters for plot or image may also be passed as arguments to this function.

Value

A list with elements:

zlim The z (intensity limits)

nlevels.actual The actual number of plotted levels

nlevels.orig The requested number of plotted levels

levels The chosen levels

colors A character vector of colours

Examples

```
## Not run:
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd',package='nat'))
image(imslice(LHMask,10), asp=TRUE)
# useRaster is appreciably quicker in most cases
image(imslice(LHMask,10), asp=TRUE, useRaster=TRUE)

## End(Not run)
```

imexpand.grid *Convert locations of im3d voxel grid into XYZ coordinates*

Description

Convert locations of im3d voxel grid into XYZ coordinates

Usage

```
imexpand.grid(d)
```

Arguments

d	An im3d object

Value

Nx3 matrix of image coordinates

See Also

expand.grid

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
d=im3d(,dim=c(2,3,2),origin=c(10,20,30),voxdims=c(1,2,3))
imexpand.grid(d)
```

imscalebar *Make a scalebar to accompany an image.im3d plot*

Description

Make a scalebar to accompany an image.im3d plot

Usage

```
imscalebar(
  levels,
  col,
  nlevels = NULL,
  zlim = NULL,
  horizontal = TRUE,
  lab = "Density",
```

```
  mar = c(4, 2, 2, 2) + 0.1,
  border = NULL,
  ...
)
```

Arguments

levels	The levels at which z values were cut **or** a list returned by image.im3d
col	The plotted colours for each level
nlevels	The number of colour levels (inferred from levels when NULL)
zlim	The limits of the plotted z (intensity) values of the image
horizontal	Whether to make a horizontal or vertical scalebar (default: TRUE)
lab	The (single) axis label for the scale bar (default: Density)
mar	The margins for this plot
border	Color for rectangle border (see rect's border argument for details).
...	Additional arguments for plot

Examples

```
## Not run:
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd',package='nat'))
op=par(no.readonly = TRUE)
layout(matrix(c(1, 2), ncol = 2L), widths = c(1, 0.2))
rval=image(imslice(LHMask,10), asp=TRUE)
imscalebar(rval)
par(op)

## End(Not run)
```

---

imslice *Slice out a 3D subarray (or 2d matrix) from a 3D image array*

Description

Slice out a 3D subarray (or 2d matrix) from a 3D image array

Usage

```
imslice(x, slice, slicedim = "z", drop = TRUE)
```

Arguments

x	An im3d object
slice	Indices defining the slices to keep
slicedim	Character vector or integer defining axis from which slices will be removed.
drop	Whether singleton dimensions will be dropped (default: TRUE) converting 3D array to 2d matrix.

Details

Note the sample locations stored in the x,y,z attributes will be updated appropriately. FIXME: Should we also update bounding box?

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), is.im3d(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

`ind2coord` *Find XYZ coords corresponding to 1D indices into a 3D image*

Description

If you have an image-like object and you want to turn it into a matrix of 3D coords then you need `ind2coord`. For the reverse operation we offer `as.im3d.matrix` which allows you to turn a matrix of 3D coordinates into an `im3d` image object.

Usage

```
ind2coord(inds, ...)

## Default S3 method:
ind2coord(inds, dims, voxdims, origin, ...)

## S3 method for class 'array'
ind2coord(inds, voxdims = NULL, origin = NULL, ...)

## S3 method for class 'im3d'
ind2coord(inds, voxdims = NULL, origin = NULL, ...)
```

Arguments

`inds`	indices into an image array (either 1D, for which `dims` must be present, or a logical array).
`...`	extra arguments passed to methods.
`dims`	dimensions of 3D image array.
`voxdims`	vector of 3 voxel dimensions (width, height, depth).
`origin`	the origin.

See Also

coord2ind, sub2ind, xyzpos, as.im3d.matrix

intersect *Find the intersection of two collections of objects*

Description

Find the intersection of two collections of objects

Usage

```
intersect(x, y, ...)

## Default S3 method:
intersect(x, y, ...)

## S3 method for class 'neuronlist'
intersect(x, y, ...)
```

Arguments

x	the first collection to consider.
y	the second collection to consider.
...	additional arguments passed to methods

Details

Note that intersect.default calls base::intersect to ensure consistent behaviour for regular vectors.

Value

A collection of the same mode as x that contains all elements of x that are also present in y.

See Also

intersect

intersect_plane *Find the points on a plane that are intersected by an object*

Description

intersect_plane.neuron finds the place where a neuron intersection

Usage

```
intersect_plane(x, plane, ...)

## Default S3 method:
intersect_plane(x, plane, ...)

## S3 method for class 'neuron'
intersect_plane(x, plane, closestpoint = NULL, ...)
```

Arguments

x	A neuron, set of line segments or other data - see details.
plane	A plane, specified by the 4 coefficients of the plane equation (see plane_coefficients)
...	Additional arguments passed to methods
closestpoint	Used to define the closest hit when there are multiple

Value

A Nx3 matrix of the X,Y,Z positions of the intersections (NA when there is no intersection)

See Also

Other geometry: plane_coefficients()

Examples

```
## Find plane coefficients
# point on plane
cent=c(250.4987, 95.73561, 140.2052)
# vector normal to plane
vec=c(0.7709581, 0.03417276, -0.411977)
plc=plane_coefficients(cent, vec)

## intersect with plane
ip=intersect_plane(Cell07PNs[[1]], plc)
plot(Cell07PNs[[1]], WithNodes=FALSE)
points(ip[1], ip[2], pch=19, cex=2, col='red')

## Not run:
plot3d(Cell07PNs[[1]], col='grey', WithNodes=FALSE)
spheres3d(matrix(ip, ncol=3), col='red', rad=2)
planes3d(plc[,1:3], d=plc[,'d'])

## End(Not run)
```

is.amiramesh *Check if file is AmiraMesh format*

Description

Check if file is AmiraMesh format

Usage

```
is.amiramesh(f = NULL, bytes = NULL)
```

Arguments

f	Path to one or more files to be tested **or** an array of raw bytes, for one file only.
bytes	optional raw vector of at least 11 bytes from the start of a single file (used in preference to reading file f).

Details

Tries to be as fast as possible by reading only first 11 bytes and checking if they equal to "# AmiraMesh" or (deprecated) "# HyperMesh".

Value

logical

See Also

Other amira: amiratype(), read.amiramesh(), read.hxsurf(), write.hxsurf()

is.fijitraces *Check whether a file is in Fiji's simple neurite tracer format*

Description

This will check a file on disk to see if it is in Fiji's simple neurite tracer XML format.

Usage

```
is.fijitraces(f, bytes = NULL)
```

Arguments

f	path to a file on disk
bytes	optional raw vector of bytes used for prechecks

Details

Some prechecks (optionally taking place on a supplied raw vector of bytes) should weed out nearly all true negatives and identify many true positives without having to read/parse the file header.

---

is.im3d *Test if an object is of class im3d*

Description

Test if an object is of class im3d

Usage

```
is.im3d(x)
```

Arguments

x	Object to test

Value

logical

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), mask(), origin(), projection(), threshold(), unmask(), voxdims()

---

is.neuroml *Check whether a file is in NeuroML format*

Description

This will check a file on disk to see if it is in NeuroML format. Some prechecks (optionally taking place on a supplied raw vector of bytes) should weed out nearly all true negatives and identify many true positives without having to read/parse the file header.

Usage

```
is.neuroml(f, bytes = NULL)
```

Arguments

f	path to a file on disk
bytes	optional raw vector of bytes used for prechecks

is.neuronlist *Test objects of neuronlist class to store multiple neurons*

Description

Tests if object is a neuronlist.

Usage

```
is.neuronlist(x)
```

Arguments

x	the object to test

Details

is.neuronlist uses a relaxed definition to cope with older lists of neurons that do not have a class attribute of neuronlist.

Value

A logical indicating whether the object is a neuronlist.

See Also

Other neuronlist: *.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlistz(), neuronlist(), nlapply(), read.neurons(), write.neurons()

is.nrrd *Check if a file is a NRRD file*

Description

Check if a file is a NRRD file

Usage

```
is.nrrd(f = NULL, bytes = NULL, ReturnVersion = FALSE, TrustSuffix = FALSE)
```

Arguments

f	A character vector specifying the path or a raw vector with at least 8 bytes.
bytes	optional raw vector of at least 8 bytes from the start of a single file (used in preference to reading file f).
ReturnVersion	Whether to return the version of the nrrd format in which the file is encoded (1-5).
TrustSuffix	Whether to trust that a file ending in .nrrd or .nhdr is a NRRD

Details

Note that multiple files can be checked when a character vector of length > 1 is provided, but only one file can be checked when a raw byte array is provided.

See Also

Other nrrd: `nrrd.voxdims()`, `read.nrrd()`, `write.nrrd()`

`is.swc` *Test if a file is an SWC format neuron*

Description

Test if a file is an SWC format neuron

Usage

```
is.swc(f, TrustSuffix = TRUE)
```

Arguments

`f`	Path to one or more files
`TrustSuffix`	Whether to trust that a file ending in .nrrd or .nhdr is a NRRD

Details

Note that this test is somewhat expensive compared with the other file tests since SWC files do not have a consistent magic value. It therefore often has to read and parse the first few lines of the file in order to determine whether they are consistent with the SWC format.

Value

logical value

See Also

`read.neuron`

is.vaa3draw — *Check if a file is in the raw image format used by Hanchuan Peng's Vaa3D*

Description

See http://www.vaa3d.org/ and https://github.com/fiji/Vaa3d_Reader/blob/master/src/main/java/org/janelia/vaa3d/reader/V3dRawImageStream.java

Usage

```
is.vaa3draw(f, bytes = NULL)
```

Arguments

f	A character vector specifying the path or a raw vector (see bytes).
bytes	optional raw vector of at least 24 bytes from the start of a single file (used in preference to reading file f).

Details

Note that multiple files can be checked when a character vector of length > 1 is provided, but only one file can be checked when a raw byte array is provided.

kcs20 — *List of 20 Kenyon Cells from Chiang et al 2011 converted to dotprops objects*

Description

This R list (which has additional class neuronlist) contains 20 skeletonised *Drosophila* Kenyon cells as dotprops objects. Original data is due to Chiang et al. 2011, who have generously shared their raw data at http://flycircuit.tw. Image registration and further processing was carried out by Greg Jefferis.

References

[1] Chiang A.S., Lin C.Y., Chuang C.C., Chang H.M., Hsieh C.H., Yeh C.W., Shih C.T., Wu J.J., Wang G.T., Chen Y.C., Wu C.C., Chen G.Y., Ching Y.T., Lee P.C., Lin C.Y., Lin H.H., Wu C.C., Hsu H.W., Huang Y.A., Chen J.Y., et al. (2011). Three-dimensional reconstruction of brain-wide wiring networks in Drosophila at single-cell resolution. Curr Biol 21 (1), 1–11.

See Also

head.neuronlist, with.neuronlist, plot3d.neuronlist, plot3d.dotprops, dotprops

Other nat-data: Cell07PNs, MBL.surf

Examples

```
head(kcs20)
table(with(kcs20, type))
nopen3d()
# see plot3d.neuronlist documentation for more details

plot3d(kcs20, col=type)
```

---

makeboundingbox *Construct a 3D bounding box object*

Description

makeboundingbox explicitly constructs a 3D bounding box from a set of 6 numbers defining the opposite corners of a cube. Most of the time as an end user you will want to compute/get the bounding box of objects/vertices using boundingbox.

Usage

```
makeboundingbox(x, dims, input = c("boundingbox", "bounds"))
```

Arguments

x	A vector or matrix specifying a bounding box
dims	The number of voxels in each dimension when x is a BoundingBox matrix.
input	Whether x defines the boundingbox or bounds of the image (see details).

See Also

link{boundingbox}

---

make_model *Generate a 3D model from connector and/or tree node data*

Description

Generate a mesh3d model based on points contained in a neuronlist or neuron object, or another object that consists of 3D points.

Usage

```
make_model(
  x,
  substrate = c("cable", "connectors", "both"),
  alpha = 30,
  auto.selection = TRUE
)
```

Arguments

x	a neuronlist or neuron object, or another object that consists of 3D points
substrate	whether to make the model based on the 3D location of connectors, neuron cable or both. Connectors are pre-synapse locations, e.g. the pre-synapses of a catmaidneuron from the R package catmaid)
alpha	a single value or vector of values for alpha, fed to alphashape3d::ashape3d. Selection is subsequently interactive.
auto.selection	logical, whether or not to try and remove points based on interactively choosing simple values for clustering.

Details

Interactive function that allows a users to select points in 3D space from neuronlist/neuron objects, or another object that is coercible in 3D points using xyzmatrix. Points can first be automatically chosen, by selecting an integer number of nearest neighbours to find for each point using nabor::knn, and then a maximum distance at which nodes can be part of a cluster. Next, select_points is used to manually pick desired 3D points. Lastly, alphashape3d::ashape3d is used to create an alphashape around these points. The user can trial different values for alpha until they get their desired result.

Value

A mesh3d object

See Also

prune_online

Examples

```
## Not run:
# Make a model based off of fly olfactory projection neuron arbours
PN_blob = make_model(Cell07PNs)

## End(Not run)
```

---

mask *Mask an object, typically to produce a copy with some values zeroed out*

Description

Mask an object, typically to produce a copy with some values zeroed out

Usage

```
mask(x, ...)

## S3 method for class 'im3d'
mask(x, mask, levels = NULL, rval = c("im3d", "values"), invert = FALSE, ...)
```

Arguments

x	Object to be masked
...	Additional arguments passed to methods
mask	An im3d object, an array or a vector with dimensions compatible with x.
levels	Optional numeric vector of pixel values or character vector defining named materials.
rval	Whether to return an im3d object based on x or just the values from x matching the mask.
invert	Whether to invert the voxel selection (default FALSE)

Details

Note that mask.im3d passes ... arguments on to im3d

Value

an object with attributes matching x and elements with value as.vector(TRUE, mode=mode) i.e. TRUE, 1, 0x01 and as.vector(FALSE, mode=mode) i.e. FALSE, 0, 0x00 as appropriate.

A copy of x with

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), origin(), projection(), threshold(), unmask(), voxdims()

Examples

```
x=im3d(array(rnorm(1000),dim=c(10,10,10)), BoundingBox=c(20,200,100,200,200,300))
m=array(1:5,dim=c(10,10,10))
image(x[,,1])
image(mask(x, mask=m, levels=1)[,,1])
image(mask(x, mask=m, levels=1:2)[,,1])
```

`materials` *Extract or set the materials for an object*

Description

`materials.character` will read the materials from an im3d compatible image file on disk.

`materials.hxsurf` will extract the materials from an hxsurf object

Usage

```
materials(x, ...)

## Default S3 method:
materials(x, ...)

## S3 method for class 'character'
materials(x, ...)

## S3 method for class 'hxsurf'
materials(x, ...)
```

Arguments

`x`	An object in memory or, for `materials.character`, an image on disk.
`...`	additional parameters passed to methods (presently ignored)

Details

Note that the id column will be the 1-indexed order that the material appears in the `surf$Region` list for `hxsurf` objects and the 0-indexed mask values for an image.

Presently only AmiraMesh images are supported since they have a standardised way of encoding labels, whereas NRRDs would have to use key-value pairs according to some ad hoc convention.

Value

A `data.frame` with columns `name, id, col`

See Also

Other hxsurf: `as.hxsurf()`, `as.mesh3d()`, `plot3d.hxsurf()`, `read.hxsurf()`, `subset.hxsurf()`, `write.hxsurf()`

MBL.surf — *Surface object (hxsurf) for the left mushroom body in FCWB template space*

Description

This surface object is in the same space as the 20 Kenyon cells in kcs20.

See Also

hxsurf

Other nat-data: Cell07PNs, kcs20

Examples

```
plot3d(kcs20)
plot3d(MBL.surf, alpha=0.3)


## Not run:
## originally generated as follows
library(nat.flybrains)
MBL.surf=subset(FCWBNP.surf, "MB.*_L", drop = TRUE)

## End(Not run)
```

mirror — *Mirror 3D object about a given axis, optionally using a warping registration*

Description

mirroring with a warping registration can be used to account e.g. for the asymmetry between brain hemispheres.

`mirror.character` handles images on disk

Usage

```
mirror(x, ...)

## S3 method for class 'character'
mirror(x, output, mirrorAxisSize = NULL, target = x, ...)

## Default S3 method:
mirror(
  x,
```

```
  mirrorAxisSize,
  mirrorAxis = c("X", "Y", "Z"),
  warpfile = NULL,
  transform = c("warp", "affine", "flip"),
  ...
)

## S3 method for class 'neuronlist'
mirror(x, subset = NULL, OmitFailures = NA, ...)
```

Arguments

x	Object with 3D points (with named cols X,Y,Z) or path to image on disk.
...	additional arguments passed to methods or eventually to xform
output	Path to the output image
mirrorAxisSize	A single number specifying the size of the axis to mirror or a 2 vector (**recommended**) or 2x3 matrix specifying the boundingbox (see details).
target	Path to the image defining the target grid (defaults to the input image - hard to see when this would not be wanted).
mirrorAxis	Axis to mirror (default "X"). Can also be an integer in range 1:3.
warpfile	Optional registration or reglist to be applied *after* the simple mirroring.. It is called warpfile for historical reasons, since it is normally the path to a CMTK registration that specifies a non-rigid transformation to correct asymmetries in an image.
transform	whether to use warp (default) or affine component of registration, or simply flip about midplane of axis.
subset	For mirror.neuronlist indices (character/logical/integer) that specify a subset of the members of x to be transformed.
OmitFailures	Whether to omit neurons for which FUN gives an error. The default value (NA) will result in nlapply stopping with an error message the moment there is an error. For other values, see details.

Details

The mirrorAxisSize argument can be specified in 3 ways for the x axis with extreme values, x0+x1:

- a single number equal to x0+x1
- a 2-vector c(x0, x1) (**recommended**)
- the boundingbox for the 3D data to be mirrored: the relevant axis specified by mirrorAxis will be extracted.

This function is agnostic re node vs cell data, but for node data BoundingBox should be supplied while for cell, it should be bounds. See boundingbox for details of BoundingBox vs bounds.

See nlapply for details of the subset and OmitFailures arguments.

Value

Object with transformed points

See Also

xform, boundingbox

nlapply

Examples

```
nopen3d()
x=Cell07PNs[[1]]
mx=mirror(x,168)

plot3d(x,col='red')
plot3d(mx,col='green')


# also works with dotprops objects
nclear3d()
y=kcs20[[1]]
my=mirror(y,mirrorAxisSize=564.2532,transform='flip')

plot3d(y, col='red')
plot3d(my, col='green')


## Not run:
## Example with an image
# note that we must specify an output image (obviously) but that as a
# convenience mirror calculates the mirrorAxisSize for us
mirror('myimage.nrrd', output='myimage-mirrored.nrrd',
  warpfile='myimage_mirror.list')

# Simple flip along a different axis
mirror('myimage.nrrd', output='myimage-flipped.nrrd', mirrorAxis="Y",
  transform='flip')

## End(Not run)
```

---

nclear3d *Clear the rgl or plotly 3D scene*

Description

Clear the rgl or plotly 3D scene

Usage

```
nclear3d(..., plotengine = getOption("nat.plotengine"))
```

Arguments

...	Additional arguments passed to rgl::clear3d
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.

Details

rgl and plotly have quite different models for how to handle the active plot. nclear3d and nopen3d allow you to treat them more similarly. Use them wherever you use the rgl clear3d and open3d commands and your could she be able to run with both **plotly** or **rgl** as the plotengine.

See Also

rgl::clear3d, plot3d, plot3d.neuronlist, nopen3d

Examples

```
nclear3d()
plot3d(Cell07PNs[[1]])
```

---

ndigest *Calculated normalised digest value for an object*

Description

The *normalised* digest should exclude any fields or attributes irrelevant to the core contents of the object (e.g. timestamps, absolute location of the input files on disk etc). In theory then, this value should be constant for the same data regardless of the particular machine on which the digest is being computed.

Usage

```
ndigest(x, ...)

## S3 method for class 'neuronlistfh'
ndigest(x, ...)

## S3 method for class 'dotprops'
ndigest(x, absoluteVectors = TRUE, ...)

## S3 method for class 'neuron'
ndigest(
  x,
  fieldsToExclude = c("InputFileName", "CreatedAt", "NodeName", "InputFileStat",
    "InputFileMD5"),
  ...
)
```

Arguments

x	Object for which a normalised digest will be computed.
...	Additional arguments passed to methods and then on to digest
absoluteVectors	Whether to check only the absolute value of eigenvectors for equality (default TRUE, see details)
fieldsToExclude	Character vector naming the neuron fields to exclude

Details

ndigest.neuronlistfh only considers the keyfilemap and df (metadata data.frame) when computing the hash value. See neuronlistfh for the significance of these two fields.

ndigest.dotprops ignores any mtime or file attributes. It also converts tangent vectors to absolute values (when absoluteVectors=TRUE) because the direction vectors are computed using an eigenvector decomposition where the sign of the eigenvector is essentially random and subject to small numerical instabilities. Therefore it does not usually make sense to rely on the value of vect exactly.

ndigest.neuron ignores the following fields:

- InputFileName
- CreatedAt
- NodeName
- InputFileStat
- InputFileMD5

Value

A character string containing the digest of the supplied object computed by digest.

See Also

digest

all.equal.dotprops

all.equal.neuron

Examples

```
stopifnot(all.equal(ndigest(kcs20[[1]]), "4c045b0343938259cd9986494fc1c2b0"))
```

neuron *neuron: class to represent traced neurons*

Description

neuron makes a neuron object from appropriate variables.

is.neuron will check if an object looks like a neuron.

as.neuron will convert a suitable object to a neuron

as.neuron.data.frame expects a block of SWC format data

as.neuron.ngraph converts a graph (typically an ngraph object) to a neuron

as.neuron.igraph will convert an ngraph compatible igraph object into a neuron.

as.neuron.default will add class "neuron" to a neuron-like object.

Usage

```
neuron(
  d,
  NumPoints = nrow(d),
  StartPoint,
  BranchPoints = integer(),
  EndPoints,
  SegList,
  SubTrees = NULL,
  InputFileName = NULL,
  NeuronName = NULL,
  ...,
  MD5 = TRUE
)

is.neuron(x, Strict = FALSE)

as.neuron(x, ...)

