Package ‘fsbrain’

February 3, 2024

Type Package

Title Managing and Visualizing Brain Surface Data

Version 0.5.5

Maintainer Tim Schäfer <ts+code@rcmd.org>

Description Provides high-level access to neuroimaging data from standard software packages like 'FreeSurfer' <http://freesurfer.net/> on the level of subjects and groups. Load morphometry data, surfaces and brain parcellations based on atlases. Mask data using labels, load data for specific atlas regions only, and visualize data and statistical results directly in 'R'.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/dfsp-spirit/fsbrain

BugReports https://github.com/dfsp-spirit/fsbrain/issues

Imports reshape, freesurferformats (>= 0.1.17), pkgfilecache (>= 0.1.1), rgl, squash, fields, viridis, data.table, magick, methods

Suggests knitr, rmarkdown, testthat (>= 2.1.0), sphereplot (>= 1.5), misc3d, RColorBrewer, Rvcg (>= 0.20.2), igraph, pracma

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation no

Author Tim Schäfer [aut, cre] (<https://orcid.org/0000-0002-3683-8070>)

Repository CRAN

Date/Publication 2024-02-03 12:30:02 UTC

R topics documented:

`alphablend` *Perform alpha blending for pairs of RGBA colors.*

Description

Implements the *over* alpha blending operation.

Usage

```
alphablend(front_color, back_color, silent = TRUE)
```

Arguments

`front_color`	rgba color strings, the upper color layer or foreground
`back_color`	rgba color strings, the lower color layer or background
`silent`	logical, whether to suppress messages

Value

rgba color strings, the alpha-blended colors

References

see the *Alpha blending* section on https://en.wikipedia.org/wiki/Alpha_compositing

See Also

Other color functions: `desaturate()`

`annot.outline` *Compute outline vertex colors from annotation.*

Description

For each region in an atlas, compute the outer border and color the respective vertices in the region-specific color from the annot's colortable.

Usage

```
annot.outline(
  annotdata,
  surface_mesh,
  background = "white",
  silent = TRUE,
  expand_inwards = 0L,
  outline_color = NULL,
  limit_to_regions = NULL
)
```

Arguments

`annotdata`	an annotation, as returned by functions like `subject.annot`. If a character string, interpreted as a path to a file containing such data, and loaded with `freesurferformats::read.fs.annot`
`surface_mesh`	brain surface mesh, as returned by functions like `subject.surface` or `read.fs.surface`. If a character string, interpreted as a path to a file containing such data, and loaded with `freesurferformats::read.fs.surface`

`background`	color, the background color to assign to the non-border parts of the regions. Defaults to 'white'.
`silent`	logical, whether to suppress status messages.
`expand_inwards`	integer, additional thickness of the borders. Increases computation time, defaults to 0L.
`outline_color`	NULL or a color string (like 'black' or '#000000'), the color to use for the borders. If left at the default value 'NULL', the colors from the annotation color lookup table will be used.
`limit_to_regions`	vector of character strings or NULL, a list of regions for which to draw the outline (see `get.atlas.region.names`). If NULL, all regions will be used. If (and only if) this parameter is used, the 'outline_color' parameter can be a vector of color strings, one color per region.

Value

vector of colors, one color for each mesh vertex

Note

Sorry for the computational time, the mesh datastructure is not ideal for neighborhood search.

`annot.outline.border.vertices` *Compute the border vertices for each region in an annot.*

Description

Compute the border vertices for each region in an annot.

Usage

```
annot.outline.border.vertices(
  annotdata,
  surface_mesh,
  silent = TRUE,
  expand_inwards = 0L,
  limit_to_regions = NULL
)
```

Arguments

`annotdata`	an annotation, as returned by functions like `subject.annot`. If a character string, interpreted as a path to a file containing such data, and loaded with `freesurferformats::read.fs.annot`

`surface_mesh`	brain surface mesh, as returned by functions like `subject.surface` or `read.fs.surface`. If a character string, interpreted as a path to a file containing such data, and loaded with `freesurferformats::read.fs.surface`
`silent`	logical, whether to suppress status messages.
`expand_inwards`	integer, additional thickness of the borders. Increases computation time, defaults to 0L.
`limit_to_regions`	vector of character strings or NULL, a list of regions for which to draw the outline (see `get.atlas.region.names`). If NULL, all regions will be used. If (and only if) this parameter is used, the 'outline_color' parameter can be a vector of color strings, one color per region.

Value

named list, the keys are the region names and the values are vectors of integers encoding vertex indices.

`apply.label.to.morphdata`

Load a label from file and apply it to morphometry data.

Description

This function will set all values in morphdata which are *not* part of the label loaded from the file to NA (or whatever is specified by 'masked_data_value'). This is typically used to ignore values which are not part of the cortex (or any other label) during your analysis.

Usage

```
apply.label.to.morphdata(
  morphdata,
  subjects_dir,
  subject_id,
  hemi,
  label,
  masked_data_value = NA
)
```

Arguments

`morphdata`	numerical vector, the morphometry data for one hemisphere
`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`label`	string, ‘fs.label‘ instance, or label vertex data. If a string, interpreted as the file name of the label file, without the hemi part (if any), optionally including the ’.label’ suffix. E.g., ’cortex.label’ or ’cortex’ for ’?h.cortex.label’.
`masked_data_value`	numerical, the value to set for all morphometry data values of vertices which are *not* part of the label. Defaults to NA.

Value

numerical vector, the masked data.

See Also

Other label functions: `apply.labeldata.to.morphdata()`, `subject.lobes()`, `subject.mask()`, `vis.labeldata.on.subject()`, `vis.subject.label()`

Other morphometry data functions: `apply.labeldata.to.morphdata()`, `group.morph.native()`, `group.morph.standard()`, `subject.morph.native()`, `subject.morph.standard()`

`apply.labeldata.to.morphdata`

Apply a label to morphometry data.

Description

This function will set all values in morphdata which are *not* part of the labeldata to NA (or whatever is specified by ’masked_data_value’). This is typically used to ignore values which are not part of the cortex (or any other label) during your analysis.

Usage

```
apply.labeldata.to.morphdata(morphdata, labeldata, masked_data_value = NA)
```

Arguments

`morphdata`	numerical vector, the morphometry data for one hemisphere
`labeldata`	integer vector or ‘fs.label‘ instance. A label as returned by `subject.label`.
`masked_data_value`	numerical, the value to set for all morphometry data values of vertices which are *not* part of the label. Defaults to NA.

Value

numerical vector, the masked data.

See Also

Other label functions: `apply.label.to.morphdata()`, `subject.lobes()`, `subject.mask()`, `vis.labeldata.on.subject`
`vis.subject.label()`

Other morphometry data functions: `apply.label.to.morphdata()`, `group.morph.native()`, `group.morph.standard()`, `subject.morph.native()`, `subject.morph.standard()`

`apply.transform` *Apply matmult transformation to input.*

Description

Apply affine transformation, like a *vox2ras_tkr* transformation, to input. This is just matrix multiplication for different input objects.

Usage

```
apply.transform(object, matrix_fun)
```

Arguments

`object`	numerical vector/matrix or Triangles3D instance, the coorindates or object to transform.
`matrix_fun`	a 4x4 affine matrix or a function returning such a matrix. If ‘NULL‘, the input is returned as-is. In many cases you way want to use a matrix computed from the header of a volume file, e.g., the ‘vox2ras‘ matrix of the respective volume. See the ‘mghheader.*‘ functions in the *freesurferformats* package to obtain these matrices.

Value

the input after application of the affine matrix (matrix multiplication)

`arrange.brainview.images` *Combine several brainview images into a new figure.*

Description

Create a new image from several image tiles, the exact layout depends on the number of given images.

Usage

```
arrange.brainview.images(
  brainview_images,
  output_img,
  colorbar_img = NULL,
  silent = TRUE,
  grid_like = TRUE,
  border_geometry = "5x5",
  background_color = "white",
  map_bg_to_transparency = FALSE
)
```

Arguments

`brainview_images`	vector of character strings, paths to the brainview images, usually in PNG format
`output_img`	path to output image that including the file extension
`colorbar_img`	path to the main image containing the separate colorbar, usually an image in PNG format
`silent`	logical, whether to suppress messages
`grid_like`	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally.
`border_geometry`	string, a geometry string passed to `magick::image_border` to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
`background_color`	hex color string, such as "#DDDDDD" or "#FFFFFF". The color to use when extending images (e.g., when creating the border). WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
`map_bg_to_transparency`	logical, whether to map the background_color to transparency for the final PNG export.