## S3 method for class 'data.frame'
as.neuron(x, ...)

## S3 method for class 'ngraph'
as.neuron(x, vertexData = NULL, origin = NULL, Verbose = FALSE, ...)

## S3 method for class 'igraph'
as.neuron(x, ...)

## Default S3 method:
as.neuron(x, ...)
```

Arguments

d	matrix of vertices and associated data in SWC format
NumPoints	Number of points in master subtree
StartPoint, BranchPoints, EndPoints	Nodes of the neuron
SegList	List where each element contains the vertex indices for a single segments of the neuron, starting at root.
SubTrees	List of SegLists where a neuron has multiple unconnected trees (e.g. because the soma is not part of the graph, or because the neuronal arbour has been cut.)
InputFileName	Character vector with path to input file
NeuronName	Character vector containing name of neuron or a function with one argument (the full path) which returns the name. The default (NULL) sets NeuronName to the file name without the file extension.
...	Additional fields to be included in neuron. Note that if these include CreatedAt, NodeName, InputFileStat or InputFileMD5, they will override fields of that name that are calculated automatically.
MD5	Logical indicating whether to calculate MD5 hash of input
x	A neuron or other object to test/convert
Strict	Whether to check class of neuron or use a more relaxed definition based on object being a list with a SegList component.
vertexData	A dataframe with SWC fields especially X,Y,Z,W,PointNo, Parent.
origin	Root vertex, matched against names (aka PointNo) when available (see details)
Verbose	Whether to be verbose (default: FALSE)

Details

neuron objects consist of a list containing multiple fields describing the 3D location and connectivity of points in a traced neuron. The critical fields of a neuron, n, are n$d which contains a dataframe in SWC format and n$SegList which contains a representation of the neuron's topology used for most internal calculations. For historical reasons, n$SegList is limited to a *single fully-connected* tree. If the tree contains multiple unconnected subtrees, then these are stored in n$SubTrees and nTrees will be >1; the "master" subtree (typically the one with the most points) will then be stored in n$SegList and n$NumPoints will refer to the number of points in that subtree, not the whole neuron.

StartPoint, BranchPoints, EndPoints are indices matching the rows of the vertices in d **not** arbitrary point numbers typically encoded in d$PointNo.

Columns will be ordered c('PointNo','Label','X','Y','Z','W','Parent')

Uses a depth first search on the tree to reorder using the given origin.

When the graph contains multiple subgraphs, only one will be chosen as the master tree and used to construct the SegList of the resultant neuron. However all subgraphs will be listed in the SubTrees element of the neuron and nTrees will be set appropriately.

When the graph vertices have a label attribute derived from PointNo, the origin is assumed to be specified with respect to the vertex labels rather than the raw vertex ids.

Value

A list with elements: (NumPoints,StartPoint,BranchPoints,EndPoints,nTrees,NumSegs,SegList, [SubTrees]) NB SubTrees will only be present when nTrees>1.

See Also

neuronlist

graph.dfs, as.seglist

Other neuron: ngraph(), plot.dotprops(), potential_synapses(), prune(), resample(), rootpoints(), spine(), subset.neuron()

Examples

```
## See help for functions listed in See Also for more detailed examples
## Basic properties
# a sample neuron
n = Cell07PNs[[1]]

# summarise it
n

# inspect its internal structure
str(n)
# summary of 3D points
summary(xyzmatrix(n))
# identify 3d location of endpoints
xyzmatrix(n)[endpoints(n),]

## Other methods
# plot
plot(n)
# all methods for neuron objects
methods(class = 'neuron')

## Neurons as graphs
# convert to graph and find longest paths by number of nodes
ng=as.ngraph(n)
hist(igraph::distances(ng))
# ... or in distances  microns
ngw=as.ngraph(n, weights=TRUE)
hist(igraph::distances(ngw))

# converting back and forth between neurons and graphs
g=as.ngraph(Cell07PNs[[1]])
gstem=igraph::induced.subgraph(g, 1:10)
# this is fine
plot(gstem)
plot(as.neuron(gstem))

# but if you had an undirected graph
ug=igraph::as.undirected(gstem)
```

```
# you get a warning because there is no explicit origin for the graph
as.neuron(ug)

# If you need finer control of the conversion process
gstem2=as.ngraph(ug, root = 10)
plot(gstem2)
plot(as.neuron(gstem2))
```

---

neuronlist *Create a neuronlist from zero or more neurons*

Description

neuronlist objects consist of a list of neuron objects (usually of class neuron or dotprops) along with an optional attached dataframe containing information about the neurons. neuronlist objects can be indexed using their name or the number of the neuron like a regular list. Both the list itself and the attached data.frame must have the same unique (row)names. If the [operator is used to index the list, the attached dataframe will also be subsetted.

It is perfectly acceptable not to pass any parameters, generating an empty neuronlist

Usage

```
neuronlist(..., DATAFRAME = NULL)
```

Arguments

...	objects to be turned into a list
DATAFRAME	an optional data.frame to attach to the neuronlist containing information about each neuron.

Value

A new neuronlist object.

See Also

as.data.frame.neuronlist, neuronlist-dataframe-methods, neuron, dotprops

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlistz(), nlapply(), read.neurons(), write.neurons()

Examples

```
# generate an empty neuronlist
nl=neuronlist()
# slice an existing neuronlist with regular indexing
kcs5=kcs20[1:5]

# simple summary of neuronlist contents
```

```
Cell07PNs
# subset to make a smaller neuronlist
Cell07PNs[1:3]
# extract a single neuron from a neuronlist
n1=Cell07PNs[[1]]
n1


# list all methods for neuronlist objects
methods(class='neuronlist')
```

`neuronlist-dataframe-methods`
Methods for working with the dataframe attached to a neuronlist

---

Description

`[.neuronlist` and `[<-.neuronlist` behave like the corresponding base methods (`[.data.frame`, `[<-.data.frame`) allowing extraction or replacement of parts of the data.frame attached to the neuronlist.

`droplevels` Remove redundant factor levels in dataframe attached to neuronlist

`with` Evaluate expression in the context of dataframe attached to a neuronlist

`head` Return the first part of data.frame attached to neuronlist

`tail` Return the last part of data.frame attached to neuronlist

Usage

```
## S3 method for class 'neuronlist'
x[i, j, drop]

## S3 replacement method for class 'neuronlist'
x[i, j] <- value

## S3 method for class 'neuronlist'
droplevels(x, except = NULL, ...)

## S3 method for class 'neuronlist'
with(data, expr, ...)

## S3 method for class 'neuronlist'
head(x, ...)

## S3 method for class 'neuronlist'
tail(x, ...)
```

Arguments

x	A neuronlist object
i, j	elements to extract or replace. Numeric or character or, for [only, empty. Numeric values are coerced to integer as if by as.integer. See [.data.frame for details.
drop	logical. If TRUE the result is coerced to the lowest possible dimension. The default is to drop if only one column is left, but **not** to drop if only one row is left.
value	A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.
except	indices of columns from which *not* to drop levels
...	Further arguments passed to default methods (and usually ignored)
data	A neuronlist object
expr	The expression to evaluate

Value

the attached dataframe with levels dropped (NB **not** the neuronlist)

See Also

[.data.frame, @seealso [<-.data.frame

droplevels

with

head

tail

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlistfh(), neuronlistz(), neuronlist(), nlapply(), read.neurons(), write.neurons()

Examples

```
## treat kcs20 as data.frame
kcs20[1, ]
kcs20[1:3, ]
kcs20[, 1:4]
kcs20[, 'soma_side']
# alternative to as.data.frame(kcs20)
kcs20[, ]

## can also set columns
kcs13=kcs20[1:3]
kcs13[,'side']=as.character(kcs13[,'soma_side'])
head(kcs13)
# or parts of columns
kcs13[1,'soma_side']='R'
```

```
kcs13['FruMARCM-M001205_seg002','soma_side']='L'
# remove a column
kcs13[,'side']=NULL
all.equal(kcs13, kcs20[1:3])

# can even replace the whole data.frame like this
kcs13[,]=kcs13[,]
all.equal(kcs13, kcs20[1:3])

## get row/column names of attached data.frame
# (unfortunately implementing ncol/nrow is challenging)
rownames(kcs20)
colnames(kcs20)
```

`neuronlistfh` *neuronlistfh - List of neurons loaded on demand from disk or remote website*

Description

`neuronlistfh` objects consist of a list of neuron objects along with an optional attached dataframe containing information about the neurons. In contrast to `neuronlist` objects the neurons are not present in memory but are instead dynamically loaded from disk as required. `neuronlistfh` objects also inherit from `neuronlist` and therefore any appropriate methods e.g. `plot3d.neuronlist` can also be used on `neuronlistfh` objects.

`neuronlistfh` constructs a neuronlistfh object from a `filehash`, `data.frame` and `keyfilemap`. End users will **not** typically use this function to make a `neuronlistfh`. They will usually read them using `read.neuronlistfh` and sometimes create them by using `as.neuronlistfh` on a `neuronlist` object.

`is.neuronlistfh` test if an object is a neuronlistfh

`as.neuronlistfh` generic function to convert an object to neuronlistfh

`as.neuronlistfh.neuronlist` converts a regular neuronlist to one backed by a filehash object with an on disk representation

`c.neuronlistfh` adds additional neurons from one or more neuronlist objects to a `neuronlistfh` object.

Usage

```
neuronlistfh(db, df, keyfilemap, hashmap = 1000L)

is.neuronlistfh(nl)

as.neuronlistfh(x, df, ...)

## S3 method for class 'neuronlist'
as.neuronlistfh(
```

```
  x,
  df = attr(x, "df"),
  dbdir = NULL,
  dbClass = c("RDS", "RDS2", "DB1"),
  remote = NULL,
  WriteObjects = c("yes", "no", "missing"),
  ...
)

## S3 method for class 'neuronlistfh'
c(..., recursive = FALSE)
```

Arguments

db	a filehash object that manages an on disk database of neuron objects. See Implementation details.
df	Optional dataframe, where each row describes one neuron
keyfilemap	A named character vector in which the elements are filenames on disk (managed by the filehash object) and the names are the keys used in R to refer to the neuron objects. Note that the keyfilemap defines the order of objects in the neuronlist and will be used to reorder the dataframe if necessary.
hashmap	A logical indicating whether to add a hashed environment for rapid object lookup by name or an integer or an integer defining a threshold number of objects when this will happen (see Implementation details).
nl	Object to test
x	Object to convert
...	Additional arguments for methods, eventually passed to neuronlistfh() constructor.
dbdir	The path to the underlying filehash database on disk. For RDS formats, by convention this should be a path whose final element is 'data' which will be turned into a directory. For DB1 format it specifies a single file to which objects will be written.
dbClass	The filehash database class. Defaults to RDS.
remote	The url pointing to a remote repository containing files for each neuron.
WriteObjects	Whether to write objects to disk. Missing implies that existing objects will not be overwritten. Default "yes".
recursive	currently ignored

Value

a neuronlistfh object which is a character vector with classes neuronlistfh, neuronlist and attributes db, df. See Implementation details.

Modifying neuronlistfh objects

The recommended way to do this is by using the `c.neuronlistfh` method to append one or more neuronlists to a neuronlistfh object. This ensures that the attached metadata for each data.frame is handled properly. Use as `nlfh <- c(nlfh, nl2)`. If you want to combine two `neuronlistfh` objects, it may make sense to choose the bigger one as the first-listed argument to which additional neurons are appended.

There is also low-level and quite basic support for modifying neuronlistfh objects using the `[[` operator. There are two modes depending on the nature of the index in the assignment operation `nlfh[[index]]<-neuron`:

- numeric index *for replacement of items only*
- character index *for replacement **or** addition of items*

This distinction is because there must be a character key provided to name the neuron when a new one is being added, whereas an existing element can be referenced by position (i.e. the numeric index). Unfortunately the end user is responsible for manually modifying the attached data.frame when new neurons are added. Doing `nlfh[[index]]<-neuron` will do the equivalent of `attr(nlfh, 'df')[i, ]=NA` i.e. add a row containing NA values.

Implementation details

neuronlistfh objects are a hybrid between regular `neuronlist` objects that organise data and metadata for collections of neurons and a backing `filehash` object. Instead of keeping objects in memory, they are *always* loaded from disk. Although this sounds like it might be slow, for nearly all practical purposes (e.g. plotting neurons) the time to read the neuron from disk is small compared with the time to plot the neuron; the OS will cache repeated reads of the same file. The benefits in memory and startup time (<1s vs 100s for our 16,000 neuron database) are vital for collections of 1000s of neurons e.g. for dynamic report generation using knitr or for users with <8Gb RAM or running 32 bit R.

neuronlistfh objects include:

`attr("keyfilemap")` A named character vector that determines the ordering of objects in the neuronlist and translates keys in R to filenames on disk. For objects created by `as.neuronlistfh` the filenames will be the md5 hash of the object as calculated using `digest`. This design means that the same key can be used to refer to multiple distinct objects on disk. Objects are effectively versioned by their contents. So if an updated neuronlistfh object is posted to a website and then fetched by a user it will result in the automated download of any updated objects to which it refers.

`attr("db")` The backing database - typically of class `filehashRDS`. This manages the loading of objects from disk.

`attr(x,"df")` The data.frame of metadata which can be used to select and plot neurons. See `neuronlist` for examples.

codeattr(x,"hashmap") (Optional) a hashed environment which can be used for rapid lookup using key names (rather than numeric/logical indices). There is a space potential to pay for this redundant lookup method, but it is normally worth while given that the dataframe object is typically considerably larger. To give some numbers, the additional environment might occupy ~ 1 reduce mean lookup time from 0.5 ms to 1us. Having located the object, on my machine it can take as little as 0.1ms to load from disk, so these savings are relevant.

Presently only backing objects which extend the `filehash` class are supported (although in theory other backing objects could be added). These include:

- filehash RDS
- filehash RDS2 (experimental)
- filehash DB1 (experimental)

We have also implemented a simple remote access protocol (currently only for the `RDS` format). This allows a neuronlistfh object to be read from a url and downloaded to a local path. Subsequent attempts to access neurons stored in this list will result in automated download of the requested neuron to the local cache.

An alternative backend, the experimental `RDS2` format is supported (available at https://github.com/jefferis/filehash). This is likely to be the most effective for large (5,000-500,000) collections of neurons, especially when using network filesystems (`NFS`, `AFP`) which are typically very slow at listing large directories.

Finally the DB1 backend keeps the data in a single monolithic file on disk. This may work better when there are many small neurons (think >10,000 files occupying only a few GB) on NFS network file systems or Google Drive, neither of which are keen on having many files especially in the same folder. It does not allow updates from a remote location. See `filehashDB1-class` for more details.

Note that objects are stored in a filehash, which by definition does not have any ordering of its elements. However neuronlist objects (like lists) do have an ordering. Therefore the names of a neuronlistfh object are not necessarily the same as the result of calling `names()` on the underlying filehash object.

See Also

`filehash-class`

Other neuronlistfh: `[.neuronlistfh()`, `read.neuronlistfh()`, `remotesync()`, `write.neuronlistfh()`

Other neuronlist: `*.neuronlist()`, `is.neuronlist()`, `neuronlist-dataframe-methods`, `neuronlistz()`, `neuronlist()`, `nlapply()`, `read.neurons()`, `write.neurons()`

Examples

```
## Not run:
kcnl=read.neuronlistfh('http://jefferislab.org/si/nblast/flycircuit/kcs20.rds',
'path/to/my/project/folder')
# this will automatically download the neurons from the web the first time
# it is run
plot3d(kcnl)

kcfh <- as.neuronlistfh(kcs20[1:18])
# add more neurons
kcfh <- c(kcfh, kcs20[19], kcs20[20])
# convert back to regular (in memory) neuronlist
all.equal(as.neuronlist(kcfh), kcs20)

## End(Not run)
## Not run:
# create neuronlistfh object backed by filehash with one file per neuron
```

```
# by convention we create a subfolder called data in which the objects live
kcs20fh=as.neuronlistfh(kcs20, dbdir='/path/to/my/kcdb/data')
plot3d(subset(kcs20fh,type=='gamma'))
# ... and, again by convention, save the neuronlisfh object next to filehash
# backing database
write.neuronlistfh(kcs20fh, file='/path/to/my/kcdb/kcdb.rds')

# in a new session
read.neuronlistfh("/path/to/my/kcdb/kcdb.rds")
plot3d(subset(kcs20fh, type=='gamma'))

# using the DB1 backing store (a single file on disk for all objects)
kcs20fh=as.neuronlistfh(kcs20, dbdir='/path/to/my/kcdb/kcs20fh')
# store metadata on disk
write.neuronlistfh(kcs20fh, file='/path/to/my/kcdb/kcs20fh.rds')
# read in again in a new session. You will need these two files
# kcs20fh kcs20fh.rds
kcs20fh2 <- read.neuronlistfh("/path/to/my/kcdb/kcs20fh.rds")

## End(Not run)
```

`neuronlistz` *A neuronlist object that will read neurons from a zip file on demand*

Description

`as.neuronlist.neuronlistz` converts a `neuronlistz` to a regular (in memory) `neuronlist`

Usage

```
neuronlistz(zip, patt = NULL, df = NULL, ...)

## S3 method for class 'neuronlistz'
as.neuronlist(l, ...)
```

Arguments

`zip`	Path to the zip file
`patt`	Optional regex pattern or function to specify a subset of files.
`df`	A `data.frame` of metadata that will be attached to the `neuronlistz` and will define the order of the objects inside it.
`...`	Additional arguments currently ignored
`l`	An existing list or a single neuron to start a list

Details

`neuronlistz` is designed to wrap zip files containing neurons saved in the RDS or faster/smaller qs format for rapid access. You should be able to read typical files in <20 ms. For files of ~3 GB there is a fixed cost of the order of 10-15ms per read.

Value

A neuronlist object with additional class neuronlistz

See Also

neuronlist, neuronlistfh, write.neurons

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlist(), nlapply(), read.neurons(), write.neurons()

Examples

```
write.neurons(Cell07PNs[1:5], tf <- tempfile(fileext = '.zip'), format='rds')
nz=neuronlistz(tf)
nz[[1]]
nz[1:5]

## Not run:
write.neurons(Cell07PNs[1:5], tf <- tempfile(fileext = '.zip'), format='qs')
nz2=neuronlistz(tf)
all.equal(nz2[1:3], nz[1:3])

## End(Not run)
```

---

ngraph *ngraph: a graph to encode a neuron's connectivity*

Description

the ngraph class contains a (completely general) graph representation of a neuron's connectivity in an igraph object. It may additionally contain vertex name or position data. See Connectivity section.

ngraph() creates an ngraph from edge and vertex information.

as.ngraph converts an object to an ngraph

as.ngraph.dataframe construct ngraph from a data.frame containing SWC format data

as.ngraph.neuron construct ngraph from a neuron

Usage

```
ngraph(
  el,
  vertexnames,
  xyz = NULL,
  diam = NULL,
  directed = TRUE,
  weights = FALSE,
  vertex.attributes = NULL,
```

```
  graph.attributes = NULL
)

as.ngraph(x, ...)

## S3 method for class 'data.frame'
as.ngraph(x, directed = TRUE, ...)

## S3 method for class 'neuron'
as.ngraph(x, directed = TRUE, method = c("swc", "seglist"), ...)
```

Arguments

el	A two column matrix (start, end) defining edges. start means closer to the root (soma) of the neuron.
vertexnames	Integer names for graph nodes - the edge list is specified using these names (see details).
xyz	3D coordinates of vertices (optional, Nx3 matrix, or Nx4 matrix when 4th column is assumed to be diameter)
diam	Diameter of neuron at each vertex (optional)
directed	Whether the resultant graph should be directed (default TRUE)
weights	Logical value indicating whether edge weights defined by the 3D distance between points should be added to graph (default FALSE) *or* a numeric vector of weights.
vertex.attributes, graph.attributes	List of named attributes to be added to the graph. The elements of vertex.attributes must be vectors whose length is compatible with the number of elements in the graph. See set.vertex.attribute for details.
x	Object to convert (see method descriptions)
...	Arguments passed to methods
method	Whether to use the swc data (x$d) or the seglist to define neuronal connectivity to generate graph.

Details

Note that the as.ngraph.neuron method *always* keeps the original vertex names (a.k.a. PointNo) as read in from the original file.

Value

an igraph object with additional class ngraph, having a vertex for each entry in vertexnames, each vertex having a label attribute. All vertices are included whether connected or not.

Connectivity

We make the following assumptions about neurons coming in

- They have an integer vertex name that need not start from 1 and that may have gaps. This is analogous to the PointNo field of the core data block of neuron objects.
- The edge list that defines connectivity specifies those edges using pairs of vertex names, _not_ raw vertex indices.

We make no attempt to determine the root points at this stage.

The raw vertex ids in the graph will be in the order of vertexnames and can therefore be used to index a block of vertex coordinates. The vertexnames will be stored using the vertex attribute name. The underlying igraph class allows nodes to be specified by their name. This provides a convenient way to define nodes in an ngraph object by the numeric identifier encoded by the PointNo field of the corresponding neuron.

When the graph is directed (default) the edges will be from the root to the other tips of the neuron.

Morphology

The morphology of the neuron is encoded by the combination of connectivity information (i.e. the graph) and spatial data encoded as the 3D position and diameter of each vertex. Position information is stored as vertex attributes X, Y, and Z.

See Also

igraph, set.vertex.attribute, subset.neuron for example of graph-based manipulation of a neuron, plot3d.ngraph

Other neuron: neuron(), plot.dotprops(), potential_synapses(), prune(), resample(), rootpoints(), spine(), subset.neuron()

Examples

```
n=Cell07PNs[[1]]
g=as.ngraph(n)
library(igraph)
# check that vertex attributes of graph match X position
all.equal(V(g)$X, n$d$X)

# Use 3D segment lengths as edge length of graph
gw=as.ngraph(n, weights=TRUE)
# find longest path across graph
d=get.diameter(gw)
# make a new neuron using the longest path
gw_spine=as.neuron(induced.subgraph(gw, d))
# make a new neuron containing all nodes except those in longest path
gw_antispine=as.neuron(delete.vertices(gw, d))

# note use of bounding box of original neuron to set plot axes
plot(gw_spine, col='red', boundingbox=boundingbox(n))
plot(gw_antispine, col='blue', add=TRUE)
```

nlapply *lapply and mapply for neuronlists (with optional parallelisation)*

Description

Versions of lapply and mapply that look after the class and attached dataframe of neuronlist objects. `nlapply` can apply a function to only a `subset` of elements in the input neuronlist. Internally `nlapply` uses `plyr::llply` thereby enabling progress bars and simple parallelisation (see plyr section and examples).

`progress_natprogress` provides a progress bar compatible with the `progress::progress_bar`.

Usage

```
nlapply(
  X,
  FUN,
  ...,
  subset = NULL,
  OmitFailures = NA,
  .progress = getOption("nat.progress", default = "auto")
)

progress_natprogress(...)

nmapply(
  FUN,
  X,
  ...,
  MoreArgs = NULL,
  SIMPLIFY = FALSE,
  USE.NAMES = TRUE,
  subset = NULL,
  OmitFailures = NA,
  .progress = getOption("nat.progress", default = "auto")
)
```

Arguments

`X`	A neuronlist
`FUN`	Function to be applied to each element of X
`...`	Additional arguments for FUN (see details)
`subset`	Character, numeric or logical vector specifying on which subset of `X` the function `FUN` should be applied. Elements outside the subset are passed through unmodified.

OmitFailures	Whether to omit neurons for which FUN gives an error. The default value (NA) will result in nlapply stopping with an error message the moment there is an error. For other values, see details.
.progress	Character vector specifying the type of progress bar (see Progress bar section for details) The default value of "auto" shows a progress bar in interactive use after 2s. The default value can be overridden for the current session by setting the value of options(nat.progressbar) (see examples). Values of T and F are aliases for 'text' and 'none', respectively.
MoreArgs	a list of other arguments to FUN.
SIMPLIFY	logical or character string; attempt to reduce the result to a vector, matrix or higher dimensional array; see the simplify argument of sapply.
USE.NAMES	logical; use the names of the first ... argument, or if that is an unnamed character vector, use that vector as the names.

Details

When OmitFailures is not NA, FUN will be wrapped in a call to try to ensure that failure for any single neuron does not abort the nlapply/nmapply call. When OmitFailures=TRUE the resultant neuronlist will be subsetted down to return values for which FUN evaluated successfully. When OmitFailures=FALSE, "try-error" objects will be left in place. In either of the last 2 cases error messages will not be printed because the call is wrapped as try(expr, silent=TRUE).

Value

A neuronlist

plyr

The arguments of most interest from plyr are:

- .inform set to TRUE to give more informative error messages that should indicate which neurons are failing for a given applied function.
- .progress set to "text" for a basic progress bar
- .parallel set to TRUE for parallelisation after registering a parallel backend (see below).
- .paropts Additional arguments for parallel computation. See llply for details.

Before using parallel code within an R session you must register a suitable parallel backend. The simplest example is the multicore option provided by the doMC package that is suitable for a spreading computational load across multiple cores on a single machine. An example is provided below.

Note that the progress bar and parallel options cannot be used at the same time. You may want to start a potentially long-running job with the progress bar option and then abort and re-run with .parallel=TRUE if it looks likely to take a very long time.

Progress bar

There are currently two supported approaches to defining progress bars for nlapply. The default (when progress="auto") now uses a progress bar built using progress_bar from the progress package, which can be highly customised. The alternative is to use the progress bars distributed directly with the plyr package such as progress_text.

In either case the value of the .progress argument must be a character vector which names a function. According to plyr's convention an external function called progress_myprogressbar will be identified by setting the argument to .progress="myprogressbar". By default the supplied progress_natprogress function will be used when .progress="auto"; this function will probably not be used directly by end users; however it must be exported for nlapply with progress to work properly in other functions.

For nmapply only the default nat_progress bar can be shown for architectural reasons. It will be shown in interactive mode when .progress='auto' (the default). The progress bar can be suppressed by setting .progress='none'. Any other value will result in a progress bar being shown in both interactive and batch modes.

See Also

lapply

mapply

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlistz(), neuronlist(), read.neurons(), write.neurons()

Examples

```
## nlapply example
kcs.reduced=nlapply(kcs20,function(x) subset(x,sample(nrow(x$points),50)))
open3d()
plot3d(kcs.reduced,col='red', lwd=2)
plot3d(kcs20,col='grey')
close3d()

## Not run:
# example of using plyr's .inform argument for debugging error conditions
xx=nlapply(Cell07PNs, prune_strahler)
# oh dear there was an error, let's get some details about the neuron
# that caused the problem
xx=nlapply(Cell07PNs, prune_strahler, .inform=TRUE)

## End(Not run)

## Not run:
## nlapply example with plyr
## dotprops.neuronlist uses nlapply under the hood
## the .progress and .parallel arguments are passed straight to
system.time(d1<-dotprops(kcs20,resample=1,k=5,.progress='text'))
## plyr+parallel
library(doMC)
# can also specify cores e.g. registerDoMC(cores=4)
```

```
registerDoMC()
system.time(d2<-dotprops(kcs20,resample=1,k=5,.parallel=TRUE))
stopifnot(all.equal(d1,d2))

## End(Not run)

## nmapply example
# flip first neuron in X, second in Y and 3rd in Z
xyzflip=nmapply(mirror, kcs20[1:3], mirrorAxis = c("X","Y","Z"),
  mirrorAxisSize=c(400,20,30))

# this artificial example will show a progress bar in interactive use
xyzflip=nmapply(function(...) {Sys.sleep(.2);mirror(...)}, kcs20,
  mirrorAxis= sample(LETTERS[24:26], size = 20, replace = TRUE),
  mirrorAxisSize=runif(20, min=40, max=200))

open3d()
plot3d(kcs20[1:3])
plot3d(xyzflip)
close3d()


## Not run:
## Override default progress bar behaviour via options
# sleep 50ms per neuron to ensure progress bar gets triggered
sl=nlapply(Cell07PNs, FUN = function(x) {Sys.sleep(0.05);seglengths(x)})
# the default progess bar for nat < 1.9.1
options(nat.progress='traditional')
sl=nlapply(Cell07PNs, FUN = seglengths)
# no progress bar ever
options(nat.progress='none')
sl=nlapply(Cell07PNs, FUN = function(x) {Sys.sleep(0.05);seglengths(x)})
# back to normal
options(nat.progress=NULL)
sl=nlapply(Cell07PNs, FUN = seglengths)

## End(Not run)
```

---

nlscan *Scan through a set of neurons, individually plotting each one in 3D*

Description

Can also choose to select specific neurons along the way and navigate forwards and backwards.

Usage

```
nlscan(
  neurons,
  db = NULL,
```

```
  col = "red",
  Verbose = T,
  Wait = T,
  sleep = 0.1,
  extrafun = NULL,
  selected_file = NULL,
  selected_col = "green",
  yaml = TRUE,
  ...,
  plotengine = "rgl"
)
```

Arguments

neurons	a neuronlist object or a character vector of names of neurons to plot from the neuronlist specified by db.
db	A neuronlist to use as the source of objects to plot. If NULL, the default, will use the neuronlist specified by options('nat.default.neuronlist')
col	the color with which to plot the neurons (default 'red').
Verbose	logical indicating that info about each selected neuron should be printed (default TRUE).
Wait	logical indicating that there should be a pause between each displayed neuron.
sleep	time to pause between each displayed neuron when Wait=TRUE.
extrafun	an optional function called when each neuron is plotted, with two arguments: the current neuron name and the current selected neurons.
selected_file	an optional path to a yaml file that already contains a selection.
selected_col	the color in which selected neurons (such as those specified in selected_file) should be plotted.
yaml	a logical indicating that selections should be saved to disk in (human-readable) yaml rather than (machine-readable) rda format.
...	extra arguments to pass to plot3d.
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.

Value

A character vector of names of any selected neurons, of length 0 if none selected.

See Also

plot3d.character, plot3d.neuronlist

Examples

```
## Not run:
# scan a neuronlist
nlscan(kcs20)
```

```
# using neuron names
nlscan(names(kcs20), db=kcs20)
# equivalently using a default neuron list
options(nat.default.neuronlist='kcs20')
nlscan(names(kcs20))

## End(Not run)
# scan without waiting
nlscan(kcs20[1:4], Wait=FALSE, sleep=0)
## Not run:
# could select e.g. the gamma neurons with unbranched axons
gammas=nlscan(kcs20)
nclear3d()
plot3d(kcs20[gammas])

# plot surface model of brain first
# nb depends on package only available on github
devtools::install_github(username = "natverse/nat.flybrains")
library(nat.flybrains)
plot3d(FCWB)
# could select e.g. the gamma neurons with unbranched axons
gammas=nlscan(kcs20)

nclear3d()
plot3d(kcs20[gammas])

## End(Not run)
```

---

nopen3d *Open customised rgl window*

Description

Open customised rgl window

Usage

```
nopen3d(bgcol = "white", FOV = 0, ...)
```

Arguments

bgcol	background colour
FOV	field of view
...	additional options passed to open3d

Details

Pan with right button (Ctrl+click), zoom with middle (Alt/Meta+click) button. On a Mac trackpad, pan with two fingers left-right, zoom with two fingers in-out. Defaults to a white background and orthogonal projection (FOV=0)

Note that sometimes (parts of) objects seem to disappear after panning and zooming. See help for pan3d.

rgl and plotly have quite different models for how to handle the active plot. nopen3d and nclear3d allow you to treat them more similarly. Use them wherever you use the rgl clear3d and open3d commands and your could she be able to run with both **plotly** or **rgl** as the plotengine.

Value

current rgl device

See Also

open3d, pan3d, nclear3d

---

normalise_swc *Normalise an SWC format block of neuron morphology data*

Description

Normalise an SWC format block of neuron morphology data

Usage

```
normalise_swc(
  x,
  requiredColumns = c("PointNo", "Label", "X", "Y", "Z", "W", "Parent"),
  ifMissing = c("usedefaults", "warning", "stop"),
  includeExtraCols = TRUE,
 defaultValue = list(PointNo = seq.int(nrow(x)), Label = 2L, X = NA_real_, Y = NA_real_,
    Z = NA_real_, W = NA_real_, Parent = NA_integer_)
)
```

Arguments

x	A data.frame containing neuron morphology data
requiredColumns	Character vector naming columns we should have
ifMissing	What to do if x is missing a required column
includeExtraCols	Whether to include any extra columns include in codex
defaultValue	A list containing default values to use for any missing columns

Details

Note that row.names of the resultant data.frame will be set to NULL so that they have completely standard values.

Value

A data.frame containing the normalised block of SWC data with standard columns in standard order.

See Also

as.neuron.data.frame, seglist2swc

---

npop3d *Remove plotted neurons or other 3D objects*

Description

The normal usage will not specify x in which case the last neurons plotted by plot3d.neuronlist or any of its friends will be removed.

Usage

```
npop3d(x, slow = FALSE, type = "shapes")
```

Arguments

x	rgl ids of objects to remove
slow	Whether to remove neurons one by one (slowly) default: FALSE
type	Type of objects to remove see pop3d.

See Also

pop3d, plot3d.neuronlist

nrrd.voxdims *Return voxel dimensions (by default absolute voxel dimensions)*

Description

Return voxel dimensions (by default absolute voxel dimensions)

Usage

```
nrrd.voxdims(file, ReturnAbsoluteDims = TRUE)
```

Arguments

file	path to nrrd/nhdr file or a list containing a nrrd header
ReturnAbsoluteDims	Defaults to returning absolute value of dims even if there are any negative space directions

Details

NB Can handle off diagonal terms in space directions matrix, BUT assumes that space direction vectors are orthogonal.

Will produce a warning if no valid dimensions can be found.

Value

numeric vector of voxel dimensions (NA_real_ when missing) of length equal to the image dimension.

See Also

Other nrrd: is.nrrd(), read.nrrd(), write.nrrd()

nvertices *Find the number of vertices in an object (or each element of a neuronlist)*

Description

Find the number of vertices in an object (or each element of a neuronlist)

Usage

```
nvertices(x, ...)

## Default S3 method:
nvertices(x, ...)

## S3 method for class 'neuronlist'
nvertices(x, ...)

## S3 method for class 'shapelist3d'
nvertices(x, ...)
```

Arguments

x	An object with 3d vertices (e.g. neuron, surface etc)
...	Additional arguments passed to methods (currently ignored)

Value

an integer number of vertices (or a vector of length equal to a neuronlist)

Examples

```
nvertices(Cell07PNs[[1]])
nvertices(kcs20)
```

---

nview3d *Set the 3D viewpoint of an RGL window using anatomical terms*

Description

Set the 3D viewpoint of an RGL window using anatomical terms

Usage

```
nview3d(
 viewpoint = c("frontal", "anterior", "dorsal", "ventral", "posterior", "left", "right",
    "oblique_right", "oblique_left"),
  FOV = 0,
  extramat = NULL,
  ...
)
```

Arguments

viewpoint	Character vector specifying viewpoint
FOV	The Field of View (defaults to 0 => orthographic projection) (see par3d for details).
extramat	An optional extra transformation matrix to be applied after the one implied by the viewpoint argument.
...	additional arguments passed to par3d

See Also

nopen3d, view3d

Examples

```
plot3d(kcs20, soma=TRUE)
nview3d('frontal')
nview3d('ant')
nview3d()
nview3d('posterior')
nview3d('oblique_right')
# a slightly oblique frontal view
nview3d('frontal', extramat=rotationMatrix(pi/10, 1, 1, 0))
```

Ops.dotprops *Arithmetic for nat dotprops and surface objects*

Description

Arithmetic for nat dotprops and surface objects

Usage

```
## S3 method for class 'dotprops'
Ops(e1, e2 = NULL)

## S3 method for class 'mesh3d'
Ops(e1, e2 = NULL)

## S3 method for class 'hxsurf'
Ops(e1, e2 = NULL)
```

Arguments

e1	A dotprops or surface (hxsurf, mesh3d) object
e2	A scalar or 3-vector that will be applied to the dotprops object

Value

A new dotprops / surface object

See Also

scale.dotprops, Ops.neuron

Examples

```
kcs20.shift=kcs20+c(2,3,4)

plot3d(kcs20, col='grey')
plot3d(kcs20.shift, col='red')
```

Ops.neuron *Arithmetic for neuron coordinates*

Description

If x is a 1-vector or a 3-vector, operate on xyz only If x is a 4-vector, apply operation to xyz and diameter

Usage

```
## S3 method for class 'neuron'
Ops(e1, e2 = NULL)
```

Arguments

e1	a neuron
e2	(a numeric vector to multiply neuron coords in neuron)

Value

modified neuron

See Also

neuron

Examples

```
n1<-Cell07PNs[[1]]*2
n2<-Cell07PNs[[1]]*c(2,2,2,1)
stopifnot(all.equal(n1,n2))
n3<-Cell07PNs[[1]]*c(2,2,4)
```

origin — *Return the space origin of a 3D image object*

Description

Defined as the first coordinates (x,y,z) of the bounding box, which in turn matches the nrrd definition of the location of the "centre" of the first voxel.

Usage

```
origin(x, ...)
```

Arguments

x	Object for which origin should be returned. See boundingbox.
...	Additional arguments passed to boundingbox

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), projection(), threshold(), unmask(), voxdims()

overlap_score — *Generate a connectivity matrix based on euclidean distance between points*

Description

Generates an 'overlap matrix' of overlap scores between neurons in the outputneurons and inputneurons pools. For every point in a given neuron in outputneurons, a distance score is calculated to every point in a neuron in inputneurons. The sum of this score is added to the final output matrix. The score is calculated as e(-d^2/(2*delta^2)), where d is the euclidean distance between the two points, and delta is the expected distance in um that is considered 'close'. It is recommended that the user resamples neurons before use, using resample.

Usage

```
overlap_score(outputneurons, inputneurons, delta = 1, progress = TRUE)
```

Arguments

outputneurons	first set of neurons
inputneurons	second set of neurons
delta	the distance (in um) at which a synapse might occur
progress	whether or not to have a progress bar

Value

a matrix of overlap scores

See Also

potential_synapses, resample

Examples

```
## Not run:
# Calculate how much some neurons overlap with one another
## Example requires the package nat.flybrains
Cell07PNs_overlap = overlap_score(outputneurons = Cell07PNs, inputneurons = Cell07PNs)

## Plot the results
heatmap(Cell07PNs_overlap)

## End(Not run)
```

---

pan3d *Some useful extensions / changes to rgl defaults*

Description

Set up pan call back for current rgl device

Usage

```
pan3d(button)
```

Arguments

button	Integer from 1 to 3 indicating mouse button

Details

Copied verbatim from ?rgl.setMouseCallbacks for rgl version 0.92.892 Mouse button 2 is right and button 3 is middle (accessed by Meta/Alt key)

Note that sometimes (parts of) objects seem to disappear after panning and zooming. The example in rgl.setMouseCallbacks from which this is copied includes a note that "this doesn't play well with rescaling"

Author(s)

Duncan Murdoch

See Also

rgl.setMouseCallbacks

Examples

```
## Not run:
 open3d()
 pan3d(2)

## End(Not run)
```

`plane_coefficients` *Find the coefficients of the plane equation*

Description

Find the coefficients of the plane equation

Usage

```
plane_coefficients(p, n)
```

Arguments

p	A point on the plane (or N x 3 matrix of multiple points)
n	vector normal to the plane (or N x 3 matrix of multiple vectors)

Details

Both `p` and `n` can accept multiple points/vectors to calculate many planes at once.

Value

a matrix with 4 columns `a,b,c,d` where `ax + by + cz + d = 0`

See Also

Other geometry: `intersect_plane()`

Examples

```
# Mushroom Body Entry Point - plane perpendicular to axon tract as
# projection neurons enter mushroom body calyx
mbe=plane_coefficients(p=c(207, 102, 142), n=c(.6,-0.1,0.3))
## Not run:
plot3d(Cell07PNs)
planes3d(mbe[1:3], d=mbe[4])

## End(Not run)
```

plot.dotprops *Plot a 2D projection of a neuron*

Description

plot.dotprops plots a 2D projection of a dotprops format object

plot.neuron plots a 2D projection of a neuron

Usage

```
## S3 method for class 'dotprops'
plot(
  x,
  scalevecs = 1,
  alpharange = NULL,
  col = "black",
  PlotPoints = FALSE,
  PlotVectors = TRUE,
  UseAlpha = FALSE,
  asp = 1,
  add = FALSE,
  axes = TRUE,
  tck = NA,
  boundingbox = NULL,
  xlim = NULL,
  ylim = NULL,
  soma = FALSE,
  ...
)

## S3 method for class 'neuron'
plot(
  x,
  WithLine = TRUE,
  WithNodes = TRUE,
  WithAllPoints = FALSE,
  WithText = FALSE,
  PlotSubTrees = TRUE,
  soma = FALSE,
  PlotAxes = c("XY", "YZ", "XZ", "ZY"),
  axes = TRUE,
  asp = 1,
  main = x$NeuronName,
  sub = NULL,
  xlim = NULL,
  ylim = NULL,
```

```
  AxisDirections = c(1, -1, 1),
  add = FALSE,
  col = NULL,
  PointAlpha = 1,
  tck = NA,
  lwd = par("lwd"),
  boundingbox = NULL,
  ...
)
```

Arguments

x	a neuron to plot.
scalevecs	Factor by which to scale unit vectors (numeric, default: 1.0)
alpharange	Restrict plotting to points with alpha values in this range to plot (default: null => all points). See dotprops for definition of alpha.
col	the color in which to draw the lines between nodes.
PlotPoints, PlotVectors	Whether to plot points and/or tangent vectors (logical, default: tangent vectors only)
UseAlpha	Whether to scale tangent vector length by the value of alpha
asp	the y/x aspect ratio, see plot.window.
add	Whether the plot should be superimposed on one already present (default: FALSE).
axes	whether axes should be drawn.
tck	length of tick mark as fraction of plotting region (negative number is outside graph, positive number is inside, 0 suppresses ticks, 1 creates gridlines).
boundingbox	A 2 x 3 matrix (ideally of class boundingbox) that enables the plot axis limits to be set without worrying about axis selection or reversal (see details)
xlim	limits for the horizontal axis (see also boundingbox)
ylim	limits for the vertical axis (see also boundingbox)
soma	Whether to plot a circle at neuron's origin representing the soma. Either a logical value or a numeric indicating the radius (default FALSE). When soma=TRUE the radius is hard coded to 2.
...	additional arguments passed to plot
WithLine	whether to plot lines for all segments in neuron.
WithNodes	whether points should only be drawn for nodes (branch/end points)
WithAllPoints	whether points should be drawn for all points in neuron.
WithText	whether to label plotted points with their id.
PlotSubTrees	Whether to plot all sub trees when the neuron is not fully connected.
PlotAxes	the axes for the plot.
main	the title for the plot
sub	sub title for the plot

AxisDirections	the directions for the axes. By default, R uses the bottom-left for the origin, whilst most graphics software uses the top-left. The default value of c(1, -1, 1) makes the produced plot consistent with the latter.
PointAlpha	the value of alpha to use in plotting the nodes.
lwd	line width relative to the default (default=1).

Details

plot.dotprops is limited in that 1) it cannot plot somata directly (this is handled by plot.neuronlist) and 2) it can only plot a frontal (XY) view.

plot.neuron sets the axis ranges based on the chosen PlotAxes and the range of the data in x. It is still possible to use PlotAxes in combination with a boundingbox, for example to set the range of a plot of a number of objects.

nat assumes the default axis convention used in biological imaging, where the origin of the y axis is the top rather than the bottom of the plot. This is achieved by reversing the y axis of the 2D plot when the second data axis is the Y axis of the 3D data. Other settings can be achieved by modifying the AxisDirections argument.

Value

list of plotted points (invisibly)

See Also

plot3d.neuron

Other neuron: neuron(), ngraph(), potential_synapses(), prune(), resample(), rootpoints(), spine(), subset.neuron()

Examples

```
plot(kcs20[[1]], col='red')
# NB soma ignored
plot(kcs20[[1]], col='red', soma=TRUE)
plot(kcs20[1], col='red', soma=TRUE)
# Draw first example neuron
plot(Cell07PNs[[1]])
# Overlay second example neuron
plot(Cell07PNs[[2]], add=TRUE)
# Clear the current plot and draw the third neuron from a different view
plot(Cell07PNs[[3]], PlotAxes="YZ")
# Just plot the end points for the fourth example neuron
plot(Cell07PNs[[4]], WithNodes=FALSE)
# Plot with soma (of default radius)
plot(Cell07PNs[[4]], WithNodes=FALSE, soma=TRUE)
# Plot with soma of defined radius
plot(Cell07PNs[[4]], WithNodes=FALSE, soma=1.25)
```

`plot.neuronlist` *2D plots of the elements in a neuronlist, optionally using a subset expression*

Description

2D plots of the elements in a neuronlist, optionally using a subset expression

Usage

```
## S3 method for class 'neuronlist'
plot(
  x,
  subset = NULL,
  col = NULL,
  colpal = rainbow,
  add = NULL,
  boundingbox = NULL,
  soma = FALSE,
  ...,
  SUBSTITUTE = TRUE
)
```

Arguments

`x`	a neuron list or, for `plot3d.character`, a character vector of neuron names. The default neuronlist used by plot3d.character can be set by using `options(nat.default.neuronlist=` See ?nat for details.
`subset`	Expression evaluating to logical mask for neurons. See details.
`col`	An expression specifying a colour evaluated in the context of the dataframe attached to nl (after any subsetting). See details.
`colpal`	A vector of colours or a function that generates colours
`add`	Logical specifying whether to add data to an existing plot or make a new one. The default value of `NULL` creates a new plot with the first neuron in the neuronlist and then adds the remaining neurons.
`boundingbox`	A 2 x 3 matrix (ideally of class boundingbox) that enables the plot axis limits to be set without worrying about axis selection or reversal (see details)
`soma`	Whether to plot a sphere at neuron's origin representing the soma. Either a logical value or a numeric indicating the radius (default `FALSE`). When `soma=TRUE` the radius is hard coded to 2.
`...`	options passed on to plot (such as colours, line width etc)
`SUBSTITUTE`	Whether to `substitute` the expressions passed as arguments `subset` and `col`. Default: `TRUE`. For expert use only, when calling from another function.

Details

The col and subset parameters are evaluated in the context of the dataframe attribute of the neuronlist. If col evaluates to a factor and colpal is a named vector then colours will be assigned by matching factor levels against the named elements of colpal. If there is one unnamed level, this will be used as catch-all default value (see examples).

If col evaluates to a factor and colpal is a function then it will be used to generate colours with the same number of levels as are used in col.

Value

list of values of `plot` with subsetted dataframe as attribute `'df'`

See Also

nat-package, plot3d.neuronlist

Examples

```
# plot 4 cells
plot(Cell07PNs[1:4])
# modify some default plot arguments
plot(Cell07PNs[1:4], ylim=c(140,75), main='First 4 neurons')
# plot one class of neurons in red and all the others in grey
plot(Cell07PNs, col=Glomerulus, colpal=c(DA1='red', 'grey'), WithNodes=FALSE)
# subset operation
plot(Cell07PNs, subset=Glomerulus%in%c("DA1", "DP1m"), col=Glomerulus,
  ylim=c(140,75), WithNodes=FALSE)
```

`plot3d` *plot3d methods for different nat objects*

Description

These methods enable nat objects including neuronlists and dotprops objects to be plotted in 3D. See the help for each individual method for details along with the help for the generic in the rgl package.

See Also

plot3d, plot3d.boundingbox, plot3d.character, plot3d.cmtkreg, plot3d.dotprops, plot3d.hxsurf, plot3d.neuron, plot3d.neuronlist

Examples

```
# all known plot3d methods
methods("plot3d")

# up to date list of all plot3d nethods in this package
intersect(methods("plot3d"), ls(asNamespace("nat")))
```

plot3d.boundingbox *Plot a bounding box in 3D*

Description

Plot a bounding box in 3D

Usage

```
## S3 method for class 'boundingbox'
plot3d(
  x,
  col = "black",
  gridlines = FALSE,
  plotengine = getOption("nat.plotengine"),
  ...
)
```

Arguments

x	the boundingbox object to plot.
col	The colour of the bounding box lines (default 'black')
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE)
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.
...	additional arguments to pass to segments3d.

Value

A list of rgl object IDs (as returned by segments3d) **or** a plotly object.

See Also

boundingbox

Examples

```
# find the bounding box of all the neurons in a list
boundingbox(kcs20)
boundingbox(kcs20[1:3])

# plot those neurons
plot3d(kcs20)
# ... with their bounding box
plot3d(boundingbox(kcs20))

plot3d(kcs20)
```

```
# plot bounding box (in matching colours) for each neuron
# NB makes use of nlapply/neuronlist in slightly unsusual context -
# plot3d.neuronlist can cope with lists containing anything with
# a valid plot3d method.
plot3d(nlapply(kcs20,boundingbox))
```

`plot3d.cmtkreg` *Plot the domain of a CMTK registration*

Description

Plot the domain of a CMTK registration

Usage

```
## S3 method for class 'cmtkreg'
plot3d(x, ..., gridlines = FALSE, plotengine = getOption("nat.plotengine"))
```

Arguments

`x`	A cmtk registration (the path to the registration folder on disk) or the resulting of reading one in with `read.cmtkreg`.
`...`	Additional arguments passed to `plot3d`
`gridlines`	Whether to display gridlines when using plotly as the backend plotting engine (default: `FALSE`)
`plotengine`	the plotting backend engine to use either 'rgl' or 'plotly'.

See Also

`cmtkreg`, `read.cmtkreg`, `plot3d`

Examples

```
testdatadir=system.file("tests/testthat/testdata/cmtk", package="nat")
regpath=file.path(testdatadir,'FCWB_JFRC2_01_warp_level-01.list/')
# only run this if file is present (not always installed)
if(file.exists(regpath)){
plot3d(cmtkreg(regpath))

# or read registration into memory if you want to work with it
reg=read.cmtkreg(regpath)
plot3d(reg)
}
```

plot3d.dotprops *3D plots of dotprops objects using rgl package*

Description

3D plots of dotprops objects using rgl package

Usage

```
## S3 method for class 'dotprops'
plot3d(
  x,
  scalevecs = 1,
  alpharange = NULL,
  color = "black",
  PlotPoints = FALSE,
  PlotVectors = TRUE,
  UseAlpha = FALSE,
  ...,
  gridlines = FALSE,
  plotengine = getOption("nat.plotengine")
)
```

Arguments

x	A dotprops object
scalevecs	Factor by which to scale unit vectors (numeric, default: 1.0)
alpharange	Restrict plotting to points with alpha values in this range to plot (default: null => all points). See dotprops for definition of alpha.
color	Character or numeric vector specifying colours for points/vectors. See details.
PlotPoints, PlotVectors	Whether to plot points and/or tangent vectors (logical, default: tangent vectors only)
UseAlpha	Whether to scale tangent vector length by the value of alpha
...	Additional arguments passed to points3d and/or segments3d
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE)
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.

Details

Tangent vectors are plotted by segments3d and centred on the relevant point. Points are plotted by points3d.

color will be recycled by points3d and segments3d. However in the special case that color has length equal to the number of points in x, then it will be duplicated before being passed to

segments3d so that the result is that each vector is coloured uniformly according to color (since segments3d expects 2 colours for each line segment, blending them if they are different).

Value

invisible list of results of rgl plotting commands

See Also

dotprops, plot3d, points3d, segments3d

Examples

```
open3d()
plot3d(kcs20[[1]])
nclear3d()
plot3d(kcs20[[1]],col='red')
nclear3d()
plot3d(kcs20[[1]],col='red',lwd=2)
plot3d(kcs20[[2]],col='green',lwd=2)
```

---

plot3d.hxsurf *Plot amira surface objects in 3D using rgl*

Description

Plot amira surface objects in 3D using rgl

Usage

```
## S3 method for class 'hxsurf'
plot3d(
  x,
  materials = NULL,
  col = NULL,
  gridlines = FALSE,
  ...,
  plotengine = getOption("nat.plotengine")
)
```

Arguments

x	An hxsurf surface object
materials	Character vector or regex naming materials to plot (defaults to all materials in x). See subset.hxsurf.

col	Character vector specifying colors for the materials, or a function that will be called with the number of materials to plot. When NULL (default) will use material colours defined in Amira (if available), or rainbow otherwise.
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE)
...	Additional arguments passed to triangles3d
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.

See Also

read.hxsurf

Other hxsurf: as.hxsurf(), as.mesh3d(), materials(), read.hxsurf(), subset.hxsurf(), write.hxsurf()

Examples

```
plot3d(kcs20)
plot3d(MBL.surf)


# plot only vertical lobe
nclear3d()
plot3d(MBL.surf, materials="VL", alpha=0.3)

# everything except vertical lobe
nclear3d()
plot3d(MBL.surf, alpha=0.3,
  materials=grep("VL", MBL.surf$RegionList, value = TRUE, invert = TRUE))
```

---

plot3d.neuron *Plot neurons in 3D using rgl library or plotly module*

Description

Plot neurons in 3D using rgl library or plotly module

Usage

```
## S3 method for class 'neuron'
plot3d(
  x,
  WithLine = TRUE,
  NeuronNames = FALSE,
  WithNodes = TRUE,
  WithAllPoints = FALSE,
  WithText = FALSE,
```

```
  PlotSubTrees = TRUE,
  add = TRUE,
  col = NULL,
  soma = FALSE,
  ...,
  gridlines = FALSE,
  plotengine = getOption("nat.plotengine")
)
```

Arguments

x	A neuron to plot
WithLine	Whether to plot lines for all segments in neuron
NeuronNames	Logical indicating whether to label the neuron in the plot using the NeuronName field **or** a character vector of names.
WithNodes	Whether to plot dots for branch and end points
WithAllPoints	Whether to plot dots for all points in the neuron
WithText	Whether to label plotted points with their numeric id (see details)
PlotSubTrees	Whether to plot all sub trees when the neuron is not fully connected.
add	Whether to add the neuron to existing rgl plot rather than clearing the scene (default TRUE)
col	Colour specification (see rgl materials)
soma	Whether to plot a sphere at neuron's origin representing the soma. Either a logical value or a numeric indicating the radius (default FALSE). When soma=TRUE the radius is hard coded to 2.
...	Additional arguments passed to lines3d (and spheres3d if somata are being plotted).
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE)
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.

Details

Note that when WithText=TRUE, the numeric identifiers plotted are *raw indices* into the x$d array of the neuron, *not* the values of the PointNo column.

Note that ... is passed to both lines3d and spheres3d (if somata are being plotted). Not all ... elements are necessarily relevant to both of these drawing calls. Furthermore plotting a large number of somata with transparency (i.e. alpha < 1) can quickly result in very slow rgl draw and refresh speeds; you will likely want to set skipRedraw=FALSE when using plot3d.neuronlist to plot a collection of neurons.

Value

list of rgl plotting ids (invisibly) separated into lines, points, texts according to plot element. See rgl::plot3d for details.

See Also

plot3d.neuronlist, plot3d.dotprops, nat::plot3d, rgl::plot3d

Examples

```
# A new plot would have been opened if required
open3d()
plot3d(Cell07PNs[[1]],col='red')
plot3d(Cell07PNs[[2]],col='green')