Value

named list with entries: 'brainview_images': vector of character strings, the paths to the input images. 'output_img_path': character string, path to the output image. 'merged_img': the magick image instance.

`arrange.brainview.images.grid` *Combine several brainview images as a grid into a new figure.*

Description

Create a new image from several image tiles, the exact layout is a grid with n per row.

Usage

```
arrange.brainview.images.grid(
  brainview_images,
  output_img,
  colorbar_img = NULL,
  silent = TRUE,
  num_per_row = 10L,
  border_geometry = "5x5",
  background_color = "white",
  captions = NULL
)
```

Arguments

`brainview_images`	vector of character strings, paths to the brainview images, usually in PNG format
`output_img`	path to output image that including the file extension
`colorbar_img`	path to the main image containing the separate colorbar, usually an image in PNG format
`silent`	logical, whether to suppress messages
`num_per_row`	positive integer, the number of image tiles per row.
`border_geometry`	string, a geometry string passed to `magick::image_border` to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
`background_color`	hex color string, such as "#DDDDDD" or "#FFFFFF". The color to use when extending images (e.g., when creating the border). WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
`captions`	vector of character strings or NULL, the (optional) text annotations for the images. Useful to print the subject identifier onto the individual tiles. Length must match number of image tiles in 'brainview_images'.

Value

named list with entries: 'brainview_images': vector of character strings, the paths to the input images. 'output_img_path': character string, path to the output image. 'merged_img': the magick image instance.

Note

The tiles are written row-wise, in the order in which they occur in the parameter 'brainview_images'.

`brainviews` *Show one or more views of the given meshes in rgl windows.*

Description

Show one or more views of the given meshes in rgl windows.

Usage

```
brainviews(
  views,
  coloredmeshes,
  rgloptions = rglo(),
  rglactions = list(),
  style = "default",
  draw_colorbar = FALSE,
  background = "white"
)
```

Arguments

`views`	list of strings. Valid entries include: 'si': single interactive view. 'sd_<angle>': single view from angle <angle>. The <angle> part must be one of the strings returned by `get.view.angle.names`. Example: 'sd_caudal'. 'sr': single rotating view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`coloredmeshes`	list of coloredmesh or renderable. A coloredmesh is a named list as returned by the coloredmesh.from.* functions. It has the entries 'mesh' of type tmesh3d, a 'col', which is a color specification for such a mesh.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
`draw_colorbar`	logical, whether to draw a colorbar. WARNING: The colorbar is drawn to a subplot, and this only works if there is enough space for it. You will have to increase the plot size using the 'rlgoptions' parameter for the colorbar to show up. Defaults to FALSE. See `coloredmesh.plot.colorbar.separate` for an alternative.
`background`	the background color for the visualization, e.g., 'white' or '#FF0000'. Note that alpha/transparency is not supported by rgl.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

`get.view.angle.names`

`clip.data` *Clip data at quantiles to remove outliers.*

Description

Set all data values outside the given quantile range to the border values. This is useful to properly visualize morphometry data that includes outliers. These outliers negatively affect the colormap, as all the non-outlier values become hard to distinguish. This function can be used to filter the data before plotting it.

Usage

```
clip.data(data, lower = 0.05, upper = 0.95)
```

Arguments

`data,`	numeric vector. The input data. Can also be a `hemilist`.
`lower,`	numeric. The probability for the lower quantile, defaults to ‘0.05‘.
`upper,`	numeric. The probability for the upper quantile, defaults to ‘0.95‘.

Value

numeric vector. The output data.

See Also

The `clip_fun` function is more convenient when used in `rglactions`, as it allows specification of custom quantiles.

Examples

```
full_data = rnorm(50, 3, 1);
clipped = clip.data(full_data);
```

`clip_fun` *Get data clipping function.*

Description

Get data clipping function to use in `rglactions` as 'trans_fun' to transform data. This is typically used to limit the colorbar in a plot to a certain range. This uses percentiles to clip. Clipping means that values more extreme than the gíven quantiles will be set to the quantile values.

Usage

```
clip_fun(lower = 0.05, upper = 0.95)
```

Arguments

`lower`	numeric. The probability for the lower quantile, defaults to '0.05'.
`upper`	numeric. The probability for the upper quantile, defaults to '0.95'.

Value

a function that takes as argument the data, and clips it to the requested range. I.e., values outside the range will be set to the closest border value. Designed to be used as `rglactions$trans_fun` in vis functions, to limit the colorbar and data range.

See Also

`rglactions`

Examples

```
rglactions = list("trans_fun"=clip_fun(0.10, 0.90));
rglactions = list("trans_fun"=clip_fun());
f = clip_fun();
f(rnorm(100));
```

`cm.cbry` *Get cyan blue red yellow colormap function.*

Description

Get cyan blue red yellow colormap function.

Usage

```
cm.cbry()
```

Note

Returns a diverging palette with negative values in blue/cyan and positive ones in red/yellow, suitable for visualizing data that is centered around zero. Often used for clusters in neuroscience.

`cm.div` *Return the standard fsbrain diverging colormap.*

Description

Return the standard fsbrain diverging colormap.

Usage

```
cm.div(report = FALSE)
```

Arguments

`report`	logical, whether to print a message with a name of the chosen colormap, in format `package::function#palette`.

Note

Returns some diverging palette, suitable for visualizing data that is centered around zero.

`cm.heat` *Return the standard fsbrain heat colormap.*

Description

Return the standard fsbrain heat colormap.

Usage

```
cm.heat(report = FALSE)
```

Arguments

`report`	logical, whether to print a message with a name of the chosen colormap, in format `package::function#palette`.

Note

The heat palette is a sequential, single-hue palette.

`cm.qual` *Return the standard fsbrain qualitative colormap.*

Description

Return the standard fsbrain qualitative colormap.

Usage

```
cm.qual(report = FALSE)
```

Arguments

`report`	logical, whether to print a message with a name of the chosen colormap, in format `package::function#palette`.

Note

Returns some qualitative palette, suitable for visualizing categorical data.

`cm.seq` *Return the standard fsbrain sequential colormap.*

Description

Return the standard fsbrain sequential colormap.

Usage

```
cm.seq(report = FALSE)
```

Arguments

`report`	logical, whether to print a message with a name of the chosen colormap, in format `package::function#palette`.

Note

This returns a sequential, multi-hue palette.

`collayer.bg` *Compute binarized mean curvature surface color layer.*

Description

Compute a binarized mean curvature surface color layer, this is intended as a background color layer. You can merge it with your data layer using `collayers.merge`.

Usage

```
collayer.bg(subjects_dir, subject_id, bg, hemi = "both")
```

Arguments

`subjects_dir`	character string, the FreeSurfer SUBJECTS_DIR.
`subject_id`	character string, the subject identifier.
`bg`	character string, a background name. One of 'curv', 'curv_light', 'sulc', 'sulc_light', or 'aparc'. If this is already a colorlayer in a hemilist, it will be returned as-is.
`hemi`	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.

Value

a color layer, i.e., vector of color strings in a hemilist

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas`(), `collayer.bg.meancurv`(), `collayer.bg.sulc`(), `collayer.from.annotdata`(), `collayer.from.annot`(), `collayer.from.mask.data`(), `collayer.from.morphlike.da`
`collayers.merge`()

`collayer.bg.atlas` *Compute atlas or annotation surface color layer.*

Description

Compute atlas or annotation surface color layer.

Usage

```
collayer.bg.atlas(
  subjects_dir,
  subject_id,
  hemi = "both",
  atlas = "aparc",
  grayscale = FALSE,
  outline = FALSE,
  outline_surface = "white"
)
```

Arguments

`subjects_dir`	character string, the FreeSurfer SUBJECTS_DIR.
`subject_id`	character string, the subject identifier.
`hemi`	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
`atlas`	character string, the atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`grayscale`	logical, whether to convert the atlas colors to grayscale
`outline`	logical, whether to draw an outline only instead of filling the regions. Defaults to 'FALSE'. Instead of passing 'TRUE', one can also pass a list of extra parameters to pass to `annot.outline`, e.g., `outline=list('outline_color'='#000000')`.
`outline_surface`	character string, the surface to load. Only relevant when 'outline' is used. (In that case the surface mesh is needed to compute the vertices forming the region borders.)

Value

a color layer, i.e., vector of color strings in a hemilist

Note

Using 'outline' mode is quite slow, and increasing the border thickness makes it even slower.

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayer.from.morphlike.da` `collayers.merge()`

`collayer.bg.meancurv` *Compute binarized mean curvature surface color layer.*

Description

Compute a binarized mean curvature surface color layer, this is intended as a background color layer. You can merge it with your data layer using `collayers.merge`.

Usage

```
collayer.bg.meancurv(
  subjects_dir,
  subject_id,
  hemi = "both",
  cortex_only = FALSE,
  bin_colors = c("#898989", "#5e5e5e"),
  bin_thresholds = c(0)
)
```

Arguments

`subjects_dir`	character string, the FreeSurfer SUBJECTS_DIR.
`subject_id`	character string, the subject identifier.
`hemi`	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
`cortex_only`	logical, whether to restrict pattern computation to the cortex.
`bin_colors`	vector of two character strings, the two colors to use.
`bin_thresholds`	vector of 1 or 2 double values, the curvature threshold values used to separate gyri from sulci.

Value

a color layer, i.e., vector of color strings in a hemilist

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayer.from.morphlike.data()`, `collayers.merge()`

`collayer.bg.sulc` *Compute binarized sulcal depth surface color layer.*

Description

Compute a binarized sulcal depth surface color layer, this is intended as a background color layer. You can merge it with your data layer using `collayers.merge`.

Usage

```
collayer.bg.sulc(
  subjects_dir,
  subject_id,
  hemi = "both",
  cortex_only = FALSE,
  bin_colors = c("#898989", "#5e5e5e"),
  bin_thresholds = c(0)
)
```

Arguments

`subjects_dir`	character string, the FreeSurfer SUBJECTS_DIR.
`subject_id`	character string, the subject identifier.
`hemi`	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
`cortex_only`	logical, whether to restrict pattern computation to the cortex.
`bin_colors`	vector of two character strings, the two colors to use.
`bin_thresholds`	vector of 1 or 2 double values, the curvature threshold values used to separate gyri from sulci.

Value

a color layer, i.e., vector of color strings in a hemilist

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayer.from.morphlike.da` `collayers.merge()`

`collayer.from.annot` *Compute surface color layer from annotation or atlas data.*

Description

Compute surface color layer from annotation or atlas data.

Usage

```
collayer.from.annot(subjects_dir, subject_id, hemi, atlas)
```

Arguments

`subjects_dir`	character string, the FreeSurfer SUBJECTS_DIR.
`subject_id`	character string, the subject identifier.
`hemi`	character string, one of 'lh', 'rh', or 'both'.
`atlas`	character string, the atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

Value

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the atlas data.

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.mask.data()`, `collayer.from.morphlike.data()`, `collayers.merge()`

`collayer.from.annotdata` *Compute surface color layer from annotation or atlas data.*

Description

Compute surface color layer from annotation or atlas data.

Usage

```
collayer.from.annotdata(lh_annotdata = NULL, rh_annotdata = NULL)
```

Arguments

`lh_annotdata`	loaded annotation data for left hemi, as returned by `subject.annot`
`rh_annotdata`	loaded annotation data for right hemi

Value

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the atlas data.

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayer.from.morphlike.data()`, `collayers.merge()`

---

collayer.from.mask.data

Compute surface color layer from morph-like data.

Description

Compute surface color layer from morph-like data.

Usage

```
collayer.from.mask.data(
  lh_data = NULL,
  rh_data = NULL,
  makecmap_options = list(colFn = label.colFn)
)
```

Arguments

`lh_data`	integer vector, can be NULL
`rh_data`	numerical vector, can be NULL
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.

Value

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the label data.

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.morphlike.data()`, `collayers.merge()`

`collayer.from.morphlike.data`

Compute surface color layer from morph-like data.

Description

Compute surface color layer from morph-like data.

Usage

```
collayer.from.morphlike.data(
  lh_morph_data = NULL,
  rh_morph_data = NULL,
  makecmap_options = list(colFn = cm.seq()),
  return_metadata = FALSE
)
```

Arguments

`lh_morph_data`	numerical vector, can be NULL
`rh_morph_data`	numerical vector, can be NULL
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.
`return_metadata`	logical, whether to return additional metadata as entry 'metadata' in the returned list

Value

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the morph data.

See Also

You can plot the return value using `vis.color.on.subject`.

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayers.merge()`

`collayers.merge` *Merge two or more color layers based on their transparency values.*

Description

Merge several color layers into one based on their transparency and alpha blending. In the final result, the lower layers are visible through the transparent or 'NA' parts (if any) of the upper layers.

Usage

```
collayers.merge(collayers, opaque_background = "#FFFFFF")
```

Arguments

`collayers`	named list, the values must be vectors, matrices or arrays of color strings (as produced by `rgb`. The names are free form and do not really matter. All values must have the same length.
`opaque_background`	a single color string or 'NULL'. If a color string, this color will be used as a final opaque background layer to ensure that the returned colors are all opaque. Pass 'NULL' to skip this, which may result in a return value that contains non-opaque color values.

Value

a color layer, i.e., vector of color strings in a hemilist

See Also

Other surface color layer: `collayer.bg.atlas()`, `collayer.bg.meancurv()`, `collayer.bg.sulc()`, `collayer.bg()`, `collayer.from.annotdata()`, `collayer.from.annot()`, `collayer.from.mask.data()`, `collayer.from.morphlike.data()`

`coloredmesh.from.annot`
Create a coloredmesh from an annotation of an atlas.

Description

Create a coloredmesh from an annotation of an atlas.

Usage

```
coloredmesh.from.annot(
  subjects_dir,
  subject_id,
  atlas,
  hemi,
  surface = "white",
  outline = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`atlas`	string or a loaded annotation. If a string, interpreted as the atlas name that should be loaded to get te annotation. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`outline`	logical, whether to draw an outline only instead of filling the regions. Defaults to FALSE. Only makes sense if you did not pass an outline already. The current implementation for outline computation is rather slow, so setting this to TRUE will considerably increase computation time.

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other coloredmesh functions: `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.nat`
`coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

`coloredmesh.from.label`

Create a coloredmesh from a label.

Description

Create a coloredmesh from a label.

Usage

```
coloredmesh.from.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  surface = "white",
  makecmap_options = list(colFn = squash::rainbow2),
  binary = TRUE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`label`	string or vector of integers. If a string, the name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'. Alternatively, the already loaded label data as a vector of integers.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.
`binary`	logical, whether to treat the label as binary

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.nat` `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

`coloredmesh.from.mask` *Create a coloredmesh from a mask.*

Description

Create a coloredmesh from a mask.

Usage

```
coloredmesh.from.mask(
  subjects_dir,
  subject_id,
  mask,
  hemi,
  surface = "white",
  surface_data = NULL,
  makecmap_options = list(colFn = squash::rainbow2)
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`mask`	logical vector, contains one logical value per vertex.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`surface_data`	optional surface mesh object, as returned by `subject.surface`. If given, used instead of loading the surface data from disk (which users of this function may already have done). Defaults to NULL.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other mask functions: `mask.from.labeldata.for.hemi()`, `vis.mask.on.subject()`

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.morph.na` `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

`coloredmesh.from.morph.native` *Create a coloredmesh from native space morphometry data.*

Description

Create a coloredmesh from native space morphometry data.

Usage

```
coloredmesh.from.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  surface = "white",
  clip = NULL,
  cortex_only = FALSE,
  makecmap_options = mkco.seq()
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`measure`	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render the surface in white, without any data. One can also pass the pre-loaded morphometry data as a numerical vector, the length of which must match the number of surface vertices.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or ‘fs.surface‘ instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`clip`	numeric vector of length 2 or NULL. If given, the 2 values are interpreted as lower and upper percentiles, and the morph data is clipped at the given lower and upper percentile (see `clip.data`). Defaults to NULL (no data clipping).
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file ‘label/?h.cortex.label‘) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

`coloredmesh.from.morph.standard`

Create a coloredmesh from standard space morphometry data.

Description

Create a coloredmesh from standard space morphometry data.