# clear the current plot
nclear3d()
plot3d(Cell07PNs[[2]],col='blue',add=FALSE)
# plot the number of all nodes
nclear3d()
plot3d(Cell07PNs[[2]],col='red',WithText=TRUE,add=FALSE)
# include cell bodies
plot3d(Cell07PNs[3:4], col='red', soma=TRUE)
plot3d(Cell07PNs[5], col='red', soma=3)
```

---

plot3d.neuronlist *3D plots of the elements in a neuronlist, optionally using a subset expression*

Description

plot3d.character is a convenience method intended for exploratory work on the command line.

Usage

```
## S3 method for class 'neuronlist'
plot3d(
  x,
  subset = NULL,
  col = NULL,
  colpal = rainbow,
  skipRedraw = TRUE,
  add = TRUE,
  WithNodes = FALSE,
  soma = FALSE,
  ...,
  SUBSTITUTE = TRUE,
  gridlines = FALSE,
  plotengine = getOption("nat.plotengine")
)

## S3 method for class 'character'
plot3d(x, db = NULL, ...)
```

Arguments

x	a neuron list or, for plot3d.character, a character vector of neuron names. The default neuronlist used by plot3d.character can be set by using options(nat.default.neuronlist= See ?nat for details. nat-package.
subset	Expression evaluating to logical mask for neurons. See details.
col	An expression specifying a colour evaluated in the context of the dataframe attached to nl (after any subsetting). See details.
colpal	A vector of colours or a function that generates colours
skipRedraw	By default TRUE which is much faster when plotting large numbers of neurons). Can also accept FALSE (never skip) or integers specifying a threshold number of neurons, above which redrawing is skipped.
add	Whether to add the neuron to existing rgl plot rather than clearing the scene (default TRUE)
WithNodes	Whether to plot points for end/branch points. Default: FALSE.
soma	Whether to plot a sphere at neuron's origin representing the soma. Either a logical value or a numeric indicating the radius (default FALSE). When soma=TRUE the radius is hard coded to 2.
...	options passed on to plot3d (such as colours, line width etc)
SUBSTITUTE	Whether to substitute the expressions passed as arguments subset and col. Default: TRUE. For expert use only, when calling from another function.
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE)
plotengine	the plotting backend engine to use either 'rgl' or 'plotly'.
db	A neuronlist to use as the source of objects to plot. If NULL, the default, will use the neuronlist specified by options('nat.default.neuronlist')

Details

The col and subset parameters are evaluated in the context of the dataframe attribute of the neuronlist. If col evaluates to a factor and colpal is a named vector then colours will be assigned by matching factor levels against the named elements of colpal. If there is one unnamed level, this will be used as catch-all default value (see examples).

If col evaluates to a factor and colpal is a function then it will be used to generate colours with the same number of levels as are used in col.

WithNodes is FALSE by default when using plot3d.neuronlist but remains TRUE by default when plotting single neurons with plot3d.neuron. This is because the nodes quickly make plots with multiple neurons rather busy.

When soma is TRUE or a vector of numeric values (recycled as appropriate), the values are used to plot cell bodies. For neurons the values are passed to plot3d.neuron for neurons. In contrast dotprops objects still need special handling. There must be columns called X,Y,Z in the data.frame attached to x, that are then used directly by code in plot3d.neuronlist.

Whenever plot3d.neuronlist is called, it will add an entry to an environment .plotted3d in nat that stores the ids of all the plotted shapes (neurons, cell bodies) so that they can then be removed by a call to npop3d.

plot3d.character will check if options('nat.default.neuronlist') has been set and then use x as an identifier to find a neuron in that neuronlist.

Value

list of values of `plot3d` with subsetted dataframe as attribute `'df'`

See Also

nat-package

Examples

```
open3d()
plot3d(kcs20,type=='gamma',col='green')

nclear3d()
plot3d(kcs20,col=type)

nclear3d()
plot3d(Cell07PNs,Glomerulus=="DA1",col='red')
plot3d(Cell07PNs,Glomerulus=="VA1d",col='green')

# Note use of default colour for non DA1 neurons
nclear3d()
plot3d(Cell07PNs,col=Glomerulus, colpal=c(DA1='red', 'grey'))

# a subset expression
nclear3d()
plot3d(Cell07PNs,Glomerulus%in%c("DA1",'VA1d'),
  col=c("red","green")[factor(Glomerulus)])
# the same but not specifying colours explicitly
nclear3d()
plot3d(Cell07PNs,Glomerulus%in%c("DA1",'VA1d'),col=Glomerulus)

## Not run:
## more complex colouring strategies for a larger neuron set
# see https://github.com/jefferis/frulhns for details
library(frulhns)
# notice the sexually dimorphic projection patterns for these neurons
plot3d(jkn,cluster=='aSP-f' &shortGenotype=='JK1029',
  col=sex,colpal=c(male='green',female='magenta'))

## colour neurons of a class by input resistance
jkn.aspg=subset(jkn, cluster=='aSP-g')
# NB this comes in as a factor
Ri=with(jkn.aspg,as.numeric(as.character(Ri..GOhm.)))
# the matlab jet palette
jet.colors<-colorRampPalette(c('navy','cyan','yellow','red'))
plot3d(jkn.aspg,col=cut(Ri,20),colpal=jet.colors)

## End(Not run)
```

plot3d.ngraph *Plot 3d representation of neuron (ngraph) with directed edges*

Description

Plot 3d representation of neuron (ngraph) with directed edges

Usage

```
## S3 method for class 'ngraph'
plot3d(x, type = "lines", soma = 1, labels = c("none", "nodes", "all"), ...)
```

Arguments

x	A ngraph object
type	They type of arrows (lines by default, see arrow3d for details).
soma	radius of soma (or FALSE to suppress plotting)
labels	Whether to label nodes/all points with their raw index (not id)
...	Additional arguments passed to arrow3d

Examples

```
plot3d(as.ngraph(Cell07PNs[[1]]), labels='nodes')
```

pointsinside *Find which points of an object are inside a surface*

Description

Find which points of an object are inside a surface

Usage

```
pointsinside(x, surf, ...)

## Default S3 method:
pointsinside(
  x,
  surf,
  ...,
  rval = c("logical", "distance", "mesh3d", "consistent_logical")
)
```

Arguments

x	an object with 3D points.
surf	The reference surface - either a mesh3d object or any object that can be converted using as.mesh3d including hxsurf and ashape3d objects.
...	additional arguments for methods, eventually passed to as.mesh3d.
rval	what to return.

Details

Note that hxsurf surface objects will be converted to mesh3d before being passed to Rvcg::vcgClostKD, so if you are testing repeatedly against the same surface, it may make sense to pre-convert.

pointsinside depends on the face normals for each face pointing out of the object (see example). The face normals are defined by the order of the three vertices making up a triangular face. You can flip the face normal for a face by permuting the vertices (i.e. 1,2,3 -> 1,3,2). If you find for a given surface that points are outside when you expect them to be inside then the face normals are probably all the wrong way round. You can invert them yourself or use the Morpho::invertFaces function to fix this.

The rval argument determines the return value. These options should be fairly clear, but the difference between logical and consistent_logical needs some explanation. The logical method now does a pre-test to remove any points that are not in the 3D bounding box (cuboid) enclosing the surf object. This often results in a significant speed-up by rejecting distant points and has the additional benefit of rejecting distant points that sometimes are erroneously declared inside the mesh (see below). Regrettably it is not yet possible to extend this approach when distances are being returned, which means there will be a discrepancy between the results of rval="logical" and looking for points with distance >=0. If you want to ensure consistency between these approaches, use rval="consistent_logical".

If you find that some points but not all points are not behaving as you would expect, then it may be that some faces are not coherently oriented. The Rvcg::vcgClean function can sometimes be used to correct the orientation of the faces. Fixing more problematic cases may be possible by generating a new surface using alphashape3d::ashape3d (see examples).

Value

A vector of logical values or distances (positive inside, negative outside) equal to the number of points in x or the mesh3d object returned by Rvcg::vcgClostKD.

Examples

```
# check if the vertices in these neurons are inside the mushroom body calyx
# surface object
inout=pointsinside(kcs20, surf=subset(MBL.surf, "MB_CA_L"))
table(inout)
# you can also check if points are inside a bounding box
mbcalbb=boundingbox(subset(MBL.surf, "MB_CA_L"))
inout2=pointsinside(kcs20, mbcalbb)
# compare those two
table(inout, inout2)
pts=xyzmatrix(kcs20)
```

```
# nb that colour expressions maps combinations of two logicals onto 1:4
plot(pts[,1:2], col=1+inout+inout2*2)
# the colours are defined by
palette()[1:4]

# be a bit more lenient and include points less than 5 microns from surface
MBCAL=subset(MBL.surf, "MB_CA_L")
inout5=pointsinside(kcs20, surf=MBCAL, rval='distance') > -5
table(inout5)

# show which points are in or out
# Hmm seems like there are a few red points in the vertical lobe
# that are well outside the calyx
points3d(xyzmatrix(kcs20), col=ifelse(inout5, 'red', 'black'))
plot3d(MBL.surf, alpha=.3)

# Let's try to make an alphashape for the mesh to clean it up
library(alphashape3d)
MBCAL.as=ashape3d(xyzmatrix(MBCAL), alpha = 10)
# Plotting the points, we can see that is much better behaved
points3d(xyzmatrix(kcs20),
  col=ifelse(pointsinside(kcs20, MBCAL.as), 'red', 'black'))


## Not run:
# Show the face normals for a surface
if(require('Morpho')) {
  # convert to a mesh3d object used by rgl and Morpho packge
  MBCAL.mesh=as.mesh3d(subset(MBL.surf, "MB_CA_L"))
  fn=facenormals(MBCAL.mesh)
  wire3d(MBCAL.mesh)
  # show that the normals point out of the object
  plotNormals(fn, long=5, col='red')

  # invert the faces of the mesh and show that normals point in
  MBCAL.inv=invertFaces(MBCAL.mesh)
  plotNormals(facenormals(MBCAL.inv), long=5, col='cyan')
}

## End(Not run)
```

`potential_synapses` *Calculate number of potential synapses between two neurons*

Description

This implements the method of Stepanyants and Chklovskii

Usage

```
potential_synapses(a, b, s, ...)

## S3 method for class 'neuronlist'
potential_synapses(a, b, s, ...)

## S3 method for class 'neuron'
potential_synapses(
  a,
  b,
  s,
  sigma = s,
  bounds,
  method = c("direct", "approx"),
  ...
)

## S3 method for class 'dotprops'
potential_synapses(
  a,
  b,
  s,
  sigma = s,
  seglength = 1,
  bounds = NULL,
  method = c("direct", "approx"),
  ...
)
```

Arguments

a, b	neurons or neuronlists
s	the approach distance to consider a potential synapse
...	Additional arguments passed to methods (see details)
sigma	the smoothing parameter in the approximate method (see details)
bounds	Optional bounding box to restrict comparison
method	Whether to use the direct or approximate method (see details)
seglength	how long to consider each distance between points.

Details

Note that potential_synapses.neuronlist uses nlapply to process its first argument (a). This enables progress bars, robustness to errors and simple parallel execution. See the nlapply examples for further details of these arguments in action.

For this reason if you have two neuronlists of unequal sizes, it is recommended to put the larger one in argument a.

References

Neurogeometry and potential synaptic connectivity. Stepanyants A, Chklovskii DB. Trends Neurosci. 2005 Jul;28(7):387-94. doi:10.1016/j.tins.2005.05.006

See Also

Other neuron: `neuron()`, `ngraph()`, `plot.dotprops()`, `prune()`, `resample()`, `rootpoints()`, `spine()`, `subset.neuron()`

Examples

```
potential_synapses(Cell07PNs[1], Cell07PNs[1:3], s=2)
## Not run:
# if you have many neurons to calculate you should get a progress bar
potential_synapses(Cell07PNs[1:10], Cell07PNs[11:20], s=2)

# you can also use parallel execution, here over 7 cores
# doMC::registerDoMC(7)
potential_synapses(Cell07PNs[1:10], Cell07PNs[11:20], s=2, .parallel=TRUE)

## End(Not run)
```

---

projection *Make 2D (orthogonal) projection of 3D image data*

Description

Make 2D (orthogonal) projection of 3D image data

Usage

```
projection(
  a,
  projdim = "z",
  projfun = c("integrate", "mean", "sum"),
  na.rm = T,
  mask = NULL,
  ...
)
```

Arguments

a	Array of image data (im3d format)
projdim	The image dimension down which to project
projfun	The function that collapses each vector of image data down to a single pixel. Can be a character vector naming a function or a function. See details.

na.rm	Logical indicating whether to ignore NA values in the image data when calculating function results. default: TRUE
mask	A mask with the same extent as the image.
...	Additional arguments for projfun

Details

Note that projfun must have an argument na.rm like the S3 Summary groupGeneric functions such as sum, min etc.

Note also that the BoundingBox of a 2d projection is not well-defined for the axis along which the projection was made. Presently both the evaluation location and the BoundingBox extremes are set to 0 after a projection is made but FIXME this is not completely satisfactory. Perhaps defining this to be NA or the midpoint of the original axis would be better justified.

See Also

groupGeneric, clampmax

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), threshold(), unmask(), voxdims()

Examples

```
## Not run:
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd',package='nat'))
d=unmask(rnorm(sum(LHMask),mean=5,sd=5),LHMask)
op=par(mfrow=c(1,2))
rval=image(projection(d,projfun=max))
image(projection(d,projfun=clampmax(0,10)),zlim=rval$zlim)
par(op)

## End(Not run)
## Not run:
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd',package='nat'))
image(projection(LHMask),asp=TRUE)

## End(Not run)
```

---

prune *prune an object by removing points near (or far) from a target object*

Description

prune an object by removing points near (or far) from a target object

Usage

```
prune(x, target, ...)

## S3 method for class 'neuron'
prune(x, target, ...)

## S3 method for class 'dotprops'
prune(x, target, ...)

## S3 method for class 'neuronlist'
prune(x, target, ...)

## Default S3 method:
prune(x, target, maxdist, keep = c("near", "far"), return.indices = FALSE, ...)
```

Arguments

x	The object to prune. (e.g. dotprops object, see details)
target	Another object with 3D points that will determine which points in x are kept.
...	Additional arguments for methods (eventually passed to prune.default)
maxdist	The threshold distance for keeping points
keep	Whether to keep points in x that are near or far from the target
return.indices	Whether to return the indices that pass the test rather than the 3D object/points (default FALSE)

Details

prune.neuron depends on a more basic function prune_vertices and is also related to subset.neuron.

See Also

prune_strahler, spine, prune_vertices, subset.neuron

subset.neuron

subset.dotprops

Other neuron: neuron(), ngraph(), plot.dotprops(), potential_synapses(), resample(), rootpoints(), spine(), subset.neuron()

Examples

```
## prune single neurons

plot3d(kcs20[[1]],col='blue')
plot3d(kcs20[[2]],col='red')

# prune neuron 2 down to points that are close to neuron 1
neuron2_close=prune(kcs20[[2]], target=kcs20[[1]], maxdist=10)
```

```
plot3d(neuron2_close, col='cyan', lwd=3)

neuron2_far=prune(kcs20[[2]], target=kcs20[[1]], maxdist=10, keep='far')

plot3d(neuron2_far, col='magenta', lwd=3)


## Prune a neuron with a neuronlist
pruned=prune(kcs20[[11]], kcs20[setdiff(1:20, 11)], maxdist=8)

plot3d(pruned, col='red', lwd=3)
plot3d(kcs20[[11]], col='green', lwd=3)
plot3d(kcs20,col='grey')
```

`prune_in_volume` *Prune neuron(s) within a volume defined by a 3D mesh*

Description

Prune neuron(s) within a volume defined by a 3D mesh

Usage

```
prune_in_volume(x, surf, neuropil = NULL, invert = TRUE, ...)

## S3 method for class 'neuron'
prune_in_volume(x, surf, neuropil = NULL, invert = TRUE, ...)

## S3 method for class 'neuronlist'
prune_in_volume(x, surf, neuropil = NULL, invert = TRUE, ...)
```

Arguments

`x`	a neuron object
`surf`	An `hxsurf` or `mesh3d` object, or any object coercible into `mesh3d` by `as.mesh3d`.
`neuropil`	Character vector specifying a subset of the `surf` object. This is only relevant when `surf` is of class `hxsurf`. If NULL (default), then the full object given as `surf` will be used for the pruning.
`invert`	Logical when `TRUE` indicates that points inside the mesh are kept.
`...`	Additional arguments for methods (eventually passed to `prune_vertices`) surface should be pruned.

Details

Prune a neuron to be within, or to exclude arbour within, a 3D object that can be coerced into the mesh3d data structure

Value

A pruned neuron/neuronlist object

See Also

as.neuron.ngraph, subset.neuron, prune.neuron, prune

Examples

```
## Not run:
### Example requires the package nat.flybrains
LH_arbour = prune_in_volume(x = Cell07PNs, surf = nat.flybrains::IS2NP.surf,
  neuropil = "LH_L", OmitFailures = TRUE)

## End(Not run)
```

---

prune_online *Prune a neuron interactively in an rgl window*

Description

Remove points from a neuron, keeping the root node intact.

Usage

```
prune_online(x, ...)

## S3 method for class 'neuron'
prune_online(x, ...)

## S3 method for class 'neuronlist'
prune_online(x, ...)
```

Arguments

x	The object to prune. (e.g. dotprops object, see details)
...	Additional methods passed to prune_vertices

Details

The neuron is plotted initially with all nodes selected (and shown with black points). You can interactively select points to remove (they will now be plotted in red). You can also add points back again (they will return to black). When you are finished, press [e] to exit and then indicate that you have finished (yes).

Value

A pruned neuron/neuronlist object

See Also

as.neuron.ngraph, subset.neuron, prune.neuron

Examples

```
## Not run:
## Interactively choose which bit of the neuron you wish to keep
pruned.as.you.like.it = prune_online(Cell07PNs[1:2])

## End(Not run)
```

---

prune_strahler *Prune a neuron by removing segments with a given Strahler order*

Description

Prune a neuron by removing segments with a given Strahler order

Usage

```
prune_strahler(x, orderstoprune = 1:2, ...)
```

Arguments

x	A neuron
orderstoprune	Integer indices of which Strahler orders to prune - defaults to the lowest two orders (1:2)
...	Additional arguments passed to as.neuron.data.frame

Value

The pruned neuron

See Also

strahler_order, spine, for finding the longest path in a neuron, prune for subsetting dotprops style neurons by spatial proximity, as.neuron.data.frame, which is used to generate the new neuron.

Examples

```
x=Cell07PNs[[1]]
pruned12=prune_strahler(x)
pruned1=prune_strahler(x, 1)
plot(x)
plot(pruned1, lwd=3, col='blue', add=TRUE)
plot(pruned12, lwd=3, col='red', add=TRUE)
```

prune_twigs *Remove all twigs less than a certain path length from a neuron*

Description

prune_twigs will prune twigs less than a certain path length from a neuron

Usage

```
prune_twigs(x, ...)

## S3 method for class 'neuron'
prune_twigs(x, twig_length, ...)

## S3 method for class 'neuronlist'
prune_twigs(x, twig_length, OmitFailures = NA, ...)
```

Arguments

x	A neuron or neuronlist object
...	Additional arguments passed to nlapply, prune_vertices and eventually as.ngraph.
twig_length	Twigs shorter than this will be pruned
OmitFailures	Whether to omit neurons for which FUN gives an error. The default value (NA) will result in nlapply stopping with an error message the moment there is an error. For other values, see details.

Author(s)

Gregory Jefferis <jefferis@gmail.com>

Examples

```
# Prune twigs up to 5 microns long
pt5=prune_twigs(Cell07PNs[1:3], twig_length = 5)
# compare original (coloured) and pruned (black) neurons
plot(Cell07PNs[1:3], WithNodes=FALSE, lwd=2, xlim=c(240,300), ylim=c(120, 90))
plot(pt5, WithNodes=FALSE, add=TRUE, lwd=2, col='black')
```

prune_vertices *Prune selected vertices or edges from a neuron*

Description

prune_vertices removes vertices from a neuron

prune_edges removes edges (and any unreferenced vertices)

Usage

```
prune_vertices(x, verticestoprune, invert = FALSE, ...)

prune_edges(x, edges, invert = FALSE, ...)
```

Arguments

x	A neuron to prune. This can be any object that can be converted by as.ngraph — see details.
verticestoprune	An integer vector describing which vertices to remove.
invert	Whether to keep vertices rather than dropping them (default FALSE).
...	Additional arguments passed to as.neuron.ngraph
edges	The edges to remove. One of i) an Nx2 matrix, each row specifying a single edge defined by its **raw** edge id, ii) an integer vector defining a *path* of raw vertex ids or iii) an igraph.es edge sequence — see details and the P and path arguments of igraph::E.

Details

These are relatively low-level functions and you will probably want to use subset.neuron or prune.neuron and friends in many cases.

Note that prune_vertices and prune_edges both use **raw** vertex ids to specify the vertices/edges to be removed. If you want to use the id in the PointNo field, then you must translate yourself (see examples).

Both prune_vertices and prune_edges first convert their input x to the ngraph representation of the neuron before removing points. The input x can therefore be in any form compatible with as.ngraph including an existing ngraph. There is an additional requirement that the input must be compatible with xyzmatrix if invert=TRUE.

Note that the edges argument of prune_edges must specify a path traversing a set of vertices in a valid order. However if the input is a matrix or vector the direction of each individual edge in this path is ignored. So if your neuron has edges 2->1 2->3 3->4 then an edge sequence 1:3 would successfully delete 2 edges.

Value

A pruned neuron

See Also

as.neuron.ngraph, subset.neuron, prune.neuron

Examples

```
n=prune_vertices(Cell07PNs[[1]], 1:25)
# original neuron
plot(Cell07PNs[[1]])
# with pruned neuron superimposed
plot(n, col='green', lwd=3, add=TRUE)

# use the PointNo field (= the original id from an SWC file)
n2=prune_vertices(n, match(26:30, n$d$PointNo))
y=prune_edges(Cell07PNs[[1]], edges=1:25)

# remove the spine of a neuron
spine_ids=spine(Cell07PNs[[1]], rval='ids')
pruned=prune_edges(Cell07PNs[[1]], spine_ids)

# NB this is subtly different from this, which removes vertices along the
# spine *even* if they are part of an edge that is outside the spine.
pruned2=prune_vertices(Cell07PNs[[1]], spine_ids)
```

---

read.amiramesh *Read AmiraMesh data in binary or ascii format*

Description

Read AmiraMesh data in binary or ascii format

Read the header of an AmiraMesh file

Usage

```
read.amiramesh(
  file,
  sections = NULL,
  header = FALSE,
  simplify = TRUE,
  endian = NULL,
  ReadByteAsRaw = FALSE,
  Verbose = FALSE
)

read.amiramesh.header(file, Parse = TRUE, Verbose = FALSE)
```

Arguments

file	Name of file (or connection) to read
sections	character vector containing names of sections
header	Whether to include the full unprocessed text header as an attribute of the returned list.
simplify	If there is only one datablock in file do not return wrapped in a list (default TRUE).
endian	Whether multibyte data types should be treated as big or little endian. Default of NULL checks file or uses .Platform$endian
ReadByteAsRaw	Logical specifying whether to read 8 bit data as an R raw vector rather than integer vector (default: FALSE).
Verbose	Print status messages
Parse	Logical indicating whether to parse header (default: TRUE)

Details

reading byte data as raw arrays requires 1/4 memory but complicates arithmetic.

read.amiramesh.header will open a connection if file is a character vector and close it when finished reading.

Value

list of named data chunks

See Also

readBin, .Platform

Other amira: amiratype(), is.amiramesh(), read.hxsurf(), write.hxsurf()

---

read.cmtk *Read CMTK TypedStream file to a list in memory*

Description

This function is primarily of developer interest. End users will typically want to use more specialised functions for reading registrations and landmarks.

Usage

```
read.cmtk(con, CheckLabel = TRUE)
```

Arguments

con	Path to (optionally gzipped) file or (open) connection.
CheckLabel	Check, fix and warn for invalid or duplicate labels (default TRUE)

Details

This is the default format used by CMTK for registration, studylist, landmarks and image files. Although this is largely a generic function, there is special handling of the coefficients and active members of the spline warp component of a CMTK nonrigid registration.

Note that if an open connection is passed to read.cmtk the version number of the CMTK TypedStream will not be checked or recorded.

See Also

Other cmtk-io: `cmtk.extract_affine()`, `read.cmtkreg()`, `write.cmtkreg()`, `write.cmtk()`

`read.cmtkreg` *Read a CMTK format registration*

Description

Read a CMTK format registration

Usage

```
read.cmtkreg(filename, ReturnRegistrationOnly = FALSE, ...)
```

Arguments

`filename`	Path to a CMTK registration file
`ReturnRegistrationOnly`	When FALSE (default) will not attempt to extract the registration element from the registration file.
`...`	Additional arguments passed to read.cmtk

See Also

Other cmtk-io: `cmtk.extract_affine()`, `read.cmtk()`, `write.cmtkreg()`, `write.cmtk()`

`read.hxsurf` *Read Amira surface (aka HxSurface or HyperSurface) files into hxsurf object*

Description

Read Amira surface (aka HxSurface or HyperSurface) files into hxsurf object

Usage