Usage

```
coloredmesh.from.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm,
  surface = "white",
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  clip = NULL,
  cortex_only = FALSE,
  makecmap_options = mkco.seq()
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`measure`	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render the surface in white, without any data. One can also pass the pre-loaded morphometry data as a numerical vector, the length of which must match the number of surface vertices.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`fwhm`	string, smoothing setting. The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.

`surface`	character string or ‘fs.surface‘ instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`template_subject`	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to ’fsaverage’.
`template_subjects_dir`	The template subjects dir. If ‘NULL‘, the value of the parameter ’subjects_dir’ is used. Defaults to NULL. If you have FreeSurfer installed and configured, and are using the standard fsaverage subject, try passing the result of calling ’file.path(Sys.getenv(’FREESURFER_HOME’), ’subjects’)’.
`clip`	numeric vector of length 2 or NULL. If given, the 2 values are interpreted as lower and upper percentiles, and the morph data is clipped at the given lower and upper percentile (see `clip.data`). Defaults to NULL (no data clipping).
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file ‘label/?h.cortex.label‘) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from ’measure’.

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of ’lh’ or ’rh’.

See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.native()`, `coloredmesh.from.morphdata()`, `coloredmeshes.from.color()`

`coloredmesh.from.morphdata`

Create a coloredmesh from arbitrary data.

Description

Create a coloredmesh from arbitrary data.

Usage

```
coloredmesh.from.morphdata(
  subjects_dir,
  vis_subject_id,
  morph_data,
  hemi,
  surface = "white",
  makecmap_options = mkco.seq()
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`morph_data`	string. The morphometry data to use. E.g., 'area' or 'thickness.'
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or ‘fs.surface‘ instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.

Value

coloredmesh. A named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.native()`, `coloredmesh.from.morph.standard()`, `coloredmeshes.from.color()`

`coloredmesh.from.preloaded.data`

Generate coloredmesh from loaded data.

Description

Generate coloredmesh from loaded data.

Usage

```
coloredmesh.from.preloaded.data(
  fs_surface,
  morph_data = NULL,
  col = NULL,
  hemi = "lh",
  makecmap_options = mkco.seq()
)
```

Arguments

`fs_surface`	an fs.surface instance or a character string, which will be interpreted as the path to a file and loaded with `freesurferformats::read.fs.surface`.
`morph_data`	numerical vector, per-vertex data (typically morphometry) for the mesh. If given, takes precedence over 'col' parameter.
`col`	vector of colors, typically hex color strings like '#FF00FF'. The per-vertex-colors for the mesh. Alternative to morph_data.
`hemi`	character string, one of 'lh' or 'rh'. Metadata, the hemisphere. May be used by visualization functions to decide whether to draw the mesh in certain views.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'.

Value

as fs.coloredmesh instance

`coloredmesh.plot.colorbar.separate`

Draw colorbar for coloredmeshes in separate 2D plot.

Description

Draw a colorbar for the coloredmeshes to a separate 2D plot. Due to the suboptimal handling of colorbar drawing in the three-dimensional multi-panel views, it is often desirable to plot the colorbar in a separate window, export it from there and then manually add it to the final plot version in some image manipulation software like Inkscape. If you need more control over the colormap than offered by this function (e.g., setting the color value for NA values or making a symmetric colormap to ensure that the zero point for divergent colormaps is a neutral color), you should write custom code, and the return value from this function will come in handy to do that.

Usage

```
coloredmesh.plot.colorbar.separate(
  coloredmeshes,
  show = FALSE,
 image.plot_extra_options = list(horizontal = FALSE, legend.cex = 1.8, legend.width = 2,
    legend.mar = 12, axis.args = list(cex.axis = 5)),
 png_options = list(filename = "fsbrain_cbar.png", width = 1400, height = 1400, bg =
    "#FFFFFF00"),
  silent = FALSE,
  trim_png = TRUE,
  log_breaks = FALSE
)
```

Arguments

`coloredmeshes`	list of coloredmeshes. A coloredmesh is a named list as returned by the ‘coloredmesh.from‘ functions. It has the entries ’mesh’ of type tmesh3d, a ’col’, which is a color specification for such a mesh. The ‘vis*‘ functions (like `vis.subject.morph.native`) all return a list of coloredmeshes.
`show`	logical, Whether to open the resulting plot. Defaults to ‘TRUE‘.
`image.plot_extra_options`	named list of extra optins to pass to `image.plot`. This can be used to add a legend to the colorbar, rotate the colorbar, or whatever. The options "legend_only", "zlim", and "col" are computed and set for you by this function, so there is no need to pass these. Your list will be merged with the internal options, so you could overwrite named arguments if needed.
`png_options`	Options to pass to `png`, see the docs of that function for details. Allow you to save the plot as a png bitmap image. Example: `png_options = list("filename"="fsbrain_cbar.png"` `"width"=800)`. Defaults to NULL, which will not save anything.
`silent`	logical, whether to suppress messages. Defaults to ‘FALSE‘.
`trim_png`	logical, whether to trim the output PNG image using image magick, i.e., remove everything but the foreground. Ignored unless an output PNG image is actually written (see ’png_options’) and the ’magick’ package is installed.
`log_breaks`	logical, scalar int, or vector of ints. Whether to use log10 scale for plotting the cbar. If logical and TRUE, uses log scale with default number (=5) ticks auto-computed from the data. If a single integer N, uses N ticks auto-computed from the data instead. If a numeric vector, uses the supplied values in the vector as ticks, note that they must be on a ‘log(data)‘ scale. If the ’makecmap_options’ stored in the passed ’coloredmeshes’ contain a ’base’ value of 10, log 10 is assumed (with the default 5 ticks), even if this parameter is left at its default value, logical FALSE.

Value

named list, entries: ’output_img_path’: character string, the path to the output file, or NULL.

Note

If you increase the output resolution of the colorbar (using 'png_options'), you will have to increase the font sizes as well (using 'image.plot_extra_options'), otherwise the axis and legend labels will be hard to read.

See Also

Other colorbar functions: `combine.colorbar.with.brainview.animation()`, `combine.colorbar.with.brainview.ima`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   coloredmeshes = vis.subject.morph.native(subjects_dir, 'subject1',
    'thickness', 'lh', views=c('t4'));
   coloredmesh.plot.colorbar.separate(coloredmeshes);

   # Or plot a colorbar with a label:
   coloredmesh.plot.colorbar.separate(coloredmeshes,
    image.plot_extra_options = list("legend.lab"="Thickness [mm]",
    horizontal=TRUE, legend.cex=1.5, legend.line=-3));

## End(Not run)
```

`coloredmeshes.from.color`

Create coloredmeshes for both hemis using pre-defined colors.

Description

Create coloredmeshes for both hemis using pre-defined colors.

Usage

```
coloredmeshes.from.color(
  subjects_dir,
  subject_id,
  color_data,
  hemi,
  surface = "white",
  metadata = list()
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`color_data`	a hemilist containing vectors of hex color strings
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	character string or 'fs.surface' instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`metadata`	a named list, can contain whatever you want. Typical entries are: 'src_data' a hemilist containing the source data from which the 'color_data' was created, optional. If available, it is encoded into the coloredmesh and can be used later to plot a colorbar. 'makecmap_options': the options used to created the colormap from the data.

Value

named list of coloredmeshes. Each entry is a named list with entries: "mesh" the `tmesh3d` mesh object. "col": the mesh colors. "render", logical, whether to render the mesh. "hemi": the hemisphere, one of 'lh' or 'rh'.

See Also

Other coloredmesh functions: `coloredmesh.from.annot()`, `coloredmesh.from.label()`, `coloredmesh.from.mask()`, `coloredmesh.from.morph.native()`, `coloredmesh.from.morph.standard()`, `coloredmesh.from.morphdata()`

`colorlist.brain.clusters`

Return diverging color list

Description

Return diverging color list

Usage

```
colorlist.brain.clusters(num_colors)
```

Arguments

`num_colors`	integer, the number of colors you want

Value

vector of colors

`colors.are.grayscale` *Check for the given color strings whether they represent gray scale colors.*

Description

Check for the given color strings whether they represent gray scale colors.

Usage

```
colors.are.grayscale(col_strings, accept_col_names = TRUE)
```

Arguments

`col_strings`	vector of RGB(A) color strings, like `c("#FFFFFF", ("#FF00FF"))`.
`accept_col_names`	logical, whether to accept color names like 'white'. Disables all sanity checks.

Value

logical vector

Examples

```
colors.are.grayscale(c("#FFFFFF", "#FF00FF"));
all((colors.are.grayscale(c("#FFFFFF00", "#ABABABAB"))));
```

`colors.have.transparency`
Check for the given color strings whether they have transparency, i.e., an alpha channel value != fully opaque.

Description

Check for the given color strings whether they have transparency, i.e., an alpha channel value != fully opaque.

Usage

```
colors.have.transparency(col_strings, accept_col_names = TRUE)
```

Arguments

`col_strings`	vector of RGB(A) color strings, like `c("#FFFFFF", ("#FF00FF"))`.
`accept_col_names`	logical, whether to accept color names like 'white'. Disables all sanity checks.

Value

logical vector

Examples

```
colors.have.transparency(c("#FFFFFF", "#FF00FF", "#FF00FF00", "red", "#FF00FFDD"));
all((colors.have.transparency(c("#FFFFFF00", "#ABABABAB"))));
```

`combine.colorbar.with.brainview.animation` *Combine a colorbar and a brain animation in gif format into a new animation.*

Description

Combine a colorbar and a brain animation in gif format into a new animation.

Usage

```
combine.colorbar.with.brainview.animation(
  brain_animation,
  colorbar_img,
  output_animation,
  offset = "+0+0",
  extend_brainview_img_height_by = 0L,
  silent = FALSE,
  allow_colorbar_shrink = TRUE,
  background_color = "white"
)
```

Arguments

`brain_animation`	path to the brain animation in GIF format
`colorbar_img`	path to the main image containing the separate colorbar, usually an image in PNG format
`output_animation`	path to output image in gif format, must include the '.gif' file extension
`offset`	offset string passed to `magick::image_composite`. Allows you to shift the location of the colorbar in the final image.
`extend_brainview_img_height_by`	integer value in pixels, the size of the lower border to add to the brainview_img. Use this if the lower part of the colorbar is off the image canvas.
`silent`	logical, whether to silence all messages

`allow_colorbar_shrink`	logical, whether to shrink the colorbar to the width of the animation in case it is considerably wider (more than 20 percent). Defaults to TRUE.
`background_color`	color string, the background color to use. Use 'transparency_color' if you want a transparent background.

See Also

Other colorbar functions: `coloredmesh.plot.colorbar.separate()`, `combine.colorbar.with.brainview.image()`

`combine.colorbar.with.brainview.image`

Combine a colorbar and a brainview image into a new figure.

Description

Combine a colorbar and a brainview image into a new figure.

Usage

```
combine.colorbar.with.brainview.image(
  brainview_img = "fsbrain_arranged.png",
  colorbar_img = "fsbrain_cbar.png",
  output_img = "fsbrain_merged.png",
  offset = "+0+0",
  extend_brainview_img_height_by = NULL,
  silent = FALSE,
  allow_colorbar_shrink = TRUE,
  horizontal = FALSE,
  background_color = "#FFFFFF",
  transparency_color = NULL,
  delete_colorbar_img = TRUE
)
```

Arguments

`brainview_img`	path to the main image containing the view of the brain, usually an image in PNG format.
`colorbar_img`	path to the main image containing the separate colorbar, usually an image in PNG format.
`output_img`	path to output image, including the file extension.
`offset`	offset string passed to `magick::image_composite`. Allows you to shift the location of the colorbar in the final image.
`extend_brainview_img_height_by`	integer value in pixels, the size of the lower border to add to the brainview_img. Increase this if the lower part of the colorbar is off the image canvas.

`silent`	logical, whether to silence all messages
`allow_colorbar_shrink`	logical, whether to shrink the colorbar to the width of the animation in case it is considerably wider (more than 20 percent). Defaults to TRUE.
`horizontal`	logical, whether the colorbar is horizontal. If so, it will be added below the 'brainview_img'. If it is vertical, it will be added to the right of the 'brainview_img'.
`background_color`	color string, the background color to use. Use 'transparency_color' if you want a transparent background.
`transparency_color`	the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try 'white' or 'black' if in doubt.
`delete_colorbar_img`	logical, whether to delete the colorbar_img after the combine operation.

Value

named list with entries 'output_img_path': character string, path to saved image. 'merged_img': magick image instance, the merged image

See Also

Other colorbar functions: `coloredmesh.plot.colorbar.separate()`, `combine.colorbar.with.brainview.animation(`

`constant.pervertexdata`

Get vertex data for a single fs.surface or a hemilist of surfaces.

Description

Get vertex data for a single fs.surface or a hemilist of surfaces.

Usage

```
constant.pervertexdata(surfaces, value = NA)
```

Arguments

`surfaces`	an fs.surface instance or a `hemilist` of the latter
`value`	the morphometry data value you want.

Value

a vector or hemilist of vectors of values

`cube3D.tris` *Return triangles for a 3D cube or cuboid.*

Description

Each row of the returned matrix encodes a point (the x, y, and z coordinates), and 3 consecutive rows encode a triangle. Obvisouly, a point will occur several times (as part of several triangles). The result can be passed to `triangles3d` to render a 3D box. The defaults for the parameters will create a cube with edge length 1 centered at (0, 0, 0).

Usage

```
cube3D.tris(
  xmin = -0.5,
  xmax = 0.5,
  ymin = -0.5,
  ymax = 0.5,
  zmin = -0.5,
  zmax = 0.5,
  center = NULL,
  edge_length = 1
)
```

Arguments

`xmin`	numeric, minimal x coordinate
`xmax`	numeric, maximal x coordinate
`ymin`	numeric, minimal y coordinate
`ymax`	numeric, maximal y coordinate
`zmin`	numeric, minimal z coordinate
`zmax`	numeric, maximal z coordinate
`center`	numeric vector of length 3 or NULL, coordinates where to center a cube with the edge length defined in parameter ‘edge_length‘. If this is not ‘NULL‘, the parameters ‘xmin‘, ‘xmax‘, ... will be ignored, and their values will be computed for a cube based on the ‘center‘ and ‘edge_length‘. Note that you can only create cubes using ‘center‘ and ‘edge_length‘, while the min/max methods allows the construction of cuboids.
`edge_length`	numeric, the edge length of the cube. Only used if parameter ‘center‘ is used, ignored otherwise.

Value

numerical matrix with 36 rows and 3 columns, the 3D coordinates. Each row encodes a point (the x, y, and z coordinates), and 3 consecutive rows encode a triangle.

Examples

```
# Create a cube with edge length 2, centered at (3,4,5):
cube_coords = cube3D.tris(center=c(3,4,5), edge_length=2.0);
# Create the same cube using the min/max method:
cube_coords = cube3D.tris(xmin=2, xmax=4, ymin=3, ymax=5, zmin=4, zmax=6);
# Create a cuboid:
cuboid_coords = cube3D.tris(xmin=2, xmax=4, ymin=3, ymax=9, zmin=4, zmax=5);
# To render the cuboid:
#rgl::triangles3d(cuboid_coords, col="red");
```

`cubes3D.tris` *Vectorized version of cube3D.tris*

Description

Vectorized version of cube3D.tris

Usage

```
cubes3D.tris(centers, edge_length = 1)
```

Arguments

`centers`	numerical matrix with 3 columns. Each column represents the x, y, z coordinates of a center at which to create a cube.
`edge_length`	numerical vector or scalar, the edge length. Must have length 1 (same edge length for all cubes), or the length must be identical to the number of rows in parameter ‘centers‘.

Value

matrix of triangle coordinates, see `cube3D.tris`.

Examples

```
# Plot a 3D cloud of 20000 voxels:
centers = matrix(rnorm(20000*3)*100, ncol=3);
rgl::triangles3d(cubes3D.tris(centers));
```

`delete_all_optional_data` *Delete all data in the package cache.*

Description

Delete all data in the package cache.

Usage

```
delete_all_optional_data()
```

Value

integer. The return value of the unlink() call: 0 for success, 1 for failure. See the unlink() documentation for details.

`demographics.to.fsgd.file` *Write FreeSurfer Group Descriptor (FSGD) file from demographics dataframe.*

Description

Write FreeSurfer Group Descriptor (FSGD) file from demographics dataframe.

Usage

```
demographics.to.fsgd.file(
  filepath,
  demographics_df,
  group_column_name = "group",
  subject_id_column_name = "id",
  var_columns = NULL,
  ftitle = "OSGM",
  fsgd_flag_lines = c("DeMeanFlag 1", "ReScaleFlag 1")
)
```

Arguments

`filepath`	character string, the path to the output file in FSGD format
`demographics_df`	data.frame, as returned by `read.md.demographics` or created manually. Note that the data.frame must not contain any character columns, they should be converted to factors.

`group_column_name`	character string, the column name of the group column in the 'demographics_df'
`subject_id_column_name`	character string, the column name of the subject identifier column in the 'demographics_df'
`var_columns`	vector of character strings, the column names to include as variables in the FSGD file. If NULL (the default), all columns will be included (with the exception of the group column and the subject id column).
`ftitle`	character string, freeform title for the FSGD file
`fsgd_flag_lines`	vector of character strings, extra flag lines to write to the file. The default setting will activate de-meaning and rescaling.

Value

vector of character strings, the lines written to the 'filepath', invisible.

See Also

Other metadata functions: `read.md.demographics()`, `read.md.subjects()`, `report.on.demographics()`

`demographics.to.qdec.table.dat`

Convert a dataframe containing demographics data to a qdec.table.dat file and related files.