```
read.hxsurf(
  filename,
  RegionNames = NULL,
  RegionChoice = "both",
  FallbackRegionCol = "grey",
  Verbose = FALSE
)
```

Arguments

`filename`	Character vector defining path to file
`RegionNames`	Character vector specifying which regions should be read from file. Default value of `NULL` => all regions.
`RegionChoice`	Whether the *Inner* or *Outer* material, or *both* (default), should define the material of the patch. See details.
`FallbackRegionCol`	Colour to set regions when no colour is defined
`Verbose`	Print status messages during parsing when `TRUE`

Details

Note that when `RegionChoice="both"` or `RegionChoice=c("Inner", "Outer")` both polygons in inner and outer regions will be added to named regions. To understand the significance of this, consider two adjacent regions, A and B, with a shared surface. For the polygons in both A and B, Amira will have a patch with (say) InnerRegion A and OuterRegion B. This avoids duplication in the file. However, it might be convenient to add these polygons to both regions when we read them into R, so that regions A and B in our R object are both closed surfaces. To achieve this when `RegionChoice="both"`, `read.hxsurf` adds these polygons to region B (as well as region A) but swaps the order of the vertices defining the polygon to ensure that the surface directionality is correct.

As a rule of thumb, stick with `RegionChoice="both"`. If you get more regions than you wanted, then try switching to `RegionChoice="Inner"` or `RegionChoice="Outer"`.

Note that the support for reading Amira's binary mesh format (HxSurface binary) is less mature and in particular only a few multi region mesh files have been tested. Finally there is no support to read meshes from the newer "Amira Binary Surface format" although such files can be read into a list using the `read.amiramesh` function.

Value

A list with S3 class hxsurf with elements

Vertices A data.frame with columns `X`, `Y`, `Z`, `PointNo`

Regions A list with 3 column data.frames specifying triplets of vertices for each region (with reference to `PointNo` column in `Vertices` element)

RegionList Character vector of region names (should match names of `Regions` element)

RegionColourList Character vector specifying default colour to plot each region in R's `rgb` format

See Also

`plot3d.hxsurf`, `rgb`

Other amira: `amiratype()`, `is.amiramesh()`, `read.amiramesh()`, `write.hxsurf()`

Other hxsurf: `as.hxsurf()`, `as.mesh3d()`, `materials()`, `plot3d.hxsurf()`, `subset.hxsurf()`, `write.hxsurf()`

Examples

```
## Not run:
read.hxsurf("my.surf", RegionChoice="both")

## End(Not run)
```

---

read.landmarks *Generic functions to read/write landmarks in any supported format*

Description

Generic functions to read/write landmarks in any supported format

Usage

```
read.landmarks(f, ...)

write.landmarks(
  x,
  file,
  format = "amiralandmarks",
  ext = NULL,
  Force = FALSE,
  MakeDir = TRUE,
  ...
)
```

Arguments

f	Path to a file (can also be a URL)
...	Additional arguments passed on to format specific functions
x	The landmarks object to write. Can also be a plain matrix or data.frame.
file	The path to the output file. If this does not end in an extension like .landmarksAscii, then one will be added based on the value of the ext argument.
format	Character vector specifying output format. Defaults to "amiralandmarks". Partial matching is used (e.g. amira is sufficient).
ext	Optional character vector specifying a new or non-standard extension to use for output file, including the period (e.g. ext='.am'). When ext=NULL, the default, the default extension for the selected format will be added if f does not have an extension. When ext=NA, the extension will not be modified and no extension will be appended if f does not have one.
Force	Whether to overwrite an existing file
MakeDir	Whether to create directory implied by file argument.

Details

Presently the supported formats are

- Amira (format name amiralandmarks)
- CMTK (format name cmtklandmarks)
- Fiji (format name fijilandmarks) see https://imagej.net/plugins/name-landmarks-and-register

See examples section for how to produce a listing of all currently available formats with fileformats.

Value

for read.landmarks a matrix or list of additional class landmarks, where the rownames specify the names of each landmark if available.

For write.landmarks the path to the written file, invisibly.

Paired landmarks

Only the amiralandmarks format supports the use of paired landmarks

See Also

fileformats

Examples

```
## Listing of supported fileformats for landmarks
fileformats(class = 'landmarks', rval = "info")

## round trip tests
m=matrix(rnorm(6), ncol=3)
```

```
rownames(m)=c("nose", "ear")
f=write.landmarks(m, file='knee', format='cmtk')
read.landmarks(f)

# write in amira format which does not support named landmarks
f2=write.landmarks(m, file='knee', format='amira')
read.landmarks(f2)

# clean up
unlink(c(f,f2))
```

`read.morphml` *Return parsed XML or R list versions of a NeuroML file*

Description

`read.morphml` is designed to expose the full details of the morphology information in a NeuroML file either as a parsed XML structure processed by the `XML` package *or* as an extensively processed R list object. To obtain a `neuron` object use `read.neuron.neuroml`.

Usage

```
read.morphml(f, ..., ReturnXML = FALSE)
```

Arguments

`f`	Path to a file on disk or a remote URL (see `xmlParse` for details).
`...`	Additional arguments passed to `xmlParse`
`ReturnXML`	Whether to return a parsed XML tree (when `ReturnXML=TRUE`) or a more extensively processed R list object when `ReturnXML=FALSE`, the default.

Details

NeuroML files consist of an XML tree containing one more or more **cells**. Each **cell** contains a tree of **segments** defining the basic connectivity/position and an optional tree **cables** defining attributes on groups of **segments** (e.g. a name, whether they are axon/dendrite/soma etc).

`read.morphml` will either provide the parsed XML tree which you can query using XPath statements or a more heavily processed version which provides as much information as possible from the segments and cables trees in two R data.frames. The latter option will inevitably drop some information, but will probably be more convenient for most purposes.

Value

Either an R list of S3 class containing one `morphml_cell` object for every cell in the NeuroML document or an object of class `XMLDocument` when `ReturnXML=TRUE`.

References

https://docs.neuroml.org/Userdocs/Specification.html

See Also

link[XML]{xmlParse}, read.neuron.neuroml

---

read.neuron *Read a single neuron from a file*

Description

Read a single neuron from a file

Usage

```
read.neuron(f, format = NULL, class = c("neuron", "ngraph"), ...)
```

Arguments

f	Path to file. This can be a URL, in which case the file is downloaded to a temporary location before reading.
format	The file format of the neuron. When format=NULL, the default, read.neuron will infer the file format from the extension or file header (aka magic) using the fileformats registry.
class	The class of the returned object - presently either "neuron" or "ngraph"
...	additional arguments passed to format-specific readers

Details

This function will handle neuron and dotprops objects saved in R .rds or .rda format by default. Additional file formats can be registered using fileformats.

At the moment the following formats are supported using file readers already included with the nat package:

- **swc** See read.neuron.swc. SWC files can also return an ngraph object containing the neuron structure in a (permissive) general graph format that also contains the 3D positions for each vertex.
- **neuroml** See read.neuron.neuroml
- **fijitraces** See read.neuron.fiji. The file format used by the SNT plugin of Fiji/ImageJ.
- **hxlineset,hxskel** Two distinct fileformats used by Amira. hxlineset is the generic one, hxskel is used by the hxskeletonize extension of Schmitt and Evers (see refs).
- **rda,rds** Native R cross-platform binary formats (see load, readRDS). Note that RDS only contains a single unnamed neuron, whereas rda contains one or more named neurons.
- **obj,ply** 3D Mesh formats encoding surface models of neurons. These depend on the suggested package Rvcg (for 'ply' format) and readobj (for Wavefront 'obj' format).

References

Schmitt, S. and Evers, J. F. and Duch, C. and Scholz, M. and Obermayer, K. (2004). New methods for the computer-assisted 3-D reconstruction of neurons from confocal image stacks. Neuroimage 4, 1283–98. doi:10.1016/j.neuroimage.2004.06.047

See Also

write.neuron, read.neurons, fileformats

Examples

```
## Not run:
# note that we override the default NeuronName field
n=read.neuron(system.file("tests/testthat/testdata","neuron","EBT7R.CNG.swc",package='nat'),
  NeuronName="EBT7R")
# use a function to set the NeuronName field
n3=read.neuron(system.file("tests/testthat/testdata","neuron","EBT7R.CNG.swc",package='nat'),
  NeuronName=function(x) sub("\\..*","",x))
# show the currently registered file formats that we can read
fileformats(class='neuron', read=TRUE)

## End(Not run)
```

---

read.neuron.fiji *Read a neuron saved by Fiji's Simple Neurite Tracer Plugin*

Description

Read a neuron saved by Fiji's Simple Neurite Tracer Plugin

Usage

```
read.neuron.fiji(
  f,
  ...,
  simplify = TRUE,
  components = c("path", "fill"),
  Verbose = FALSE
)
```

Arguments

f	Path to a file
...	Additional arguments passed to xmlParse.
simplify	Whether to return a single neuron as a neuron object rather than a neuronlist of length 1.
components	Which components to read in (path or fill). Only paths are properly supported at present (see details).
Verbose	Whether to print status messages during parsing.

Details

simple neurite tracer .traces files are an XML based format so parsing it depends on installation of the suggested XML package.

They can contain both paths (skeleton lines) and fill information (saved as XYZ coordinates of voxels inside the object). The latter cannot currently be handled very well by `read.neuron`. If you wish to access them you will probably need to use the private `read.fijixml` function to do so (see examples).

Your best best if you want to produce a fully 3D object with "width" information would be to generate a 3D mesh using Fiji's 3D viewer. You can do this by selecting the object in the viewer and the choosing `File ... Export Surface ... Wavefront` *while the 3D viewer window is active*. The resultant obj file can then be read in by `read.neurons`. You could use this mesh to find radius information for a skeleton by shooting rays from skeleton to mesh to estimate the radius.

References

https://imagej.net/plugins/snt/ https://imagej.net/plugins/snt/extending

Examples

```
## Not run:
n=read.neuron.fiji("my.traces")
plot3d(n)
fill=read.neuron.fiji("my.traces", components='fill')
points3d(fill, col='grey')

## End(Not run)
```

`read.neuron.neuroml` *Read one or more neurons from a NeuroML v1 file*

Description

Read one or more neurons from a NeuroML v1 file

Usage

```
read.neuron.neuroml(f, ..., AlwaysReturnNeuronList = FALSE)
```

Arguments

`f`	Path to a NeuroML format XML file
`...`	Additional arguments passed to read.morphml (and on to `xmlParse`)
`AlwaysReturnNeuronList`	See **Value** section (default `FALSE`)

Value

When the XML file contains only 1 cell *and* AlwaysReturnNeuronList=FALSE, a neuron object, otherwise a neuronlist containing one or more neurons.

References

https://docs.neuroml.org/Userdocs/Specification.html

See Also

read.morphml

---

read.neuron.swc *Read a neuron in swc file format*

Description

read.neuron.swc reads an SWC file on disk into a fully parsed neuron representation. However we normally recommend using read.neuron(s) since those functions cope with any file type.

read.ngraph.swc reads an SWC file on disk into the more generic (and forgiving) ngraph representation which provides a bridge to the igraph library.

Usage

```
read.neuron.swc(f, ...)

read.ngraph.swc(f, weights = FALSE, directed = TRUE, ...)
```

Arguments

f	path to file
...	Additional arguments. read.neuron.swc passes these to as.neuron and then on to neuron. read.neuron.swc passes them to ngraph.
weights	Logical value indicating whether edge weights defined by the 3D distance between points should be added to graph (default FALSE) *or* a numeric vector of weights.
directed	Whether the resultant graph should be directed (default TRUE)

Details

These functions will accept SWC neurons with multiple trees and arbitrary point index order. However only read.ngraph.swc will accept SWC files with cycles.

These functions would normally be called from read.neuron(s) rather than used directly. The only benefit of using read.neuron.swc is to avoid a very small overhead in identifying the SWC file type. Note that only read.neurons can read many files in one command to construct a neuronlist object.

SWC Format

According to http://www.neuronland.org/NLMorphologyConverter/MorphologyFormats/SWC/Spec.html SWC file format has a radius not a diameter specification

See Also

is.swc, read.neuron

`read.neuronlistfh` *Read a local, or remote, neuronlistfh object saved to a file.*

Description

Read a local, or remote, neuronlistfh object saved to a file.

Usage

```
read.neuronlistfh(file, localdir = NULL, update = FALSE, ...)
```

Arguments

`file`	The file path of the neuronlistfh object. Can be local, or remote (via http or ftp).
`localdir`	If the file is to be fetched from a remote location, this is the folder in which downloaded RDS file will be saved. The default value of `NULL` will save to a folder in the current R sessions temporary folder. See details.
`update`	Whether to update local copy of neuronlistfh (default: FALSE, see details)
`...`	Extra arguments to pass to `download.file`.

Details

When reading a remote `neuronlistfh` object, it is downloaded and cached to `localdir`. If there is already a cached file at the appropriate location and `update=TRUE` then the md5sums are checked and the downloaded file will be copied on top of the original copy if they are different; if `udpate=FALSE`, the default, then no action will be taken. After downloading a remote `neuronlistfh` object, a check is made for the existence of the `data` directory that will be used to individual objects. If this does not exist it will be created.

Note also that there is a *strict convention* for the layout of the files on disk. The neuronlistfh object will be saved in R's `RDS` format and will be placed next to a folder called `data` which will contain the data objects, also saved in RDS format. For example if `myneurons.rds` is downloaded to `localdir="\path\to\localdir"` the resultant file layout will be as follows:

- `\path\to\localdir\myneurons.rds`
- `\path\to\localdir\data\2f88e16c4f21bfcb290b2a8288c05bd0`
- `\path\to\localdir\data\5b58e040ee35f3bcc6023fb7836c842e`
- `\path\to\localdir\data\... etc`

Given this arrangement, the data directory should always be at a fixed location with respect to the saved neuronlistfh object and this is enforced on download and the default behaviour on read and write. However it does remain possible (if not recommended) to site the neuronlistfh and filehash database directory in different relative locations; if the neuronlistfh object specified by file does not have a filehash database with a valid `dir` slot and there is no 'data' directory adjacent to the neuronlistfh object, an error will result.

See Also

Other neuronlistfh: `[.neuronlistfh()`, `neuronlistfh()`, `remotesync()`, `write.neuronlistfh()`

`read.neurons` *Read one or more neurons from file to a neuronlist in memory*

Description

Read one or more neurons from file to a neuronlist in memory

Usage

```
read.neurons(
  paths,
  pattern = NULL,
  neuronnames = NULL,
  format = NULL,
  nl = NULL,
  df = NULL,
  OmitFailures = TRUE,
  SortOnUpdate = FALSE,
  ...
)
```

Arguments

`paths`	Paths to neuron input files *or* a directory containing neurons *or* a `neuronlistfh` object, *or* a zip archive containing multiple neurons.
`pattern`	If paths is a directory, `regex` that file names must match.
`neuronnames`	Character vector or function that specifies neuron names. See details.
`format`	File format for neuron (see `read.neuron`)
`nl`	An existing neuronlist to be updated (see details)
`df`	Optional data frame containing information about each neuron
`OmitFailures`	Omit failures (when TRUE) or leave an NA value in the list
`SortOnUpdate`	When `nl!=NULL` the resultant neuronlist will be sorted so that neurons are ordered according to the value of the `paths` argument.
`...`	Additional arguments to passed to `read.neuron` methods

Details

This function will cope with the same set of file formats offered by read.neuron.

If the paths argument specifies a (single) directory then all files in that directory will be read unless an optional regex pattern is also specified. Similarly, if paths specifies a zip archive, all neurons within the archive will be loaded.

neuronnames must specify a unique set of names that will be used as the names of the neurons in the resultant neuronlist. If neuronnames is a function then this will be applied to the path of each input file. The default value of basename=NULL results in each neuron being named for the input file from which it was read *after trimming the file extension*. This should match the NeuronName field of each individual neuron.

The optional dataframe (df) detailing each neuron should have rownames that match the names of each neuron. It would also make sense if the same key was present in a column of the data frame. If the dataframe contains more rows than neurons, the superfluous rows are dropped with a warning. If the dataframe is missing rows for some neurons an error is generated. If SortOnUpdate is TRUE then updating an existing neuronlist should result in a new neuronlist with ordering identical to reading all neurons from scratch.

Value

neuronlist object containing the neurons

See Also

read.neuron, write.neurons, fileformats

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlistz(), neuronlist(), nlapply(), write.neurons()

Examples

```
## Not run:
## Read C. elegans neurons from OpenWorm github repository
vds=paste0("VD", 1:13)
vdurls=paste0("https://raw.githubusercontent.com/openworm/CElegansNeuroML/",
  "103d500e066125688aa7ac5eac7e9b2bb4490561/CElegans/generatedNeuroML/",vds,
  ".morph.xml")
vdnl=read.neurons(vdurls, neuronnames=vds)
plot3d(vdnl)

## The same, but this time add some metadata to neuronlist
# fetch table of worm neurons from wormbase
library(rvest)
nlurl="http://wormatlas.org/neurons/Individual%20Neurons/Neuronframeset.html"
wormneurons = html_table(read_html(nlurl), fill=TRUE)[[4]]
vddf=subset(wormneurons, Neuron%in%vds)
rownames(vddf)=vddf$Neuron
# attach metadata to neuronlist
vdnl=read.neurons(vdurls, neuronnames=vds, df=vddf)
# use metadata to plot a subset of neurons
nclear3d()
```

```
plot3d(vdnl, grepl("P[1-6].app", Lineage))

## End(Not run)
```

`read.nrrd` *Read NRRD files/headers into memory*

Description

`read.nrrd` reads data into a raw array. If you wish to generate a `im3d` object that includes spatial calibration (but is limited to representing 3D data) then you should use `read.im3d`.

`nrrd.datafiles` returns the path to the separate data files listed in a detached NRRD header file.

Usage

```
read.nrrd(
  file,
  origin = NULL,
  ReadData = TRUE,
  AttachFullHeader = TRUE,
  Verbose = FALSE,
  ReadByteAsRaw = c("unsigned", "all", "none")
)

read.nrrd.header(file, Verbose = FALSE)

nrrd.datafiles(file, full.names = TRUE)
```

Arguments

`file`	Path to a nrrd (or a connection for `read.nrrd.header`)
`origin`	Add a user specified origin (x,y,z) to the returned object
`ReadData`	When FALSE just return attributes (i.e. the nrrd header)
`AttachFullHeader`	Include the full nrrd header as an attribute of the returned object (default TRUE)
`Verbose`	Status messages while reading
`ReadByteAsRaw`	Either a character vector or a logical vector specifying when R should read 8 bit data as an R `raw` vector rather than `integer` vector.
`full.names`	Whether to return the full paths to each data file (by analogy with `list.files`)

Details

`ReadByteAsRaw="unsigned"` (the default) only reads unsigned byte data as a raw array. This saves quite a bit of space and still allows data to be used for logical indexing.

Value

An array object, optionally with attributes from the nrrd header.

A list with elements for the key nrrd header fields

See Also

write.nrrd, read.im3d

Other nrrd: is.nrrd(), nrrd.voxdims(), write.nrrd()

---

read.vaa3draw *Read Vaa3d format image data*

Description

Read Vaa3d format image data

Usage

```
read.vaa3draw(f, ReadData = TRUE, Verbose = FALSE, ReadByteAsRaw = FALSE)
```

Arguments

f	Path to image to read
ReadData	Whether to read in data or just parse header
Verbose	Whether to print status messages
ReadByteAsRaw	Can reduce memory footprint by reading 8 bit data as a raw rather than 4 byte integers.

---

reglist *A simple wrapper class for multiple transformations*

Description

A reglist is read as a set of transformations to be applied sequentially starting with the first element, then applying the second transformation to the result of the first and so on. Each individual transformation is considered to map data from the sample (floating/moving) space to the reference (fixed/template) space.

Each transformation may have an attribute "swap" indicating that the natural direction of the transformation should be swapped (i.e. inverted). This can be done trivially in the case of affine transformations, expensively for others such as CMTK registrations (see cmtkreg) and not at all for others. Note that the term 'swap' is used to avoid a direct equivalence with inversion - many registration tools use the term *inverse* for directions that one might naively think of as as the natural direction of the transformation (see xformpoints.cmtkreg for discussion).

invert_reglist inverts a reglist object

c.reglist combines multiple reglists into a single reglist.

Usage

```
reglist(..., swap = NULL)

invert_reglist(x)

## S3 method for class 'reglist'
c(..., recursive = FALSE)
```

Arguments

...	One or more transformations/reglists to combine
swap	A vector of the same length as ... indicating whether the direction of each transformation should be swapped (i.e. mapping reference -> sample).
x	A reglist object to invert
recursive	Presently ignored

Details

The swap argument is provided as a convenience, but an attribute 'swap' can also be set directly on each registration.

Inversion

invert_reglist takes a minimal approach to inversion. It reverses the order of the individual elements of the registration and tags each of them with a swap attribute (or changes the value of the attribute if it already exists)

See Also

xform

c

Examples

```
I=diag(4)
S=I
diag(S)=c(1, 2, 3, 1)
rl=reglist(S, I)
rli=invert_reglist(rl)

## We can check the inversion by simplifying
m=simplify_reglist(rl)[[1]]
mi=simplify_reglist(rli)[[1]]
# NB solve will invert a homogeneous affine matrix
all.equal(m, solve(mi))
I=diag(4)
S=I
diag(S)=c(1, 2, 3, 1)
rl=reglist(S, I)
```

```
rl2=c(rl, 'path/to/my/reg.list')
rl3=c(reglist('path/to/my/reg.list'), rl)
```

`remotesync` *Synchronise a remote object*

Description

Synchronise a remote object

Usage

```
remotesync(
  x,
  remote = attr(x, "remote"),
  download.missing = TRUE,
  delete.extra = FALSE,
  ...
)

## S3 method for class 'neuronlistfh'
remotesync(
  x,
  remote = attr(x, "remote"),
  download.missing = FALSE,
  delete.extra = FALSE,
  indices = NULL,
  update.object = TRUE,
  ...
)
```

Arguments

`x`	Object to synchronise with a remote URL
`remote`	The remote URL to update from
`download.missing`	Whether to download missing objects (default TRUE)
`delete.extra`	Whether to delete objects (default TRUE)
`...`	Additional arguments passed to methods
`indices`	Character vector naming neurons to update (default `indices=NULL` implies all neurons).
`update.object`	Whether to update the `neuronlistfh` object itself on disk (default TRUE). Note that this assumes that the `neuronlistfh` object has not been renamed after it was downloaded.

Value

The updated neuronlistfh object (invisibly)

See Also

Other neuronlistfh: [.neuronlistfh(), neuronlistfh(), read.neuronlistfh(), write.neuronlistfh()

Examples

```
## Not run:
kcs20=read.neuronlistfh('http://flybrain.mrc-lmb.cam.ac.uk/si/nblast/flycircuit/kcs20.rds')
# update object from the web
kcs20=remotesync(kcs20)
# download all neurons with significant innervation of the vertical lobe
mbvl_neurons=subset(kcs20, (MB_VL_R+MB_VL_L)>200, rval='names')
kcs20=remotesync(kcs20, indices=mbvl_neurons, download.missing=TRUE)

## End(Not run)
```

---

reroot *Reroot neurons*

Description

Change the root node of a neuron (typically denoting the soma) to a new node specified by a node index, identifier or an XYZ position.

Usage

```
reroot(x, ...)

## S3 method for class 'neuron'
reroot(x, idx = NULL, pointno = NULL, point = NULL, ...)

## S3 method for class 'neuronlist'
reroot(x, idx = NULL, pointno = NULL, point = NULL, ...)
```

Arguments

x	A neuron or neuronlist object
...	Additional arguments passed to methods
idx	index of the node for the new root (between 1 and the number of nodes in the neuron).
pointno	new root node identifier (i.e. the PointNo column in the point array of the neuron, see details).
point	3-vector with X,Y,Z coordinates (data.frame or Nx3 matrix for neuronlist)

Details

All neurons in the natverse have a root point, which is used for during many operations on the branching structure of the neuron. This will often correspond to the soma of a neuron, but the soma is not always present and sometimes its position may be unknown. For example some connectomics datasets will have a certain position on a neuron marked as `to soma` when the soma is not present in the reconstruction but it is known to which branch it is attached.

The root point of a neuron is stored in the `StartPoint` field of the neuron (see Examples) and can also be accessed using the `rootpoints` function. For further details, please consult the Neurons as graph structures vignette. As an extension to the original nat specification, the point identifier (not point index) of the anatomical soma can be stored in the `tags$soma` field of the neuron

The node index refers is a number between 1 and N, the number of points in the neuron. It provides an index into the point array. The node id is an arbitrary identifier which may sometime be the same as the index, but may be e.g. a 64 bit integer that uniquely identifies nodes across all neurons in a database. Node ids can be retained after neurons are pruned even if the indices for each point change. For further details, again see the vignette mentioned above.

Value

neuron with a new root position (unless `idx`, `pointno`, and `point` are all `NULL`, when the original neuron is returned).

Examples

```
newCell07PN <- reroot(Cell07PNs[[2]], 5)
newCell07PN$StartPoint # 5
```

---

resample *Resample an object with a new spacing*

Description

Resample an object with a new spacing

resample a neuron with a new spacing

Usage

```
resample(x, ...)

## S3 method for class 'neuron'
resample(x, stepsize, ...)

## S3 method for class 'neuronlist'
resample(x, stepsize, ...)
```

Arguments

x	An object to resample
...	Additional arguments passed to methods
stepsize	The new spacing along the tracing

Details

resample.neuron Floating point columns including X,Y,Z,W will be interpolated using linear interpolation, while integer or factor columns will be interpolated using constant interpolation. See approx for details.

See Also

approx, seglengths

Other neuron: neuron(), ngraph(), plot.dotprops(), potential_synapses(), prune(), rootpoints(), spine(), subset.neuron()

---

rootpoints *Return the root, branch, or end points of a neuron or graph*

Description

rootpoints returns the root point(s) (one per tree, often the soma).

branchpoints returns the branch points.

endpoints returns the end points (aka leaf nodes); the root point will be returned if it also a leaf node.

Usage

```
rootpoints(x, ...)

## Default S3 method:
rootpoints(x, ...)

## S3 method for class 'neuron'
rootpoints(x, subtrees = 1, ...)

## S3 method for class 'igraph'
rootpoints(x, ...)

branchpoints(x, ...)

## Default S3 method:
branchpoints(x, ...)
```

```
## S3 method for class 'neuron'
branchpoints(x, subtrees = 1, ...)

## S3 method for class 'igraph'
branchpoints(x, ...)

endpoints(x, ...)

## S3 method for class 'neuron'
endpoints(x, subtrees = 1, ...)

## S3 method for class 'igraph'
endpoints(x, ...)

## Default S3 method:
endpoints(x, ...)
```

Arguments

x	Neuron or other object (e.g. igraph) which might have roots
...	Further arguments passed to methods (for ngraph or igraph objects eventually graph.nodes)).
subtrees	Integer index of the fully connected subtree in x$SubTrees. Only applicable when a neuron consists of multiple unconnected subtrees.

Details

A neuron may have multiple subtrees and therefore multiple roots. At present there is discrepancy between the *.neuron and *.igraph methods. For neurons we return the node indices, for igraph/ngraph objects the node identifiers (aka names/PointNo)

branchpoints.neuron returns a list if more than one subtree is specified

Value

FIXME Raw indices (in range 1:N) of vertices when x is a neuron, integer point identifier (aka PointNo) otherwise.

See Also

graph.nodes, ngraph

Other neuron: neuron(), ngraph(), plot.dotprops(), potential_synapses(), prune(), resample(), spine(), subset.neuron()

Examples

```
rootpoints(Cell07PNs[[1]])
endpoints(Cell07PNs[[1]])
```

`scale.neuron` *Scale and centre neuron 3D coordinates*

Description

note that `scale.dotprops` recalculates the tangent vectors after scaling the 3D coords. See `dotprops` for details.

Usage

```
## S3 method for class 'neuron'
scale(x, center = TRUE, scale = TRUE)

## S3 method for class 'dotprops'
scale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	A neuron
`center`	3-vector to subtract from x,y,z coords
`scale`	3-vector used to divide x,y,z coords

Details

If `scale=TRUE`, the neuron will be rescaled to unit sd in each axis. If `center=TRUE`, the neuron will be centred around the axis means. See `base::scale.default` for additional details.

Value

neuron with scaled coordinates

See Also

`scale.default`, `Ops.neuron`

Examples

```
n1.scaledown=scale(Cell07PNs[[1]],scale=c(2,2,3))
n1.scaleup=scale(Cell07PNs[[1]],scale=1/c(2,2,3))
```

seglengths *Calculate length of all segments in neuron*

Description

Calculate length of all segments in neuron

Usage

```
seglengths(x, all = FALSE, flatten = TRUE, sumsegment = TRUE)
```

Arguments

x	A neuron
all	Whether to calculate lengths for all segments when there are multiple subtrees (default: FALSE)
flatten	Whether to flatten the lists of lists into a single list when all=TRUE
sumsegment	Whether to return the length of each segment (when sumsegment=TRUE, the default) or a list of vectors of lengths of each individual edge in the segment.

Details

A segment is an unbranched portion of neurite consisting of at least one vertex joined by edges.Only segments in x$SegList will be calculated unless all=TRUE. Segments containing only one point will have 0 length.

Value

A vector of lengths for each segment or when sumsegment=FALSE a list of vectors

See Also

as.seglist.neuron

Examples

```
summary(seglengths(Cell07PNs[[1]]))
hist(unlist(seglengths(Cell07PNs[[1]], sumsegment = FALSE)),
  br=20, main='histogram of edge lengths', xlab='edge lengths /microns')
```

seglist *Make/convert neuron connectivity information into a seglist object*

Description

`seglist` makes a seglist object from a list of integer vectors of raw vertex ids. As a convenience if a vector of numeric ids are passed these are assumed to specify a neuron with 1 segment.

`as.seglist.neuron` will extract the seglist from a neuron, optionally extracting all subtrees (`all=TRUE`) and (in this case) flattening the list into a single hierarchy when `flatten=TRUE`. n.b. when `all=TRUE` but `flatten=FALSE` the result will *always* be a list of `seglist` objects (even if the neuron has only one subtree i.e. is fully connected).

`as.seglist.igraph` will convert a fully connected acyclic ngraph or igraph object into a seglist consisting of exactly one subtree.

Usage

```
seglist(...)

as.seglist(x, ...)

## S3 method for class 'neuron'
as.seglist(x, all = FALSE, flatten = FALSE, ...)

## S3 method for class 'igraph'
as.seglist(x, origin = NULL, Verbose = FALSE, ...)
```

Arguments

`...`	for `seglist` integer vectors to convert to a seglist
`x`	object passed to be converted to seglist
`all`	Whether to include segments from all subtrees
`flatten`	When `all=TRUE` flatten the lists of lists into a one-level list.
`origin`	The origin of the tree (see details)
`Verbose`	Whether to print progress updates to console (default FALSE)

Details

see `neuron` for further information about seglists.

If the graph vertices have `vid` attributes, typically defining the original vertex ids of a graph that was then decomposed into subgraphs, then the origin is assumed to refer to one of these vids not a raw vertex id of the current graph. The returned seglist will also contain these original vertex ids.

The head of the first segment in the seglist will be the origin.

Value

A `list` with additional class `seglist`.

a `list` with one entry for each unbranched segment.

See Also

`neuron`

`ngraph`,`igraph`

Examples

```
sl=seglist(c(1:2),c(2:6))
```

`seglist2swc` *Recalculate Neurons's SWCData using SegList and point information*

Description

Uses the SegList field (indices into point array) to recalculate point numbers and parent points for SWC data field (d).

Usage

```
seglist2swc(x, d, RecalculateParents = TRUE, ...)
```

Arguments

`x`	Neuron containing both the SegList and d fields or a plain seglist
`d`	SWC data block (only expected if x is a SegList)
`RecalculateParents`	Whether to recalculate parent points (default T)
`...`	Additional arguments passed to `normalise_swc`

Details

If any columns are missing then they are set to default values by `normalise_swc`. In particular

- PointNo integer 1:npoints
- Label = 0 (unknown)
- W NA_real

Note that each numeric entry in the incoming SegList is a raw index into the block of vertex data defined by `d`.

Value

A neuron if x was a neuron otherwise dataframe of swc data

See Also

as.neuron.data.frame, normalise_swc, neuron

---

segmentgraph *Return a simplified segment graph for a neuron*

Description

Return a simplified segment graph for a neuron

Usage

```
segmentgraph(
  x,
  weights = TRUE,
  segids = FALSE,
  exclude.isolated = FALSE,
  include.xyz = FALSE,
  reverse.edges = FALSE
)
```

Arguments

x	neuron
weights	Whether to include the original segment lengths as edge weights in the graph.
segids	Whether to include the integer segment ids as an edge attribute in the graph
exclude.isolated	Whether to eliminate isolated nodes
include.xyz	Whether to include 3D location as vertex attribute
reverse.edges	Whether to reverse the direction of each edge in the output graph to point towards (rather than away from) the root (default FALSE)

Details

The resultant graph will contain all branch and endpoints of the original neuron. This will be constructed from the SegList field, or where present, the SubTrees field (containing multiple SegLists for each isolated graph in the neuron). Each edge in the output graph will match one segment in the original SegList.

Value

igraph object containing only nodes of neuron keeping original labels (xdPointNo => V(g)$label) and vertex indices (1:nrow(x$d) => V(g)$vid).

Examples

```
sg=segmentgraph(Cell07PNs[[1]])
str(sg)
library(igraph)
plot(sg, edge.arrow.size=.4, vertex.size=10)
```

`select_points` *Interactively select 3D points in space*

Description

Plot a set of 3D points in space and select a subset of them interactively, using an rgl window

Usage

```
select_points(points, clear_plot_on_exit = FALSE)
```

Arguments

`points`	a matrix of 3D points to plot (or an object for which `xyzmatrix` can extract 3D points).
`clear_plot_on_exit`	Whether to remove points from the rgl scene when selection has been completed.

Value

A matrix describing selected 3D points

See Also

`prune_online`

Examples

```
## Not run:
# Select points from 3 olfactory projection neurons
selected_points = select_points(Cell07PNs[1:3])

## End(Not run)
```

`setdiff` *Find the (asymmetric) difference between two collections of objects*

Description

Find the (asymmetric) difference between two collections of objects

Usage

```
setdiff(x, y, ...)

## Default S3 method:
setdiff(x, y, ...)

## S3 method for class 'neuronlist'
setdiff(x, y, ...)
```

Arguments

`x`	the first collection to consider.
`y`	the second collection to consider.
`...`	additional arguments passed to methods

Details

Note that setdiff.default calls base::setdiff to ensure consistent behaviour for regular vectors.

As a convenience `setdiff.neuronlist` allows `y`, the second collection, to be a character vector of names.

Value

A collection of the same mode as `x` that contains all elements of `x` that are not present in `y`.

See Also

`setdiff`

`sholl_analysis` *Perform a Sholl analysis on neuron skeletons*

Description

Functions for Sholl analysis of neuronal skeletons

Usage

```
sholl_analysis(
  x,
  start = colMeans(xyzmatrix(x)),
  starting.radius = radius.step,
  ending.radius = 1000,
  radius.step = ending.radius/100
)

## S3 method for class 'neuron'
sholl_analysis(
  x,
  start = colMeans(xyzmatrix(x)),
  starting.radius = radius.step,
  ending.radius = 1000,
  radius.step = ending.radius/100
)

## S3 method for class 'neuronlist'
sholl_analysis(
  x,
  start = colMeans(xyzmatrix(x)),
  starting.radius = radius.step,
  ending.radius = 1000,
  radius.step = ending.radius/100
)
```

Arguments

`x`	a neuron or neuronlist object
`start`	the origin from which spheres are grown for the Sholl analysis
`starting.radius`	the radius of the first sphere. Defaults to the radius step
`ending.radius`	the radius of the last sphere. If NULL the distance to the furthest dendritic point from the start point is taken
`radius.step`	the change in radius between successive spheres. Defaults to one 100th of the radius of the ending sphere

Value

a data.frame of spheres radii and the number of dendritic intersections at each radius

Examples

```
## Not run:
# Calculate how much some neurons overlap with one another
## Example requires the package nat.flybrains
Cell07PNs_sholl = sholl_analysis(x = Cell07PNs, radius.step = 1, ending.radius = 100)
head(Cell07PNs_sholl[[1]])

## End(Not run)
```

`simplify_neuron` *Simplify a neuron to the longest tree with n branch points*

Description

Simplify a neuron to the longest tree with n branch points

Usage

```
simplify_neuron(x, n = 1, invert = FALSE, ...)
```

Arguments

`x`	A `neuron` to simplify
`n`	Required number of branch points (default=1, minimum 0)
`invert`	Whether to keep the simplified backbone (when `invert=FALSE`, the default) or its inverse.
`...`	Additional arguments (currently ignored)

Details

If the neuron already contains fewer than or exactly the requested number of branches, then the original neuron is returned. The approach is to build up the new neuron starting from the longest tree including no branches all the way up to the longest tree containing n branches. The distance calculations are only carried out once so it should be reasonably efficient. Nevertheless at each iteration, the longest path from the tree so far to the newly selected leaf is calculated and it is likely that this step could be avoided. Furthermore for large values of n, pruning excess branches rather than building would presumably be more efficient.

Value

The simplified `neuron` or the untouched original neuron for neurons that have <=n branch points.

Author(s)

Gregory Jefferis <jefferis@gmail.com>

See Also

spine

Examples

```
n=Cell07PNs[['ECA34L']]
n.simp=simplify_neuron(n)
n.simp4=simplify_neuron(n, n=4)

plot(n, col='green', WithNodes = FALSE)
plot(n.simp, col='red', add = TRUE)
plot(n.simp4, col='blue', add = TRUE)

# calculate the inverse as well
n.simp4.inv=simplify_neuron(n, n=4, invert=TRUE)
plot(n.simp4, col='blue')
plot(n.simp4.inv, col='red', add = TRUE)


# 3D plots
## Not run:
nclear3d()
plot3d(n.simp, col='red', add = TRUE)
plot3d(n.simp4, col='blue', add = TRUE)
plot3d(n, col='green', WithNodes = FALSE)

## End(Not run)

# or with plotly where transparency works
## Not run:
op <- options(nat.plotengine = 'plotly')
nclear3d()
plot3d(n.simp, col='red', alpha = 0.5, add = TRUE)
plot3d(n.simp4, col='blue', alpha = 0.5, add = TRUE)
plot3d(n, col='green', alpha = 0.5, WithNodes = FALSE)

## End(Not run)
```

---

simplify_reglist *Simplify a registration list*

Description

Simplify a registration list

Usage

```
simplify_reglist(reg, as.cmtk = NULL)
```

Arguments

reg	A registration list (reglist) containing one or more transformations.
as.cmtk	Whether to convert to a vector of CMTK format registrations (see cmtkreg). The default value of as.cmtk=NULL converts all registrations to CMTK if any one registration is in CMTK format (thus enabling them to be applied by CMTK tools in a single call). See details.

Details

This function

- inverts any affine matrices with attribute "swap"
- collapses multiple affine matrices into a single affine
- optionally converts all registrations to CMTK on disk registrations when possible.

Note that if any of the registrations are in CMTK format, the default behaviour is to try to convert all of the other registrations into CMTK format to enable them to be passed to CMTK in a single command. If as.cmtk=TRUE then there will be an error if this is not possible.

See Also

reglist, xform, cmtkreg

---

smooth_neuron *Smooth the 3D coordinates of a neuron skeleton*

Description

smooth_neuron smooths a neuron.

Usage

```
smooth_neuron(n, method = c("gauss", "spline"), ...)

smooth_segment_gauss(xyz, sigma, ...)
```

Arguments

n	Neuron to smooth
method	Smoothing method
...	Additional parameters passed to segment smoothing functions
xyz	A block of 3D coordinates defining an unbranched segment
sigma	The standard deviation of the Gaussian smoothing kernel (which has the same spatial units as the object being smoothed)

Value

A new neuron with smoothed 3d coordinates

Examples

```
ns=smooth_neuron(Cell07PNs[[1]], sigma=2)
# plot in 2D zooming in on axon terminals
plot(Cell07PNs[[1]], col='grey', xlim=c(260,290), ylim=c(115,90))
plot(ns, col='red', add=TRUE)

# 3D plot
plot3d(Cell07PNs[[1]], col='grey')
plot3d(ns, col='red')
```

---

spine *Compute the longest path (aka spine or backbone) of a neuron*

Description

Compute the longest path (aka spine or backbone) of a neuron

Usage

```
spine(
  n,
  UseStartPoint = FALSE,
  SpatialWeights = TRUE,
  invert = FALSE,
  rval = c("neuron", "length", "ids")
)
```

Arguments

n	the neuron to consider.
UseStartPoint	Whether to use the StartPoint of the neuron (often the soma) as the starting point of the returned spine.
SpatialWeights	logical indicating whether spatial distances (default) should be used to weight segments instead of weighting each edge equally.
invert	When invert=TRUE the spine is pruned away instead of being selected. This is only valid when rval='neuron' or rval='ids'.
rval	Character vector indicating the return type, one of 'neuron', 'length' or 'ids'. See **Value** section.

Details

Note that when `UseStartPoint=FALSE`, `spine` will find the path between all end points (including the root if it is an end point). Since the longest path must include an end point, this is equivalent to searching the whole graph for the longest path, but considerably faster.

Value

Either

- a neuron object corresponding to the longest path *or*
- the length of the longest path (when `rval="length"`) *or*
- an integer vector of raw point indices (when `rval="ids"`).

See Also

`diameter`, `shortest.paths`, `prune_strahler` for removing lower order branches from a neuron, `prune` for removing parts of a neuron by spatial criteria.

Other neuron: `neuron()`, `ngraph()`, `plot.dotprops()`, `potential_synapses()`, `prune()`, `resample()`, `rootpoints()`, `subset.neuron()`

Examples

```
pn.spine=spine(Cell07PNs[[1]])

plot3d(Cell07PNs[[1]])
plot3d(pn.spine, lwd=4, col='black')

# just extract length
spine(Cell07PNs[[1]], rval='length')
# same result since StartPoint is included in longest path
spine(Cell07PNs[[1]], rval='length', UseStartPoint=TRUE)

# extract everything but the spine
antispine=spine(Cell07PNs[[1]], invert=TRUE)

plot3d(Cell07PNs[[1]])
plot3d(antispine, lwd=4, col='red')
```

---

stitch_neuron *Stitch two neurons together at their closest endpoint*

Description

Stitch two neurons together at their closest endpoint

Usage

```
stitch_neuron(a, b)
```

Arguments

a, b	Neurons to join together

Details

This function joins two neurons at their nearest point (only one). Let's say you have two neurons a and b. a and b will be joined at one point that are closest to each other. However, when say there are multiple points at a and b which are closer and could be joined, then do not use this function, use the function stitch_neurons_mst, which is slower but will merge at multiple points. Note that for CATMAID neurons the neuron with the soma tag will be treated as the first (master neuron). Furthermore in this case the PointNo (aka node id) should already be unique. Otherwise it will be adjusted to ensure this.

Author(s)

Gregory Jefferis <jefferis@gmail.com>

See Also

stitch_neurons

Examples

```
dl1_main=simplify_neuron(dl1neuron, n = 1, invert = FALSE)
dl1_branches=simplify_neuron(dl1neuron, n = 1, invert = TRUE)
dl1_whole = stitch_neuron(dl1_main,dl1_branches)
```

---

stitch_neurons *Stitch multiple fragments into single neuron using nearest endpoints*

Description

Stitch multiple fragments into single neuron using nearest endpoints

Usage

```
stitch_neurons(x, prefer_soma = TRUE, sort = TRUE, warndist = 1000)
```

Arguments

x	A neuronlist containing fragments of a single neuron
prefer_soma	When TRUE (the default) the fragment tagged as the soma will be used as the master neuron.
sort	When TRUE (the default) the fragments will be sorted by the number of nodes they contain before stitching.
warndist	If distance is greater than this value, create a warning.

Details

Neurons will be ordered by default such the largest (by node count) neuron with a soma tag is the master neuron - i.e. the one containing the root node. Fragments are joined recursively in this sort order each time simply picking the closest fragment to the current *master*. Closest is here defined by the distance between nearest endpoints.

Value

A single neuron object containing all input fragments.

Author(s)

Gregory Jefferis <jefferis@gmail.com>

See Also

stitch_neuron

Examples

```
## Not run:
dl1_main=simplify_neuron(dl1neuron, n = 1, invert = FALSE)
dl1_branches=simplify_neuron(dl1neuron, n = 1, invert = TRUE)
dl1_branches1=simplify_neuron(dl1_branches, n = 1, invert = FALSE)
dl1_branches2=simplify_neuron(dl1_branches, n = 1, invert = TRUE)
dl1_fragment <- list(dl1_main,dl1_branches1,dl1_branches2)
dl1_fragment <- as.neuronlist(dl1_fragment)
dl1_whole = stitch_neurons(dl1_fragment)

## End(Not run)
```

`stitch_neurons_mst` *Stitch multiple fragments into single neuron using minimum spanning tree*

Description

Stitch multiple fragments into single neuron using minimum spanning tree

Usage

```
stitch_neurons_mst(x, threshold = Inf, k = 10L)
```

Arguments

`x`	Fragments that could be a `neuronlist` containing multiple neurons or a single neuron with multiple unconnected subtrees.
`threshold`	The threshold distance above which new vertices will not be connected (default=`Inf` disables this feature). This parameter prevents the merging of vertices that are so far away from the main neuron that they are likely to be spurious.
`k`	The number of nearest neighbours to consider when trying to merge different clusters.

Details

The neurons are joined using the minimum spanning tree i.e. the tree that minimises the sum of edge weights (here, the Euclidean distance). The neuron is rooted in the largest cluster; if this cluster contained the original root of the neuron, then this should be retained.

Note that when a threshold length is used, it must be specified in the same units (microns, nm etc) as the neuron being stitched.

If you wish to process a neuronlist containing multiple neurons each of which must have all their subtrees stitched then you must use `nlapply` to process the list iteratively.

Value

A single `neuron` object containing all input fragments.

Author(s)

Sridhar Jagannathan <j.sridharrajan@gmail.com>

See Also

`simplify_neuron`, `spine`, `ngraph`, `igraph::mst`

Examples

```
n=Cell07PNs[['ECA34L']]
# find the tree with the 10 most important branches
n_main=simplify_neuron(n, n = 10)
# and the complement
n_branches=simplify_neuron(n, n = 10, invert = TRUE)

# plot the inputs
plot(n_main, col='red', WithNodes=FALSE)
plot(n_branches, col='blue', add=TRUE, WithNodes=FALSE)

# join the two fragments back together again
# first make a neuronlist containing the two fragments
nl=neuronlist(n_main, n_branches)
# and stitch those
n_stitched=stitch_neurons_mst(nl)

## Not run:
# look at the neurons in 3D - they appear identical in this case
plot3d(n, alpha=.5, col='cyan', WithNodes=FALSE)
plot3d(n_stitched, alpha=.5, col='red', WithNodes=FALSE)

## End(Not run)

## second example neuron containing multiple sub trees
n=Cell07PNs[['ECA34L']]
# remove a single vertex, breaking the neuron in two
# note that the root (purple node) has moved
np=prune_vertices(n, 105)
plot(np)

# now stitch the broken neuron back together again
nph=stitch_neurons_mst(np)
# note that the root remains the same as the input neuron (np)
plot(nph)
```

---

strahler_order *Find the Strahler order of each point in a neuron*

Description

The Strahler order will be 1 for each tip segment and then 1 + the maximum of the Strahler order of each parent segment for internal segments. Branch points will have the Strahler order of the closest segment to the root of which they are part.

Usage

```
strahler_order(x)
```

Arguments

x	A neuron

Details

It is vital that the root of the neuron is valid since this determines the flow direction for calculation of the Strahler order. At present the function is not defined for neurons with multiple subtrees.

Internally, this function uses segmentgraph to find a reduced segmentgraph for the neuron.

Value

A list containing

- points Vector of integer Strahler orders for each point in the neuron
- segments Vector of integer Strahler orders for each segment in the neuron

References

https://en.wikipedia.org/wiki/Strahler_number

See Also

prune_strahler, a segmentgraph (a form of ngraph) representation is used to calculate the Strahler order.

---

sub2ind *Find 1D index given n-dimensional indices*

Description

Emulates the MATLAB function sub2ind.

Usage

```
sub2ind(dims, indices)
```

Arguments

dims	vector of dimensions of object to index into.
indices	vector of n-dimensional indices.

subset *Subset methods for different nat objects*

Description

These methods enable subsets of some nat objects including neurons and neuronlists to be obtained. See the help for each individual method for details.

See Also

subset.neuron, subset.dotprops, subset.hxsurf, subset.neuronlist

subset.dotprops *Subset points in dotprops object that match given conditions*

Description

Subset points in dotprops object that match given conditions

Usage

```
## S3 method for class 'dotprops'
subset(x, subset, invert = FALSE, ...)
```

Arguments

x	A dotprops object
subset	A subset of points defined by indices, an expression or a function (see Details)
invert	Whether to invert the subset criteria - a convenience when selecting by function or indices.
...	Additional parameters (currently ignored)

Details

subset defines either logical or numeric indices, in which case these are simply applied to the matrices that define the points, vect fields of the dotprops object etc OR a function (which is called with the 3D points array and returns T/F. OR an expression vector).

Value

subsetted dotprops object

See Also

prune.dotprops, subset.neuron

Examples

```
## subset using indices ...
dp=kcs20[[10]]
dp1=subset(dp, 1:50)

# ... or an expression
dp2=subset(dp, alpha>0.7)
front=subset(dp, points[,'Z']<40)
# use a helper function
between=function(x, lower, upper) x>=lower & x<=upper
middle=middle=subset(dp, between(points[,'Z'], 40, 60))

# plot results in 3D

plot3d(front, col='red')
plot3d(middle, col='green')
plot3d(dp, col='blue')


## Not run:

## subset using an selection function
s3d=select3d()
dp1=subset(dp, s3d(points))
# special case of previous version
dp2=subset(dp, s3d)
# keep the points that were removed from dp2
dp2.not=subset(dp, s3d, invert=TRUE)
# (another way of doing the same thing)
dp2.not=subset(dp, Negate(s3d))
stopifnot(all.equal(dp1, dp2))
dp2=subset(dp, alpha>0.5 & s3d(pointd))
dp3=subset(dp, 1:10)

## subset each dotprops object in a whole neuronlist
plot3d(kcs20)
s3d=select3d()
kcs20.partial = nlapply(kcs20, subset, s3d)
clear3d()
plot3d(kcs20.partial, col='red')
plot3d(kcs20, col='grey')

## End(Not run)

## Not run:
## subset dotprops by mesh
#' library(nat.flybrains)
# extract calyx surface and convert to mesh3d
calyx=as.mesh3d(subset(MBL.surf, "MB_CA_L"))
# subset one neuron with this surface
kcs20.2_ca=subset(kcs20[[2]], function(x) pointsinside(x, calyx))
shade3d(calyx, alpha=0.2)
```

```
plot3d(kcs20.2_ca, lwd=3, col='black')

## subset neuronlist of dotprops by mesh
peduncle=as.mesh3d(subset(MBL.surf, "MB_PED_L"))
kcs20.ped=nlapply(kcs20, function(x) subset(x, pointsinside(x, peduncle)))
shade3d(peduncle, alpha=.2)
plot3d(kcs20.ped)

## End(Not run)
```

`subset.hxsurf` *Subset hxsurf object to specified regions*

Description

Subset hxsurf object to specified regions

Usage

```
## S3 method for class 'hxsurf'
subset(x, subset = NULL, drop = TRUE, rval = c("hxsurf", "names"), ...)
```

Arguments

`x`	A dotprops object
`subset`	Character vector specifying regions to keep. Interpreted as `regex` if of length 1 and no fixed match.
`drop`	Whether to drop unused vertices after subsetting (default: `TRUE`)
`rval`	Whether to return a new `hxsurf` object or just the names of the matching regions
`...`	Additional parameters (currently ignored)

Value

subsetted hxsurf object

See Also

Other hxsurf: `as.hxsurf()`, `as.mesh3d()`, `materials()`, `plot3d.hxsurf()`, `read.hxsurf()`, `write.hxsurf()`

Examples

```
# plot only vertical lobe
vertical_lobe=subset(MBL.surf, "VL")

plot3d(vertical_lobe, alpha=0.3)
plot3d(kcs20)
```

```
# there is also a shortcut for this
nclear3d()
plot3d(MBL.surf, subset = "VL", alpha=0.3)
```

`subset.neuron` *Subset neuron by keeping only vertices that match given conditions*

Description

Subset neuron by keeping only vertices that match given conditions

Usage

```
## S3 method for class 'neuron'
subset(x, subset, invert = FALSE, ...)
```

Arguments

`x`	A neuron object
`subset`	A subset of points defined by indices, an expression, or a function (see Details)
`invert`	Whether to invert the subset criteria - a convenience when selecting by function or indices.
`...`	Additional parameters (passed on to `prune_vertices`)

Details

`subset` defines which vertices of the neuron to keep and is one of

- logical or numeric indices, in which case these are simply used to index the vertices in the order of the data.frame `x$d`. Note that any NA values are ignored.
- a function (which is called with the 3D points array and returns T/F vector)
- an expression evaluated in the context of the `x$d` data.frame containing the SWC specification of the points and connectivity of the neuron. This can therefore refer e.g. to the X,Y,Z location of vertices in the neuron.