Description

This creates the ‘qdec.table.dat‘ and all required related files (the factor level files) in a directory.

Usage

```
demographics.to.qdec.table.dat(
  df,
  output_path = ".",
  long = FALSE,
  add_fake_level2 = FALSE,
  long_timecolumn = "years",
  qdec_file_name = "qdec.table.dat"
)
```

Arguments

`df`	a data.frame containing demographics information. Make sure to have factors encoded as factors (not strings), so that the QDEC level files get created for them. Must contain a column named 'fsid' with the subject IDs as first column. If you want a long table, make sure to use `qdec.table.skeleton` to generate the timepoint information instead of doing it manually.
`output_path`	character string, existing directory into which to write the QDEC files. If the last directory level does not exist, it will be created.
`long`	logical, whether this is for a longitudinal run. If so, the df must contain a column named 'fsid-base' as the second column. It must also contain some column that gives the inter-scan time (from this scan timepoint to the previous one). The time unit (years, days, ...) is up to you, but typically one is interested in yearly change, the unit should be years. The name of the column (e.g., 'years') must be given to 'mris_slopes' later on the command line with the `--time <column_name>` argument. The requires information can be generated conveniently with the `qdec.table.skeleton` function.
`add_fake_level2`	logical, whether to add a 2nd fake level to the level files of factors with only a single level. Such factors make little sense, but QDEC refuses to open the resulting files at all in such a case, which seems a bit overkill. If TRUE, a 2nd level named 'level2' will be added so that one can open the output in QDEC.
`long_timecolumn`	character string, the name of the column holding the inter-scan time. Ignored unless parameter `long` is `TRUE`. See the description for parameter `long` for details.
`qdec_file_name`	character string, the filename of the QDEC file to write. Must be only the file name (with extension if you want). See `output_path` to set the ouput directory where this will be created.

Note

IMPORTANT: If you import the dataframe from a text file with functions like `read.table`, they will by default replace dashes in column names with dots. So if you have a column named `fsid-base` in there, after loading it will be named `fsid.base`. See the `check.names` parameter for `read.table` to prevent that.

See Also

The function `qdec.table.skeleton` to generate the data.frame used as the 'df' argument for this function.

Examples

```
## Not run:
   dem = readxl::read_xls("~/data/study1/demographics.xsl");
   # or: dem = read.table("~/demographics.csv", check.names=FALSE);
   # You may want to rearrange/rename/delete some columns here.
   demographics.to.qdec.table.dat(dem, "~/data/study1/qdec/");
```

```
    #
    # a second one with real data:
    dem = data.frame("ID"=paste("subject", seq(5), sep=""),
       "age"=sample.int(20, 5)+10L, "isi"=rnorm(5, 2.0, 0.1)); #sample data.
    long_table = qdec.table.skeleton(dem$ID, dem$isi);
    demographics.to.qdec.table.dat(long_table, long=TRUE);

## End(Not run)
```

`desaturate` *Perform simple desaturation or grayscale conversion of RGBA colors.*

Description

Perform simple desaturation or grayscale conversion of RGBA colors.

Usage

```
desaturate(color, gamma_correct = FALSE)
```

Arguments

`color`	rgba color strings
`gamma_correct`	logical, whether to apply non-linear gamma correction. First performs gamma expansion, then applies the gray-scale channel weigths, then gamma compression.

Value

rgba color strings, the grayscale colors. The information from one of the three rgb channels would be enough. The alpha value is not touched.

Note

Assumes sRGB color space.

References

see https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale

See Also

Other color functions: `alphablend()`

`download_fsaverage` *Download the FreeSurfer v6 fsaverage subject.*

Description

Download some relevant files from the FreeSurfer v6 fsaverage subject. The files are subject to the FreeSurfer software license, see parameter 'accept_freesurfer_license' for details. This data is not required for the package to work. If you are working on a machine that has FreeSurfer installed, you already have this data anyways and do not need to download it. If not, it is very convenient to have it if you are using the fsaverage template subject to analyze your standard space data, as it is required for visualization of such data.

Usage

```
download_fsaverage(accept_freesurfer_license = FALSE)
```

Arguments

`accept_freesurfer_license`	logical, whether you accept the FreeSurfer license for fsaverage, available at https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense. Defaults to FALSE.

Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using get_optional_data_file(). "missing": vector of strings. The names of the files that this function was unable to retrieve.

`download_fsaverage3` *Download the FreeSurfer v6 low-resolution fsaverage3 subject.*

Description

Download some relevant files from the FreeSurfer v6 fsaverage3 subject. The files are subject to the FreeSurfer software license, see parameter 'accept_freesurfer_license' for details. This data is not required for the package to work. If you are working on a machine that has FreeSurfer installed, you already have this data anyways and do not need to download it. Also downloads data for subject1 that has been mapped to fsaverage.

Usage

```
download_fsaverage3(accept_freesurfer_license = FALSE)
```

Arguments

`accept_freesurfer_license`	logical, whether you accept the FreeSurfer license for fsaverage, available at https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense. Defaults to FALSE.

Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using get_optional_data_file(). "missing": vector of strings. The names of the files that this function was unable to retrieve.

Note

The subject fsaverage3 is a downsampled (low mesh resolution) version of the standard fsaverage. If you never heard about fsaverage3, you do not need it. You will have to manually re-sample your data in FreeSurfer if you want to use it with fsaverage3.

`download_optional_data` *Download optional data for this package if required.*

Description

Ensure that the optioanl data is available locally in the package cache. Will try to download the data only if it is not available. This data is not required for the package to work, but it is used in the examples, in the unit tests and also in the example code from the vignette. Downloading it is highly recommended.

Usage

```
download_optional_data()
```

Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using get_optional_data_file(). "missing": vector of strings. The names of the files that this function was unable to retrieve.

`download_optional_paper_data`
Download extra data to reproduce the figures from the fsbrain paper.

Description

Download extra data to reproduce the figures from the fsbrain paper.

Usage

```
download_optional_paper_data()
```

Value

Named list. The list has entries: "available": vector of strings. The names of the files that are available in the local file cache. You can access them using get_optional_data_file(). "missing": vector of strings. The names of the files that this function was unable to retrieve.

Note

Called for side effect of data download.

`export` *Export high-quality brainview image with a colorbar.*

Description

This function serves as an easy (but slightly inflexible) way to export a high-quality, tight-layout, colorbar figure to disk. If no colorbar is required, one can use `vislayout.from.coloredmeshes` instead. It is an alias for ‘vis.export.from.coloredmeshes‘ that requires less typing.

Usage

```
export(
  coloredmeshes,
  colorbar_legend = NULL,
  img_only = TRUE,
  draw_colorbar = "horizontal",
  horizontal = NULL,
  silent = TRUE,
  quality = 1L,
  output_img = "fsbrain_arranged.png",
  image.plot_extra_options = NULL,
  large_legend = TRUE,
  view_angles = get.view.angle.names(angle_set = "t4"),
```

```
  style = "default",
  grid_like = TRUE,
  background_color = "white",
  transparency_color = NULL,
  ...
)
```

Arguments

`coloredmeshes`	list of coloredmesh. A coloredmesh is a named list as returned by the ‘coloredmesh.from*‘ functions (like `coloredmesh.from.morph.native`). It has the entries ’mesh’ of type tmesh3d, a ’col’, which is a color specification for such a mesh. The ‘vis*‘ functions (like `vis.subject.morph.native`) all return a list of coloredmeshes.
`colorbar_legend`	character string or NULL, the title for the colorbar.
`img_only`	logical, whether to return only the resulting image
`draw_colorbar`	logical or one of the character strings ’vertical’ or ’horizontal’, whether to draw a colorbar. Defaults to ’horizontal’.
`horizontal`	deprecated (since 0.5.0) and ignored, use parameter ’draw_colorbar’ instead.
`silent`	logical, whether to suppress messages
`quality`	integer, an arbitrary quality. This is the resolution per tile before trimming, divided by 1000, in pixels. Example: 1L means 1000x1000 pixels per tile before trimming. Currently supported values: `1L..2L`. Note that the resolution you can get is also limited by your screen resolution.
`output_img`	string, path to the output file. Defaults to "fsbrain_arranged.png"
`image.plot_extra_options`	named list, custom options for fields::image.plot. Overwrites those derived from the quality setting. If in doubt, leave this alone.
`large_legend`	logical, whether to plot extra large legend text, affects the font size of the colorbar_legend and the tick labels.
`view_angles`	list of strings. See `get.view.angle.names` for all valid strings.
`style`	the rendering style, see `material3d` or use a predefined style like ’default’ or ’shiny’.
`grid_like`	logical, passed to `vislayout.from.coloredmeshes`.
`background_color`	hex color string (like ’#FFFFFF’), the color to use for the background. Ignored if ’transparency_color’ is not NULL. To get a transparent background, use ’transparency_color’ instead of this parameter. WARNING: Do not use color names (like ’gray’), as their interpretation differs between rgl and image magick!
`transparency_color`	hex color string (like ’#FFFFFF’), the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try ’#FFFFFF’ (white) or ’#000000’ (black) if in doubt. WARNING: Do not use color names (like ’gray’), as their interpretation differs between rgl and image magick!
`...`	extra arguments passed to `vislayout.from.coloredmeshes`.