Note that due to its use of non-standard evaluation `subset.neuron`, which is convenient interactive use but can be fragile when used inside other functions. If you run into trouble it is recommended to use the underlying `prune_vertices` function.

Value

subsetted neuron

See Also

prune.neuron, prune_vertices, subset.dotprops

Other neuron: neuron(), ngraph(), plot.dotprops(), potential_synapses(), prune(), resample(), rootpoints(), spine()

Examples

```
n=Cell07PNs[[1]]
# keep vertices if their X location is > 2000
n1=subset(n, X>200)
# diameter of neurite >1
n2=subset(n, W>1)
# first 50 nodes
n3=subset(n, 1:50)
# everything but first 50 nodes
n4=subset(n, 1:50, invert=TRUE)

## subset neuron by graph structure
# first plot neuron and show the point that we will use to divide the neuron
n=Cell07PNs[[1]]
plot(n)
# this neuron has a tag defining a point at which the neuron enters a brain
# region (AxonLHEP = Axon Lateral Horn Entry Point)
points(t(xyzmatrix(n)[n$AxonLHEP, 1:2]), pch=19, cex=2.5)

# now find the points downstream (distal) of that with respect to the root
ng=as.ngraph(n)
# use a depth first search
distal_points=igraph::graph.dfs(ng, root=n$AxonLHEP, unreachable=FALSE,
  mode='out')$order
distal_tree=subset(n, distal_points)
plot(distal_tree, add=TRUE, col='red', lwd=2)

# Find proximal tree as well
# nb this does not include the AxonLHEP itself as defined here
proximal_points=setdiff(igraph::V(ng), distal_points)
proximal_tree=subset(n, proximal_points)
plot(proximal_tree, add=TRUE, col='blue', lwd=2)

## Not run:
## subset using interactively defined spatial regions
plot3d(n)
# nb you can save this select3d object using save or saveRDS functions
# for future non-interactive use
s3d=select3d()
n4=subset(n, s3d(xyzmatrix(n)))
# special case of previous version
n5=subset(n, s3d)
stopifnot(all.equal(n4,n5))
# keep the points that were removed from n1
n4.not=subset(n,Negate(s3d))
# vertices with x position > 100 and inside the selector function
```

```
n6=subset(n,X>100 & s3d(X,Y,Z))

## subset each neuron object in a whole neuronlist
n10=Cell07PNs[1:10]
plot3d(n10, lwd=0.5, col='grey')
n10.crop = nlapply(n10, subset, X>250)
plot3d(n10.crop, col='red')

## subset a neuron using a surface
library(nat.flybrains)
# extract left lateral horn surface and convert to mesh3d
lh=as.mesh3d(subset(IS2NP.surf, "LH_L"))
# subset neuron with this surface
x=subset(Cell07PNs[[1]], function(x) pointsinside(x, lh))
shade3d(lh, alpha=0.3)
plot3d(x, lwd=3, col='blue')
# Now find the parts of the neuron outside the surface
y=subset(Cell07PNs[[1]], function(x) Negate(pointsinside)(x, lh))
plot3d(y, col='red', lwd=2)

## End(Not run)
```

`subset.neuronlist` *Subset neuronlist returning either new neuronlist or names of chosen neurons*

Description

Subset neuronlist returning either new neuronlist or names of chosen neurons

Usage

```
## S3 method for class 'neuronlist'
subset(
  x,
  subset,
  filterfun,
  rval = c("neuronlist", "names", "data.frame"),
  ...
)
```

Arguments

`x`	a neuronlist
`subset`	An expression that can be evaluated in the context of the dataframe attached to the neuronlist. See details.
`filterfun`	a function which can be applied to each neuron returning `TRUE` when that neuron should be included in the return list.
`rval`	What to return (character vector, default='neuronlist')
`...`	additional arguments passed to `filterfun`

Details

The subset expression should evaluate to one of

- character vector of names
- logical vector
- vector of numeric indices

Any missing names are dropped with a warning. The `filterfun` expression is wrapped in a try. Neurons returning an error will be dropped with a warning.

You may also be interested in `find.neuron`, which enables objects in a neuronlist to be subsetted by a 3D selection box. In addition `subset.neuron`, `subset.dotprops` methods exist: these are used to remove points from neurons (rather than to remove neurons from neuronlists).

Value

A `neuronlist`, character vector of names or the attached data.frame according to the value of `rval`

See Also

`neuronlist`, `find.neuron`, `subset.data.frame`, `subset.neuron`, `subset.dotprops`

Examples

```
da1pns=subset(Cell07PNs,Glomerulus=='DA1')
with(da1pns,stopifnot(all(Glomerulus=='DA1')))
gammas=subset(kcs20,type=='gamma')
with(gammas,stopifnot(all(type=='gamma')))
# define a function that checks whether a neuron has points in a region in
# space, specifically the tip of the mushroom body alpha' lobe
aptip<-function(x) {xyz=xyzmatrix(x);any(xyz[,'X']>350 & xyz[,'Y']<40)}
# this should identify the alpha'/beta' kenyon cells only
apbps=subset(kcs20,filterfun=aptip)
# look at which neurons are present in the subsetted neuronlist
head(apbps)
# combine global variables with dataframe columns
odds=rep(c(TRUE,FALSE),10)
stopifnot(all.equal(subset(kcs20,type=='gamma' & odds),
            subset(kcs20,type=='gamma' & rep(c(TRUE,FALSE),10))))
## Not run:
# make a 3D selection function using interactive rgl::select3d() function
s3d=select3d()
# Apply a 3D search function to the first 100 neurons in the neuronlist dataset
subset(dps[1:100],filterfun=function(x) {sum(s3d(xyzmatrix(x)))>0},
  rval='names')
# combine a search by metadata, neuropil location and 3D location
subset(dps, Gender=="M" & rAL>1000, function(x) sum(s3d(x))>0, rval='name')
# The same but specifying indices directly, which can be considerably faster
# when neuronlist is huge and memory is in short supply
subset(dps, names(dps)[1:100],filterfun=function(x) {sum(s3d(xyzmatrix(x)))>0},
  rval='names')
```

```
## End(Not run)
```

---

summary.neuronlist *Summary statistics for neurons (e.g. cable length, number of nodes)*

Description

summary.neuronlist computes tree statistics for all the neurons in a neuronlist object

summary.neuron computes statistics for individual neurons

summary.mesh3d computes statistics including face numbers and surface area for meshes. See vcgArea for details of area calculation.

summary.dotprops computes statistics for individual neurons in dotprops format. Note the veclength argument.

Usage

```
## S3 method for class 'neuronlist'
summary(object, ..., include.attached.dataframe = FALSE)

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

## S3 method for class 'mesh3d'
summary(object, ...)

## S3 method for class 'dotprops'
summary(object, veclength = 1, ...)
```

Arguments

object	The neuron or neuronlist to summarise
...	For summary.neuronlist additional arguments passed on to summary methods for individual neurons
include.attached.dataframe	Whether to include the neuronlists attached metadata in the returned data.frame.
veclength	The vector length to assume for each segment so that a cable length estimate can be made.

Value

A data.frame summarising the tree properties of the neuron with columns

- root
- nodes
- segments

- branchpoints
- endpoints
- cable.length
- nTrees

See Also

seglengths, vcgArea

Examples

```
# summary for a whole neuronlist
summary(Cell07PNs)
# including the attached data.frame with additional metadata
head(summary(Cell07PNs, include.attached.dataframe = FALSE))
# for a single regular format neuron
summary(Cell07PNs[[1]])
# for a single dotprops format neuron
summary(kcs20[[1]])
# specify a different estimate for the cable length associated with a single
# point in the neuron
summary(kcs20[[1]], veclength=1.2)
```

---

threshold *Threshold an object, typically to produce a mask*

Description

Threshold an object, typically to produce a mask

Usage

```
threshold(x, ...)

## S3 method for class 'im3d'
threshold(
  x,
  threshold = 0,
  mode = c("logical", "integer", "raw", "numeric"),
  ...
)
```

Arguments

x	Object to be thresholded
...	Additional arguments passed to methods
threshold	Either a numeric value that pixels must **exceed** in order to be included in the mask *or* a logical vector defining foreground pixels.
mode	The storage mode of the resultant object (see vector

Details

Note that `threshold.im3d` passes ... arguments on to `im3d`

Value

an object with attributes matching `x` and elements with value `as.vector(TRUE, mode=mode)` i.e. `TRUE, 1, 0x01` and `as.vector(FALSE, mode=mode)` i.e. `FALSE, 0, 0x00` as appropriate.

See Also

Other im3d: `as.im3d()`, `boundingbox()`, `im3d-coords`, `im3d-io`, `im3d()`, `imexpand.grid()`, `imslice()`, `is.im3d()`, `mask()`, `origin()`, `projection()`, `unmask()`, `voxdims()`

Examples

```
x=im3d(rnorm(1000),dims=c(10,10,10), BoundingBox=c(20,200,100,200,200,300))
stopifnot(all.equal(threshold(x, 0), threshold(x, x>0)))
```

---

tpsreg *Thin plate spline registrations via xform and friends*

Description

`tpsreg` creates an object encapsulating a thin plate spine transform mapping a paired landmark set.

`xformpoints.tpsreg` enables `xform` and friends to transform 3d vertices (or more complex objects containing 3d vertices) using a thin plate spline mapping stored in a `tpsreg` object.

Usage

```
tpsreg(sample, reference, ...)

## S3 method for class 'tpsreg'
xformpoints(reg, points, swap = NULL, ...)
```

Arguments

`sample, reference`	Matrices defining the sample (or floating) and reference (desired target after transformation) spaces. See details.
`...`	additional arguments passed to `xformpoints.tpsreg`
`reg`	The `tpsreg` registration object
`points`	The 3D points to transform
`swap`	Whether to change the direction of registration (default of `NULL` checks if reg has a `attr('swap'=TRUE)`) otherwise

Details

Note that we use the **nat** convention for naming the sample/reference space arguments but these actually clash with the nomenclature in the underlying `Morpho::tps3d` function.

- refmat (Morpho3d) == sample (nat)
- tarmat (Morpho3d) == reference (nat)

See Also

`reglist`, `read.landmarks`

Examples

```
## Not run:
## A full worked example of using landmarks based registration to construct
## a mirroring registration from one side of the brain to the other.

# read in set of landmarks defined in FAFB CATMAID
library('catmaid')
emlandmarks=catmaid::read.neurons.catmaid('annotation:^GJLandmark')

# Match up L and R pairs
library('stringr')
emlandmarks[,'side']=stringr::str_match(emlandmarks[,'name'], "([LR]) Landmark")[,2]
emlandmarks[,'shortname']=stringr::str_match(emlandmarks[,'name'], "(.*)([LR]) Landmark.*")[,2]
emlandmarks[,'shortname']=sub("[_ ]+$", "", emlandmarks[,'shortname'])

library('dplyr')
lmpairs=dplyr::inner_join(
  dplyr::filter(emlandmarks[,], side=="L"),
  dplyr::filter(emlandmarks[,], side=="R"),
  by='shortname', suffix=c(".L",".R"))

# find mean xyz position of each landmark (they are drawn as a little cross)
lmxyz=t(sapply(emlandmarks, function(x) colMeans(xyzmatrix(x))))
# construct thin plate splines registration (here mapping the right side neurons to left side)
mirror_reg=tpsreg(
  lmxyz[as.character(lmpairs$skid.R),],
  lmxyz[as.character(lmpairs$skid.L),]
)
# map RHS DA2 PNs onto left and compare with LHS neurons
da2pns.R=catmaid::read.neurons.catmaid('glomerulus DA2 right')
da2pns.L=catmaid::read.neurons.catmaid('glomerulus DA2 left')

da2pns.R.L=xform(da2pns.R, reg = mirror_reg)
plot(da2pns.L, col='red')
plot(da2pns.R.L, col='blue', add=TRUE)

## End(Not run)
```

union *Find the union of two collections of objects*

Description

Find the union of two collections of objects

Usage

```
union(x, y, ...)

## Default S3 method:
union(x, y, ...)

## S3 method for class 'neuronlist'
union(x, y, ...)
```

Arguments

x	the first collection to consider.
y	the second collection to consider.
...	additional arguments passed to methods

Details

Note that union.default calls base::union to ensure consistent behaviour for regular vectors.

Value

A collection of the same mode as x that contains all unique elements of x and y.

See Also

union

unmask *Make im3d image array containing values at locations defined by a mask*

Description

Make im3d image array containing values at locations defined by a mask

Usage

```
unmask(
  x,
  mask,
  default = NA,
  attributes. = attributes(mask),
  copyAttributes = TRUE
)
```

Arguments

x	the data to place on a regular grid
mask	An im3d regular image array where non-zero voxels are the selected element.
default	Value for regions outside the mask (default: NA)
attributes.	Attributes to set on new object. Defaults to attributes of mask
copyAttributes	Whether to copy over attributes (including dim) from the mask to the returned object. default: TRUE

Details

The values in x will be placed into a grid defined by the dimensions of the mask in the order defined by the standard R linear subscripting of arrays (see e.g. arrayInd).

Value

A new im3d object with attributes/dimensions defined by mask and values from x. If copyAttributes is FALSE, then it will have mode of x and length of mask but no other attributes.

See Also

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), voxdims()

Examples

```
## Not run:
# read in a mask
LHMask=read.im3d(system.file('tests/testthat/testdata/nrrd/LHMask.nrrd', package='nat'))
# pick out all the non zero values
inmask=LHMask[LHMask!=0]
# fill the non-zero elements of the mask with a vector that iterates over the
# values 0:9
stripes=unmask(seq(inmask)%%10, LHMask)
# make an image from one slice of that result array
image(imslice(stripes,11), asp=TRUE)

## End(Not run)
```

voxdims *Return voxel dimensions of an object*

Description

This would properly be thought of as the voxel spacing when voxels are assumed not to have a physical extent (only a location).

Usage

```
voxdims(x, ...)

## S3 method for class 'im3d'
voxdims(x, ...)

## S3 method for class 'character'
voxdims(x, ...)

## Default S3 method:
voxdims(x, dims, ...)
```

Arguments

x	An im3d object with associated voxel dimensions, a path to or a 2 x 3 BoundingBox matrix.
...	Additional arguments for methods
dims	The number of voxels in each dimension when x is a BoundingBox matrix.

Details

We follow Amira's convention of returning a voxel dimension equal to the bounding box size (rather than 0) for any dimension with only 1 voxel.

Value

A numeric vector of length 3, NA when missing.

See Also

boundingbox

Other im3d: as.im3d(), boundingbox(), im3d-coords, im3d-io, im3d(), imexpand.grid(), imslice(), is.im3d(), mask(), origin(), projection(), threshold(), unmask()

wire3d *Wire frame plots*

Description

This function directs the wireframe plot based on the plotengine backend selected.

Usage

```
wire3d(
  x,
  ...,
  add = TRUE,
  gridlines = FALSE,
  plotengine = getOption("nat.plotengine")
)

## S3 method for class 'hxsurf'
wire3d(x, Regions = NULL, ...)

## S3 method for class 'mesh3d'
wire3d(x, ..., front = "lines", back = "lines")

## S3 method for class 'shapelist3d'
wire3d(x, override = TRUE, ...)
```

Arguments

x	object of type 'mesh3d' (triangular mesh or quad mesh), 'hxsurf' or 'shapelist3d'
...	Additional arguments passed to wire3d or
add	whether to add objects to an existing plot
gridlines	Whether to display gridlines when using plotly as the backend plotting engine (default: FALSE) add_trace depending on the @param plotengine option choosen
plotengine	Whether to use plotting backend of 'rgl' or 'plotly'
Regions	When x is a multi region hxsurf object. Default plots all. Seed as.mesh3d for details of how the argument is handled.
front, back	Material properties for rendering.
override	should the parameters specified here override those stored in the object?

See Also

wire3d

Examples

```
library(alphashape3d)
kcs20.a=ashape3d(xyzmatrix(kcs20), alpha = 10)
plot(kcs20.a)

# convert to mesh3d
kcs20.mesh=as.mesh3d(kcs20.a, meshColor = "edges")

# For plotly..
options(nat.plotengine = 'plotly')
wire3d(kcs20.mesh,alpha = 0.1, add = FALSE, col = 'blue')

# For rgl..
options(nat.plotengine = 'rgl')
wire3d(kcs20.mesh,alpha = 0.1, add = FALSE, col = 'blue')
```

---

write.amiramesh *Write a 3D data object to an AmiraMesh format file*

Description

Write a 3D data object to an AmiraMesh format file

Usage

```
write.amiramesh(
  x,
  file,
  enc = c("binary", "raw", "text", "hxzip"),
  dtype = c("float", "byte", "short", "ushort", "int", "double"),
  endian = .Platform$endian,
  WriteNrrdHeader = FALSE
)
```

Arguments

x	The image data to write (an im3d, or capable of being interpreted as such)
file	Character vector describing a single file
enc	Encoding of the data. NB "raw" and "binary" are synonyms.
dtype	Data type to write to disk
endian	Endianness of data block. Defaults to current value of .Platform$endian.
WriteNrrdHeader	Whether to write a separate detached nrrd header next to the AmiraMesh file allowing it to be opened by a NRRD reader. See details.

Details

Note that only 'raw' or 'text' format data can accommodate a detached NRRD format header since Amira's HxZip format is subtly different from NRRD's gzip encoding. There is a full description of the detached NRRD format in the help for write.nrrd.

See Also

.Platform, read.amiramesh, write.nrrd

Examples

```
d=array(rnorm(1000), c(10, 10, 10))
tf=tempfile(fileext='.am')
write.amiramesh(im3d(d, voxdims=c(0.5,0.5,1)), file=tf, WriteNrrdHeader=TRUE)
d2=read.nrrd(paste(tf, sep='', '.nhdr'))
all.equal(d, d2, tol=1e-6)
```

---

write.cmtk *Write a suitable list to a CMTK TypedStream file on disk*

Description

This is probably only of interest to developers. End users will probably wish to use more specific functions such as write.cmtkreg for writing out registrations.

Usage

```
write.cmtk(l, con, gzip = FALSE, version = NA_character_)
```

Arguments

l	Appropriately formatted list
con	A character string specifying a path or a connection
gzip	Whether to gzip output file (default FALSE)
version	TYPEDSTREAM version number, defaults to "1.1" if not specified in the version attribute of l.

Details

NB a version specified on the command line overrides one encoded as an attribute in the input list.

See Also

Other cmtk-io: cmtk.extract_affine(), read.cmtkreg(), read.cmtk(), write.cmtkreg()

write.cmtkreg *Write out CMTK registration list to folder*

Description

Write out CMTK registration list to folder

Usage

```
write.cmtkreg(reglist, foldername, version = "2.4")
```

Arguments

reglist	List specifying CMTK registration parameters
foldername	Path to registration folder (usually ending in .list)
version	CMTK version for registration (default 2.4). Will be converted to character vector if not already.

Details

Note that transformation in the forward direction (i.e. sample->ref) e.g. as calculated from a set of landmarks where set 1 is the sample is considered an inverse transformation by the IGS software. So in order to use such a transformation as an initial affine with the registration command the switch –initial-inverse must be used specifying the folder name created by this function.

CMTK v2.4 fixed a long-standing bug in affine (de)composition to CMTK params. This resulted in a non-backwards compatible change marked by writing the TYPEDSTREAM as version 2.4. The R code in this package implements both the new and old compose/decompose functions, using the new by default.

See Also

Other cmtk-io: `cmtk.extract_affine()`, `read.cmtkreg()`, `read.cmtk()`, `write.cmtk()`

write.hxsurf *Write Amira surface (aka HxSurface or HyperSurface) into .surf file.*

Description

Write Amira surface (aka HxSurface or HyperSurface) into .surf file.

Usage

```
write.hxsurf(surf, filename)
```

Arguments

surf	hxsurf object to write to file.
filename	character vector defining path to file.

Value

NULL or integer status from close.

See Also

plot3d.hxsurf,read.hxsurf, rgb

Other amira: amiratype(), is.amiramesh(), read.amiramesh(), read.hxsurf()

Other hxsurf: as.hxsurf(), as.mesh3d(), materials(), plot3d.hxsurf(), read.hxsurf(), subset.hxsurf()

---

write.neuron *Write out a neuron skeleton or mesh in any of the file formats we know about*

Description

If file is not specified the neuron's InputFileName field will be checked (for a dotprops object it will be the 'file' attribute). If this is missing there will be an error. If dir is specified it will be combined with basename(file). If file is specified but format is not, it will be inferred from file's extension.

Usage

```
write.neuron(
  n,
  file = NULL,
  dir = NULL,
  format = NULL,
  ext = NULL,
  Force = FALSE,
  MakeDir = TRUE,
  metadata = NULL,
  ...
)
```

Arguments

n	A neuron
file	Path to output file
dir	Path to directory (this will replace dirname(file) if specified)

format	Unique abbreviation of one of the registered file formats for neurons including 'swc', 'hxlineset', 'hxskel' (skeletons) and 'ply', 'obj' (neuron meshes). If no format can be extracted from the filename or the ext parameter, then it defaults to 'swc' for skeletons and 'ply' for meshes.
ext	Will replace the default extension for the filetype and should include the period e.g. ext='.amiramesh' or ext='_reg.swc'. The special value of ext=NA will prevent the extension from being changed or added e.g. if the desired file name does not have an extension.
Force	Whether to overwrite an existing file
MakeDir	Whether to create directory implied by file argument.
metadata	Whether to encode some metadata in the header file (currently only supported for SWC format). Either a data.frame or TRUE to indicate that the attached data.frame should be written. Default FALSE.
...	Additional arguments passed to selected writer function

Details

Note that if file does not have an extension then the default extension for the specified format will be appended. This behaviour can be suppressed by setting ext=NA.

If you find that some software rejects your SWC files, try setting normalise.ids=TRUE (see examples). This will ensure that the vertex ids are sequentially ascending integers (1:N). The default value of normalise.ids=NA will normalise PointNo vertex ids only when a vertex is connected (by the Parent field) to a vertex that had not yet been defined. Many readers make the assumption that this is true. When normalise.ids=FALSE the vertex ids will not be touched.

Value

return value

See Also

write.neurons, read.neuron, fileformats, saveRDS

Examples

```
# show the currently registered file formats that we can write
fileformats(class='neuron', write=TRUE)
## Not run:
# write neuron to "myneuron.swc" in SWC format
write.neuron(Cell07PNs[[1]], file='myneuron.swc')
# write in SWC format, normalising the integer ids that label every node
# (this is required by some SWC readers e.g. Fiji)
write.neuron(Cell07PNs[[1]], file='myneuron.swc', normalise.ids=TRUE)
# write out "myneuron.swc" in SWC format withour the final extension
write.neuron(Cell07PNs[[1]], file='myneuron.swc')
# write out "myneuron.amiramesh" in Amira hxlineset format
write.neuron(Cell07PNs[[1]], format = 'hxlineset', file='myneuron.amiramesh')

# write out "myneuron.am" in Amira hxlineset format
```

```
write.neuron(Cell07PNs[[1]], format = 'hxlineset', file='myneuron')

# write a mesh in Stanford ply
write.neuron(MBL.surf, file = 'MBL.surf.ply')
# ... or Wavefront obj format
write.neuron(MBL.surf, file = 'MBL.surf.obj')
# specify the format directly if not evident from file suffix
# not recommended though as will probably just cause trouble when reading
write.neuron(MBL.surf, file = 'MBL.surf', format='obj')

## End(Not run)
```

`write.neuronlistfh` *Write out a neuronlistfh object to an RDS file*

Description

Write out a neuronlistfh object to an RDS file

Usage

```
write.neuronlistfh(x, file = attr(x, "file"), overwrite = FALSE, ...)
```

Arguments

`x`	The neuronlistfh object to write out
`file`	Path where the file will be written (see details)
`overwrite`	Whether to overwrite an existing file
`...`	Additional parameters passed to `saveRDS`

Details

This function writes the main neuronlistfh object to disk, but makes no attempt to touch/verify the associated object files.

if `file` is not specified, then the function will first check if `x` has a `'file'` attribute. If that does not exist, then `attr(x,'db')@dir`, the backing `filehash` database directory, is inspected. The save path `file` will then be constructed by taking the directory one up from the database directory and using the name of the neuronlistfh object with the suffix '.rds'. e.g. write.neuronlistfh(kcs20) with db directory '/my/path/dps/data' will be saved as '/my/path/dps/kcs20.rds'

Note that if x has a `'file'` attribute (set by `read.neuronlistfh`) then this will be removed before the file is saved (since the file attribute must be set on read to ensure that we know exactly which file on disk was the source of the object in memory).

See Also

`saveRDS`

Other neuronlistfh: `[.neuronlistfh()`, `neuronlistfh()`, `read.neuronlistfh()`, `remotesync()`

`write.neurons` *Write neurons from a neuronlist object to individual files, or a zip archive*

Description

Write neurons from a neuronlist object to individual files, or a zip archive

Usage

```
write.neurons(
  nl,
  dir,
  format = NULL,
  subdir = NULL,
  INDICES = names(nl),
  files = NULL,
  include.data.frame = FALSE,
  metadata = FALSE,
  Force = FALSE,
  cl = NULL,
  ...
)
```

Arguments

`nl`	neuronlist object
`dir`	directory to write neurons, or path to zip archive (see Details).
`format`	Unique abbreviation of one of the registered file formats for neurons including 'swc', 'hxlineset', 'hxskel' (skeletons) and 'ply', 'obj' (neuron meshes). If no format can be extracted from the filename or the `ext` parameter, then it defaults to 'swc' for skeletons and 'ply' for meshes.
`subdir`	String naming field in neuron that specifies a subdirectory OR expression to evaluate in the context of neuronlist's df attribute
`INDICES`	Character vector of the names of a subset of neurons in neuronlist to write.
`files`	Character vector or expression specifying output filenames. See examples and `write.neuron` for details.
`include.data.frame`	Whether to include the metadata when writing a zip file (it will be called `"write.neurons.dataframe.rd`
`metadata`	Whether to encode some metadata in the header file (currently only supported for SWC format). Either a data.frame or `TRUE` to indicate that the attached data.frame should be written. Default `FALSE`.
`Force`	Whether to overwrite an existing file

cl	Either the integer number of cores to use for parallel writes (2 or 3 seem useful) or a cluster object created by makeCluster. See the cl argument of pbsapply for details.
...	Additional arguments passed to write.neuron

Details

See write.neuron for details of how to specify the file format/extension/name of the output files and how to establish what output file formats are available. A zip archive of files can be written by specifying a value of dir that ends in .zip. When rds files (R's binary data representation, which is compressed by default) are stored inside a zip file the are not further compressed (zip option 0).

Value

the path to the output file(s), absolute when this is a zip file.

Author(s)

jefferis

See Also

write.neuron, read.neurons, fileformats

Other neuronlist: *.neuronlist(), is.neuronlist(), neuronlist-dataframe-methods, neuronlistfh(), neuronlistz(), neuronlist(), nlapply(), read.neurons()

Examples

```
## Not run:
# write some neurons in swc format
write.neurons(Cell07PNs, dir="testwn", format='swc')
# write some neurons in swc format for picky software
write.neurons(Cell07PNs, dir="testwn", format='swc', normalise.ids=TRUE)
# write some neurons in swc format and zip them up
write.neurons(Cell07PNs, dir="testwn.zip", format='swc')

# write some neurons in R's native RDS format using 3 cores for
# parallel writes and then zip them up (storing rather than compressing)
write.neurons(Cell07PNs, dir="testwn.zip", format='rds', cl=3)

# write some neurons in Amira hxlineset format
write.neurons(Cell07PNs, dir="testwn", format='hxlineset')

# write some neuron meshes in Stanford ply format (the default for meshes)
write.neurons(myneurons, dir="testwn")
# specify the format to avoid a warning. Write to a zip file.
write.neurons(myneurons, dir="testmeshes.zip", format='ply')
# Wavefront obj format
write.neurons(myneurons, dir="testwn", format='obj')

# organise new files in directory hierarchy by glomerulus and Scored.By field
```

```
write.neurons(Cell07PNs,dir="testwn",
  subdir=file.path(Glomerulus,Scored.By),format='hxlineset')
# ensure that the neurons are named according to neuronlist names
write.neurons(Cell07PNs, dir="testwn", files=names(Cell07PNs),
  subdir=file.path(Glomerulus,Scored.By),format='hxlineset')
# only write a subset
write.neurons(subset(Cell07PNs, Scored.By="ACH"),dir="testwn2",
  subdir=Glomerulus,format='hxlineset')
# The same, but likely faster for big neuronlists
write.neurons(Cell07PNs, dir="testwn3",
  INDICES=subset(Cell07PNs,Scored.By="ACH",rval='names'),
  subdir=Glomerulus,format='hxlineset')
# set file name explicitly using a field in data.frame
write.neurons(subset(Cell07PNs, Scored.By="ACH"),dir="testwn4",
  subdir=Glomerulus, files=paste0(ID,'.am'), format='hxlineset')

## End(Not run)
```

`write.nrrd` *Write data and metadata to NRRD file or create a detached NRRD (nhdr) file.*

Description

`write.nrrd` writes an array, vector or im3d object to a NRRD file. When `x` is an `im3d` object, appropriate spatial calibration fields are added to the header.

`write.nrrd.header` writes a nrrd header file.

`write.nrrd.header.for.file` makes a detached NRRD (**nhdr**) file that points at another image file on disk, making it NRRD compatible. This can be a convenient way to make NRRD inputs for other tools e.g. CMTK and also allows the same data block to pointed to by different nhdr files with different spatial calibration.

Usage

```
write.nrrd(
  x,
  file,
  enc = c("gzip", "raw", "text"),
  dtype = c("float", "byte", "short", "ushort", "int", "double"),
  header = attr(x, "header"),
  endian = .Platform$endian,
  datafile = NULL
)

write.nrrd.header(header, file)

write.nrrd.header.for.file(infile, outfile = NULL)
```

Arguments

x	Data to write as an array, vector or im3d object.
file	Character string naming an output file (a detached nrrd header when file has extension 'nhdr').
enc	One of three supported nrrd encodings ("gzip", "raw", "text")
dtype	The data type to write. One of "float","byte", "short", "ushort", "int", "double"
header	List containing fields of nrrd header - see *Header* section.
endian	One of "big" or "little". Defaults to .Platform$endian.
datafile	Optional name of separate file into which data should be written (see details).
infile, outfile	Path to input and output file for write.nrrd.header.for.file. If outfile is NULL (the default) then it will be set to <infilestem.nhdr>.

Detached NRRDs

NRRD files can be written in *detached* format (see https://teem.sourceforge.net/nrrd/format.html#detached) in which a text **nhdr** file is used to described the contents of a separate (usually binary) data file. This means that the nhdr file can be inspected and edited with a text editor, while the datablock can be in a completely raw format that can be opened even by programs that do not understand the NRRD format. Furthermore detached NRRD header files can be written to accompany non-NRRD image data so that it can be opened by nrrd readers.

If file has extension .nhdr *or* datafile is non-NULL, then write.nrrd will write a separate datafile. If datafile is set, then it is interpreted as specifying a path relative to the **nhdr** file. If datafile is not specified then default filenames will be chosen according to the encoding following the conventions of the teem library:

- raw '<nhdrstem>.raw'
- gzip '<nhdrstem>.raw.gz'
- text '<nhdrstem>.ascii'

Data file paths

When a detached NRRD is written, the datafile can be specified either as *relative* or *absolute* path. Relative paths are strongly recommended - the best place is right next to the datafile. Relative paths are always specified with respect to the location of the **nhdr** file.

The datafile argument is not processed by write.nrrd so it is up to the caller to decide whether a relative or absolute path will be used.

For write.nrrd.header.for.file if outfile is not specified then the nhdr file will be placed next to the original image stack and the datafile field will therefore just be basename(infile). If outfile is specified explicitly, then datafile will be set to the full path in the infile argument. Therefore if you wish to specify outfile, you *must* set the current working directory (using setwd) to the location in which outfile will be written to ensure that the path to the datafile is correct. A future TODO would add the ability to convert an absolute datafile path to a relative one (by finding the common path between datafile and nhdr folders).

Header

For `write.nrrd`, arguments `enc`, `dtype`, and `endian` along with the dimensions of the input (`x`) will override the corresponding NRRD header fields from any supplied `header` argument. See https://teem.sourceforge.net/nrrd/format.html for details of the NRRD fields.

See Also

`write.im3d`, `.Platform`

Other nrrd: `is.nrrd()`, `nrrd.voxdims()`, `read.nrrd()`

`write.vtk` *Write object to VTK file*

Description

Write object to VTK file

Usage

```
write.vtk(x, file, ...)

## S3 method for class 'neuron'
write.vtk(
  x,
  file,
  datatype = c("float", "double"),
  title = file,
  WriteAllSubTrees = TRUE,
  ...
)
```

Arguments

`x`	Object to write
`file`	Path to output file
`...`	Additional arguments to methods
`datatype`	The VTK data type (one of float or double)
`title`	Title of the .vtk file (defaults to file)
`WriteAllSubTrees`	Whether to write all subtrees in the neuron or just the main tree.

Examples

```
## Not run:
n=Cell07PNs[[1]]
write.vtk(n, paste0(n$NeuronName, ".vtk"))
write.neuron(n, paste0(n$NeuronName, ".vtk"))

## End(Not run)
```

`xform` *Transform the 3D location of objects such as neurons*

Description

`xform` is designed to operate on a variety of data types, especially objects encapsulating neurons. `xform` depends on two specialised downstream functions `xformpoints` and `xformimage`. These are user visible any contain some useful documentation, but should only be required for expert use; in almost all circumstances, you should use only `xform`.

`xform.character` is designed to work with files on disk. Presently it is restricted to images, although other datatypes may be supported in future.

Usage

```
xform(x, reg, ...)

## Default S3 method:
xform(x, reg, na.action = c("warn", "none", "drop", "error"), ...)

## S3 method for class 'character'
xform(x, reg, ...)

## S3 method for class 'list'
xform(x, reg, FallBackToAffine = TRUE, na.action = "error", ...)

## S3 method for class 'shape3d'
xform(x, reg, FallBackToAffine = TRUE, na.action = "error", ...)

## S3 method for class 'mesh3d'
xform(x, reg, FallBackToAffine = TRUE, na.action = "error", ...)

## S3 method for class 'neuron'
xform(x, reg, FallBackToAffine = TRUE, na.action = "error", ...)

## S3 method for class 'data.frame'
xform(x, reg, subset = NULL, ...)

## S3 method for class 'dotprops'
xform(x, reg, FallBackToAffine = TRUE, ...)
```

```
## S3 method for class 'neuronlist'
xform(
  x,
  reg,
  subset = NULL,
  ...,
  OmitFailures = NA,
  VectoriseRegistrations = FALSE,
  TransformDFCoords = TRUE
)
```

Arguments

x	an object to transform
reg	A registration defined by a matrix, a function, a cmtkreg object, or a character vector specifying a path to one or more registrations on disk (see Registrations section).
...	additional arguments passed to methods and eventually to xformpoints
na.action	How to handle NAs. NB drop may not work for some classes.
FallBackToAffine	Whether to use an affine transform when a cmtk warping transformation fails.
subset	For xform.neuronlist indices (character/logical/integer) that specify a subset of the members of x to be transformed.
OmitFailures	Whether to omit neurons for which FUN gives an error. The default value (NA) will result in nlapply stopping with an error message the moment there is an error. For other values, see details.
VectoriseRegistrations	When FALSE, the default, each element of reg will be applied sequentially to each element of x. When TRUE, it is assumed that there is one element of reg for each element of x.
TransformDFCoords	If the metadata data.frame attached to x includes columns that look like x,y,z coordinates, transform those as well.

Details

Methods are provided for some specialised S3 classes. Further methods can of course be constructed for user-defined S3 classes. However this will probably not be necessary if the xyzmatrix and `xyzmatrix<-` generics are suitably overloaded *and* the S3 object inherits from list.

Note that given the behaviour of the xyzmatrix functions, the xform.data.frame method will transform the x,y,z or X,Y,Z columns of a data.frame if the data.frame has more than 3 columns, erroring out if no such unique columns exist.

TODO get this to work for matrices with more than 3 columns by working on xyzmatrix definition.

For the xform.dotprops method, dotprops tangent vectors will be recalculated from scratch after the points have been transformed (even though the tangent vectors could in theory be transformed

more or less correctly). When there are multiple transformations, xform will take care to carry out all transformations before recalculating the vectors.

With xform.neuronlist, if you want to apply a different registration to each object in the neuronlist x, then you should use VectoriseRegistrations=TRUE.

When x's attached data.frame contains columns called x,y,z or X,Y,Z then these are assumed to be coordinates and also transformed when TransformDFCoords=TRUE (the default). This provides a mechanism for transforming the soma positions of neuronlist objects containing dotprops objects (which do not otherwise store the soma position). Note that if transformation fails, a warning will be issued and the points will be replaced with NA values.

Registrations

When reg is a character vector, xform's specialised downstream functions will check to see if it defines a path to one (or more) registrations on disk. These can be of two classes

- CMTK registrations
- reglist objects saved in R's RDS format (see readRDS) which can contain any sequence of registrations supported by nat.

If the path does indeed point to a CMTK registration, this method will hand off to xformpoints.cmtkreg or xformimages.cmtkreg. In this case, the character vector may optionally have an attribute, 'swap', a logical vector of the same length indicating whether the transformation direction should be swapped. At the moment only CMTK registration files are supported.

If reg is a character vector of length >=1 defining a sequence of registration files on disk they should proceed from sample to reference.

Where reg is a function, it should have a signature like myfun(x,), ... where the ... **must** be provided in order to swallow any arguments passed from higher level functions that are not relevant to this particular transformation function.

See Also

xformpoints

Examples

```
## Not run:
kc1=kcs20[[1]]
kc1.default=xform(kc1,function(x,...) x)
stopifnot(isTRUE(all.equal(kc1,kc1.default)))
kc1.5=xform(kc1,function(x,...) x, k=5)
stopifnot(isTRUE(all.equal(kc1.5,kc1.default)))
kc1.20=xform(kc1,function(x,...) x, k=20)
stopifnot(!isTRUE(all.equal(kc1,kc1.20)))

# apply two registrations converting sample->IS2->JFRC2
reg_seq=c("IS2_sample.list", "JFRC2_IS2.list")
xform(kc1, reg_seq)
# apply two registrations, swapping the direction of the second one
# i.e. sample -> IS2 -> FCWB
```

```
reg_seq=structure(c("IS2_sample.list", "IS2_FCWB.list"), swap=c(FALSE, TRUE))
xform(kc1, reg_seq)

## End(Not run)
## Not run:
# apply reg1 to Cell07PNs[[1]], reg2 to Cell07PNs[[2]] etc
regs=c(reg1, reg2, reg3)
nx=xform(Cell07PNs[1:3], reg=regs, VectoriseRegistrations=TRUE)

## End(Not run)
```

`xformimage` *Transform image files using a registration or affine matrix*

Description

You should almost always call `xform` rather calling than`xformimage` directly.

Usage

```
xformimage(reg, image, ...)

## S3 method for class 'character'
xformimage(reg, image, ...)

## S3 method for class 'cmtkreg'
xformimage(
  reg,
  image,
  transformtype = c("warp", "affine"),
  direction = NULL,
  ...
)

## S3 method for class 'reglist'
xformimage(reg, image, ...)

## Default S3 method:
xformimage(reg, image, ...)
```

Arguments

`reg`	A registration defined by a matrix or a `cmtkreg` object, or a character vector specifying a path to a CMTK registration on disk (see details). If `reg` is a character vector of length >1 defining a sequence of registration files on disk they should proceed from sample to reference.
`image`	Nx3 matrix of image

`...`	Additional arguments passed to methods (and then eventually to `cmtk.reformatx`)
`transformtype`	Which transformation to use when the CMTK file contains both warp (default) and affine (TODO)
`direction`	Whether to transform image from sample space to reference space (called **forward** by CMTK) or from reference to sample space (called **inverse** by CMTK). Default (when `NULL` is forward).

Details

When passed a character vector, xformimage will check to see if it defines a path containing CMTK registration erroring out if this is not the case. If the path does indeed point to a CMTK registration, this method will hand off to xformimage.cmtkreg. A future TODO would be to provide a mechanism for extending this behaviour for other registration formats. If a list of transformations is passed in, these transformations are passed to the cmtk reformatx tool in the order received. Note that there is presently no support for

- using the inverse of a registration
- specifying a mask
- passing additional arguments to `reformatx`

Note that the direction of CMTK registrations can be the source of much confusion. This is because CMTK defines the *forward* direction as the transform required to reformat an image in *sample* (floating) space to an image in *template* space. Since this operation involves filling a regular grid in template space by looking up the corresponding positions in sample space, the transformation that is required is (somewhat counterintuitively) the one that maps template to sample. However in neuroanatomical work, one often has points in sample space that one would like to transform into template space. Here one needs CMTK's *inverse* transformation.

A second source of confusion is that when there are multiple transformations, CMTK's reformatx tool (wrapped by `cmtk.reformatx`) expects them to be listed:

`ref_intermediate.list intermediate_sample.list`

where `ref_intermediate.list` is the CMTK registration obtained with ref as target/reference and intermediate as sample/floating image.

For consistency, all `xform.*` methods expect multiple registrations to be listed from sample to reference and this order is then swapped when they are passed on to `cmtk.reformatx`.

whereas CMTK's `streamxform` tool (wrapped by `xformpoints`) expects them in the opposite order.

Value

Character vector with path to transformed image.

See Also

`cmtk.reformatx`, `xformpoints`, `xform`

`xformpoints` *Transform 3D points using a registration, affine matrix or function*

Description

You should almost always call `xform` rather calling than`xformpoints` directly.

Usage

```
xformpoints(reg, points, ...)

## S3 method for class 'character'
xformpoints(reg, points, ...)

## S3 method for class 'cmtkreg'
xformpoints(
  reg,
  points,
  transformtype = c("warp", "affine"),
  direction = NULL,
  FallBackToAffine = FALSE,
  ...
)

## S3 method for class 'reglist'
xformpoints(reg, points, ...)

## Default S3 method:
xformpoints(reg, points, ...)
```

Arguments

`reg`	A registration defined by a matrix, a function, a `cmtkreg` object, a `reglist` object containing a sequence of arbitrary registrations, or a character vector specifying path(s) to registrations on disk (see details).
`points`	Nx3 matrix of points
`...`	Additional arguments passed to methods
`transformtype`	Which transformation to use when the CMTK file contains both warp (default) and affine
`direction`	Whether to transform points from sample space to reference space (called **inverse** by CMTK) or from reference to sample space (called **forward** by CMTK). Default (when `NULL` is inverse).
`FallBackToAffine`	Whether to use the affine transformation for points that fail to transform under a warping transformation.

Details

If a list of transformations is passed in, these transformations are performed in sequence order, such that `xformpoints(c(a,b,c), x)` == `xformpoints(c, (xformpoints(b, xformpoints(a, x))))`

Note that the direction of CMTK registrations can be the source of much confusion. This is because CMTK defines the *forward* direction as the transform required to reformat an image in *sample* (floating) space to an image in *template* space. Since this operation involves filling a regular grid in template space by looking up the corresponding positions in sample space, the transformation that is required is (somewhat counterintuitively) the one that maps template to sample. However in neuroanatomical work, one often has points in sample space that one would like to transform into template space. Here one needs the *inverse* transformation.

`xyzmatrix` *Get and assign coordinates for classes containing 3D vertex data*

Description

`xyzmatrix` gets coordinates from objects containing 3D vertex data

`xyzmatrix.list` will parse a list containing triplets of 3 numeric values.

`xyzmatrix<-` assigns xyz elements of neuron or dotprops object and can also handle matrix like objects with columns named X, Y, Z or x, y, z.

`xyzmatrix2str` will convert the XYZ locations associated with an object to a character vector (by default comma separated).

`xyzmatrix2list` will convert the Nx3 matrix of XYZ locations associated with an object to a list of length N with each element a vector of length 3.

Usage

```
xyzmatrix(x, ...)

## Default S3 method:
xyzmatrix(x, y = NULL, z = NULL, ...)

## S3 method for class 'list'
xyzmatrix(x, empty2na = TRUE, ...)

## S3 method for class 'character'
xyzmatrix(x, ...)

## S3 method for class 'neuron'
xyzmatrix(x, ...)

## S3 method for class 'neuronlist'
xyzmatrix(x, ...)

## S3 method for class 'shapelist3d'
```

```
xyzmatrix(x, ...)

## S3 method for class 'dotprops'
xyzmatrix(x, ...)

## S3 method for class 'hxsurf'
xyzmatrix(x, ...)

## S3 method for class 'igraph'
xyzmatrix(x, ...)

## S3 method for class 'mesh3d'
xyzmatrix(x, ...)

xyzmatrix(x) <- value

## S3 replacement method for class 'character'
xyzmatrix(x) <- value

xyzmatrix2str(x, format = "%g,%g,%g", sep = NULL)

xyzmatrix2list(x)

## S3 replacement method for class 'neuron'
xyzmatrix(x) <- value

## S3 replacement method for class 'dotprops'
xyzmatrix(x) <- value

## S3 replacement method for class 'hxsurf'
xyzmatrix(x) <- value

## S3 replacement method for class 'igraph'
xyzmatrix(x) <- value

## S3 replacement method for class 'shape3d'
xyzmatrix(x) <- value

## S3 replacement method for class 'mesh3d'
xyzmatrix(x) <- value

## S3 replacement method for class 'neuronlist'
xyzmatrix(x) <- value

## S3 replacement method for class 'shapelist3d'
xyzmatrix(x) <- value
```

Arguments

x	object containing 3D coordinates
...	additional arguments passed to methods
y, z	separate y and z coordinates
empty2na	Whether or not to convert empty elements (NULL or list()) into NAs. Default TRUE.
value	Nx3 matrix specifying new xyz coords
format	A sprintf compatible format string. The default will give comma separated values.
sep	A character vector specifying a separator string. Overrides format when present. The default value of format is equivalent to sep=",".

Details

Note that xyzmatrix can extract or set 3D coordinates in a matrix or data.frame that **either** has exactly 3 columns **or** has 3 columns named X,Y,Z or x,y,z. As of Nov 2020, if these columns are character vectors, they will be correctly converted to numeric (with a warning for any NA values). As of Jan 2021 if x is a numeric vector containing exactly 3 numbers it will be parsed as a 1x3 matrix. Support has also been added for setting a list containing 3-vectors in each element.

Value

For xyzmatrix: Nx3 matrix containing 3D coordinates

For xyzmatrix<-: Original object with modified coords

Getting and setting from character vectors

xyzmatrix can also both get and set 3D coordinates from a character vector (including a single data frame column) in which each string encodes all 3 coordinates e.g. "-1, 4, 10". It should handle a range of separators such as spaces, tabs, commas, semicolons and ignore extraneous characters such as brackets. Note that data are rounded by zapsmall in the replacement version to try to avoid cases where rounding errors result in long strings of digits to the right of the decimal place.

Replacement into character vectors introduces a number of corner cases when there are not exactly 3 numbers to replace in the target vector. We handle them as follows:

- 0 values in target, >0 in replacement: use a default pattern
- 1-2 values in target, same number of "good" values in replacement: insert those replacement value
- 1-2 values in target, different number of values in replacement: use default pattern, give a warning

The default pattern will be the first entry in x with 3 numbers. Should there not be such a value, then the pattern will be "x, y, z".

See Also

xyzmatrix

Examples

```
# see all available methods for different classes
methods('xyzmatrix')
# ... and for the assignment method
methods('xyzmatrix<-')

# basic usage
xyzmatrix(cbind(-1,2,3))

# character vector - useful e.g. when encoded in 1 column of a table
str123="(-1,+2,3)"
xyzmatrix(str123)
# replace
xyzmatrix(str123) <- xyzmatrix(str123)/3
str123
xyzmatrix(str123) <- xyzmatrix(str123)*3
str123
n=Cell07PNs[[1]]
xyzmatrix(n)<-xyzmatrix(n)
stopifnot(isTRUE(
  all.equal(xyzmatrix(n),xyzmatrix(Cell07PNs[[1]]))
))
head(xyzmatrix2str(kcs20[[1]]))
head(xyzmatrix2str(kcs20[[1]], format="(%g;%g;%g)"))
# if you want to process the xyz locations (here rounded to nearest nm)
# you must extract them from complex objects yourself
xyzmatrix2str(round(xyzmatrix(kcs20[[1]])*1000), format="%d,%d,%d")[1:3]
xyzmatrix2list(kcs20[[1]])[1:2]
```

[.neuronlistfh *Extract from neuronlistfh object or its attached data.frame*

Description

`[.neuronlistfh` extracts either a sublist from a neuronlistfh (converting it to a regular in memory list in the process) *or* its attached data.frame.

Usage

```
## S3 method for class 'neuronlistfh'
x[i, j, drop]
```

Arguments

`x`	A neuronlistfh object
`i, j`	elements to extract or replace. Numeric, logical or character or, for the [get method, empty. See details and the help for `[.data.frame`.
`drop`	logical. If `TRUE` the result is coerced to the lowest possible dimension. The default is to drop if only one column is left, but **not** to drop if only one row is left.

Details

Note that if i is a numeric or logical indexing vector, it will be converted internally to a vector of names by using the (sorted) names of the objects in x (i.e. names(x)[i])

Value

A new in-memory `neuronlist` or when using two subscripts, a `data.frame` - see examples.

See Also

`neuronlistfh`, `[.neuronlist`, `[.data.frame`, `[<-.data.frame`,

Other neuronlistfh: `neuronlistfh()`, `read.neuronlistfh()`, `remotesync()`, `write.neuronlistfh()`

Examples

```
# make a test neuronlistfh backed by a temporary folder on disk
tf=tempfile('kcs20fh')
kcs20fh<-as.neuronlistfh(kcs20, dbdir=tf)

# get first neurons as an in memory neuronlist
class(kcs20fh[1:3])

# extract attached data.frame
str(kcs20fh[,])
# or part of the data.frame
str(kcs20fh[1:2,1:3])

# data.frame assignment (this one changes nothing)
kcs20fh[1:2,'gene_name'] <- kcs20fh[1:2,'gene_name']

# clean up
unlink(tf, recursive=TRUE)
```

Index