Value

magick image instance or named list, depending on the value of 'img_only'. If the latter, the list contains the fields 'rev_vl', 'rev_cb', and 'rev_ex', which are the return values of the functions `vislayout.from.coloredmeshes`, `coloredmesh.plot.colorbar.separate`, and `combine.colorbar.with.brainview.` respectively.

Note

Note that your screen resolution has to be high enough to generate the final image in the requested resolution, see the 'fsbrain FAQ' vignette for details and solutions if you run into trouble.

Examples

```
## Not run:
    rand_data = rnorm(327684, 5, 1.5);
    cm = vis.data.on.fsaverage(morph_data_both=rand_data,
      rglactions=list('no_vis'=T));
    export(cm, colorbar_legend='Random data',
      output_img="~/fsbrain_arranged.png");

## End(Not run)
```

`export.coloredmesh.ply`

Export a coloredmeshes with vertexcolors in PLY format.

Description

Exports coloredmeshes with vertex coloring to standard mesh files in Stanford Triangle (PLY) format. This is very hand for rendering in external standard 3D modeling software like Blender.

Usage

```
export.coloredmesh.ply(filepath, coloredmesh)
```

Arguments

`filepath`	The export filepath, including file name and extension.
`coloredmesh`	an 'fs.coloredmesh' instance, as returned (silently) by all surface visualization functions, like `vis.subject.morph.native`.

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   coloredmeshes = vis.subject.morph.native(subjects_dir, 'subject1', 'thickness');
   export.coloredmesh.ply('~/subject1_thickness_lh.ply', coloredmeshed$lh);

## End(Not run)
```

`face.edges` *Enumerate all edges of the given faces or mesh.*

Description

Compute edges of a tri-mesh. Can compute all edges, or only a subset, given by the face indices in the mesh.

Usage

```
face.edges(surface_mesh, face_indices = "all")
```

Arguments

`surface_mesh`	surface mesh, as loaded by `subject.surface` or `read.fs.surface`.
`face_indices`	integer vector, the face indices. Can also be the character string 'all' to use all faces.

Value

integer matrix of size (n, 2) where n is the number of edges. The indices (source and target vertex) in each row are **not** sorted, and the edges are **not** unique. I.e., each undirected edge ‘u, v‘ (with the exception of edges on the mesh border) will occur twice in the result: once as ‘u, v‘ and once as ‘v, u‘.

See Also

Other surface mesh functions: `label.border`(), `mesh.vertex.included.faces`(), `mesh.vertex.neighbors`(), `subject.surface`(), `vis.path.along.verts`()

`find.freesurferhome` *Find the FREESURFER_HOME directory on disk.*

Description

Try to find directory containing the FreeSurfer installation, based on environment variables and *educated guessing*.

Usage

```
find.freesurferhome(mustWork = FALSE)
```

Arguments

`mustWork`	logical. Whether the function should with an error stop if the directory cannot be found. If this is TRUE, the return value will be only the 'found_at' entry of the list (i.e., only the path of the FreeSurfer installation dir).

Value

named list with the following entries: "found": logical, whether it was found. "found_at": Only set if found=TRUE, the path to the FreeSurfer installation directory (including the directory itself). See 'mustWork' for important information.

See Also

`fs.home`

`find.subjectsdir.of` *Find the subject directory containing the fsaverage subject (or others) on disk.*

Description

Try to find directory containing the fsaverage subject (or any other subject) by checking in the following places and returning the first path where it is found: first, the directory given by the environment variable SUBJECTS_DIR, then in the subir 'subjects' of the directory given by the environment variable FREESURFER_HOME, and finally the base dir of the package cache. See the function `download_fsaverage` if you want to download fsaverage to your package cache and ensure it always gets found, no matter whether the environment variables are set or not.

Usage

```
find.subjectsdir.of(subject_id = "fsaverage", mustWork = FALSE)
```

Arguments

`subject_id`	string, the subject id of the subject. Defaults to 'fsaverage'.
`mustWork`	logical. Whether the function should with an error stop if the directory cannot be found. If this is TRUE, the return value will be only the 'found_at' entry of the list (i.e., only the path of the subjects dir).

Value

named list with the following entries: "found": logical, whether it was found. "found_at": Only set if found=TRUE, the path to the fsaverage directory (NOT including the fsaverage dir itself). "found_all_locations": list of all locations in which it was found. See 'mustWork' for important information.

See Also

`fsaverage.path`

`fs.coloredmesh` *fs.coloredmesh constructor*

Description

fs.coloredmesh constructor

Usage

```
fs.coloredmesh(
  mesh,
  col,
  hemi,
  render = TRUE,
  metadata = NULL,
  add_normals = FALSE
)
```

Arguments

`mesh`	a 'mesh3d' instance as returned by `tmesh3d` or an 'fs.surface' brain surface mesh as returned by functions like `subject.surface`.
`col`	vector of vertex colors for the mesh, one color per vertex. Expanded if exactly one color.
`hemi`	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.
`render`	logical, whether to render this mesh during visualization
`metadata`	optional, named list containing metadata
`add_normals`	logical, whether to compute normals and save them in the mesh.

Value

an ‘fs.coloredmesh‘ instance. The only fields one should use in client code are ’mesh’, ’hemi’ and ’col’, all others are considered internal and may change without notice.

`fs.home` *Return FreeSurfer path.*

Description

Return FreeSurfer path.

Usage

```
fs.home()
```

Value

the FreeSurfer path, typically what the environment variable ‘FREESURFER_HOME‘ points to.

Note

This function will stop (i.e., raise an error) if the directory cannot be found.

`fs.surface.as.adjacencylist` *Turn surface mesh into a igraph and return its adjacency list representation.*

Description

Turn surface mesh into a igraph and return its adjacency list representation.

Usage

```
fs.surface.as.adjacencylist(surface)
```

Arguments

`surface`	an fs.surface instance as returned by `subject.surface`, an existing igraph (which will be returned as-is) or a string which is interpreted as a path to a surface file.

Value

list of integer vectors, the adjacency list.

`fs.surface.to.igraph` *Create igraph undirected graph from a brain surface mesh.*

Description

Create igraph undirected graph from a brain surface mesh.

Usage

```
fs.surface.to.igraph(surface)
```

Arguments

`surface`	an fs.surface instance as returned by `subject.surface`, an existing igraph (which will be returned as-is) or a string which is interpreted as a path to a surface file.

Value

igraph::graph instance

Examples

```
## Not run:
  # Find the one-ring neighbors of vertex 15 on the fsaverage left hemi:
  sf = subject.surface(fsaverage.path(T), "fsaverage", "white", "lh");
  g = fs.surface.to.igraph(sf);
  igraph::neighborhood(g, order = 1, nodes = 15);

## End(Not run)
```

`fs.surface.to.tmesh3d` *Get an rgl tmesh3d instance from a brain surface mesh.*

Description

Get an rgl tmesh3d instance from a brain surface mesh.

Usage

```
fs.surface.to.tmesh3d(surface)
```

Arguments

`surface`	an fs.surface instance, as returned by `subject.surface` or `freesurferformats::read.fs.surface`.

Value

a tmesh3d instance, see `rgl::tmesh3d` for details.

`fs.surface.vertex.neighbors`

Compute vertex neighborhoods or the full adjacency list for a mesh using the Rvcg or igraph library.

Description

This is a faster replacement for `mesh.vertex.neighbors` that requires the optional dependency package 'igraph' or 'Rvcg'.

Usage

```
fs.surface.vertex.neighbors(
  surface,
  nodes = NULL,
  order = 1L,
  simplify = TRUE,
  include_self = FALSE
)
```

Arguments

`surface`	an fs.surface instance as returned by `subject.surface`, an existing igraph (which will be returned as-is) or a string which is interpreted as a path to a surface file.
`nodes`	the source vertex. Passed on to `igraph::neighborhood`. Can be a vector, in which case the neighborhoods for all these vertices are computed separately. If NULL, all graph vertices are used.
`order`	integer, the max graph distance of vertices to consider neighbors (number of neighborhood rings). Passed on to `igraph::neighborhood`
`simplify`	logical, whether to return only an integer vector if the 'nodes' parameter has length 1 (instead of a list where the first element is such a vector).
`include_self`	logical, whether to include vertices in their own neighborhood

Value

named list of integer vectors (see `igraph::neighborhood`), unless 'simplify' is TRUE, see there for details.

Note

If you intend to call several functions on the igraph, it is faster to construct it with `fs.surface.to.igraph` and keep it.

See Also

The `fs.surface.as.adjacencylist` function computes the 1-ring neighborhood for the whole graph.

`fsaverage.path` *Return path to fsaverage dir.*

Description

Return path to fsaverage dir.

Usage

```
fsaverage.path(allow_fetch = FALSE)
```

Arguments

`allow_fetch`	logical, whether to allow trying to download it.

Value

the path to the fsaverage directory (NOT including the 'fsaverage' dir itself).

Note

This function will stop (i.e., raise an error) if the directory cannot be found. The fsaverage template is part of FreeSurfer, and distributed under the FreeSurfer software license.

`fsbrain.set.default.figsize` *Set default figure size for fsbrain visualization functions.*

Description

Set default figure size for fsbrain visualization functions.

Usage

```
fsbrain.set.default.figsize(width, height, xstart = 50L, ystart = 50L)
```

Arguments

`width`	integer, default figure width in pixels
`height`	integer, default figure height in pixels
`xstart`	integer, default horizontal position of plot window on screen, left border is 0. The max value (right border) depends on your screen resolution.
`ystart`	integer, default vertical position of plot window on screen, upper border is 0. The max value (lower border) depends on your screen resolution.

Note

This function overwrites `options("fsbrain.rgloptions")`. Output size is limited by your screen resolution. To set your preferred figure size for future R sessions, you could call this function in your `'~/.Rprofile'` file.

`fup` *Transform first character of a string to uppercase.*

Description

Transform first character of a string to uppercase. This is useful when labeling plots. Important: this function does not know about different encodings, languages or anything, it just calls `toupper` for the first character.

Usage

```
fup(word)
```

Arguments

`word,`	string. Any string.

Value

string, the input string with the first character transformed to uppercase.

Examples

```
word_up = fup("word");
```

`gen.test.volume` *Generate test 3D volume of integers. The volume has an outer background area (intensity value 'bg') and an inner foreground areas (intensity value 200L).*

Description

Generate test 3D volume of integers. The volume has an outer background area (intensity value 'bg') and an inner foreground areas (intensity value 200L).

Usage

```
gen.test.volume(vdim = c(256L, 256L, 256L), bg = NA)
```

Arguments

`vdim`	integer vector of length 3, the dimensions
`bg`	value to use for outer background voxels. Typically ‘0L‘ or ‘NA‘.

Value

a 3d array of integers

Note

This function exists for software testing purposes only, you should not use it in client code.

`geod.patches.color.overlay`

Generate color overlay from geodesic patches around several vertices.

Description

Works across hemispheres (for a whole brain) if you pass a `hemilist` of meshes as parameter 'mesh', see below.

Usage

```
geod.patches.color.overlay(
  mesh,
  vertex,
  color = "#FF0000",
  bg_color = "#FEFEFE",
  ...
)
```

Arguments

`mesh`	a single `fs.surface` instance, or a `hemilist` of two such meshes. If a hemilist, the vertex indices can go from 1 to the sum of vertices in both meshes, and the proper hemisphere will be used automatically.
`vertex`	positive integer (or vector of the latter), the index of the source vertex in the mesh. If a vector, the neighborhoods for all vertices will be computed separately.
`color`	single color string like `'#FF0000'` or vector of such strings. If a vector, the length should match the number of vertices in parameter 'vertex'.
`bg_color`	character string, the background color.
`...`	extra arguments passed to `geod.vert.neighborhood`.

Value

vector of color strings (or a `hemilist` of 2 such vectors if 'mesh' is a hemilist), an overlay suitable for visualization using `vis.color.on.subject`.

Examples

```
## Not run:
  sjd = fsaverage.path(TRUE);
  surfaces = subject.surface(sjd, 'fsaverage', surface = "white", hemi = "both");
  colors = geod.patches.color.overlay(surfaces, vertex = c(12345L, 45L),
    color = c("#FF0000", "#00FF00"), max_distance = 45.0);
  vis.color.on.subject(sjd, 'fsaverage', color_lh=colors$lh, color_rh=colors$rh);

## End(Not run)
```

`geod.vert.neighborhood`

Compute all vertices within given geodesic distance on the mesh.

Description

Compute all vertices within given geodesic distance on the mesh.

Usage

```
geod.vert.neighborhood(
  mesh,
  vertex,
  max_distance = 5,
  include_max = TRUE,
  return_distances = TRUE
)
```

Arguments

`mesh`	an instance of `rgl::tmesh3d` or `freesurferformats::fs.surface`.
`vertex`	positive integer (or vector of the latter), the index of the source vertex in the mesh. If a vector, the neighborhoods for all vertices will be computed separately.
`max_distance`	double, the neighborhood size. All mesh vertices in geodesic distance smaller than / up to this distance will be returned.
`include_max`	logical, whether the max_distance value is inclusive.
`return_distances`	logical, whether to compute the 'distances' entry in the returned list. Doing so is a little bit slower, so it can be turned off if not needed.

Value

named list with the following entries: 'vertices': integer vector, the indices of all vertices in the neigborhood. 'distances': double vector, the distances to the respective vertices (unless 'return_distances' is FALSE).

Note

This function uses the pseudo-geodesic distance along the mesh edges.

Examples

```
## Not run:
  sjd = fsaverage.path(TRUE);
  surface = subject.surface(sjd, 'fsaverage', surface = "white", hemi = "lh");
  res = geod.vert.neighborhood(surface, 12345L, max_distance = 10.0);
  res$vertices;

## End(Not run)
```

`geodesic.circles` *Compute geodesic circles and ball stats for given vertices.*

Description

Compute geodesic circles and ball stats for given vertices.

Usage

```
geodesic.circles(surface, vertices = NULL, scale = 5)
```

Arguments

`surface`	an `rgl::tmesh3d` or `freesurferformats::fs.surface` instance. Can be a character string, which will be loaded as a surface file if it exists.
`vertices`	positive integer vector, the vertex indices for which to compute the stats. If NULL, it is computed for all vertices.
`scale`	double, surface area to be covered by patch in percent

Note

This takes a while for large meshes, try it with single vertices or with a surface like fsaverage3 if you want it for all vertices. This requires the optional dependency package 'pracma'.

Examples

```
## Not run:
  sjd = fsaverage.path(TRUE);
  surface = subject.surface(sjd, 'fsaverage3', hemi='lh');
  gc = geodesic.circles(surface);
  vis.data.on.subject(sjd, 'fsaverage3', morph_data_lh = gc$radius);
  vis.data.on.subject(sjd, 'fsaverage3', morph_data_lh = gc$perimeter);

## End(Not run)
```

`geodesic.dists.to.vertex` *Simple internal wrapper around* `Rvcg::vcgDijkstra` *with function check.*

Description

Simple internal wrapper around `Rvcg::vcgDijkstra` with function check.

Usage

```
geodesic.dists.to.vertex(mesh, v)
```

Arguments

`mesh`	a tmesh3d instance.
`v`	positive integer, a vertex index in the mesh.

Value

double vector with length equal to num vertices in the mesh, the geodesic distances from all other vertices to the query vertex `v`.

`geodesic.path` *Compute geodesic path from a source vertex to one or more target vertices.*

Description

Compute geodesic path from a source vertex to one or more target vertices.

Usage

```
geodesic.path(surface, source_vertex, target_vertices)
```

Arguments

`surface`	an `rgl::tmesh3d` or `freesurferformats::fs.surface` instance. Can be a character string, which will be loaded as a surface file if it exists.
`source_vertex`	a scalar positive integer, the source vertex index in the mesh
`target_vertices`	single integer or vector of integers, the target vertices to which to compute the paths from the source_vertex.

Value

list of integer vectors, the paths

Note

This can take a bit for very large graphs. This requires the optional dependency package 'Rvcg'. The backtracking is currently done in R, which is not optimal from a performance perspective. If you have a recent Rvcg version with the Rvcg::vcgGeodesicPath function, that one will be used instead, and the performance will be better.

Examples

```
## Not run:
  sjd = fsaverage.path(TRUE);
  surface = subject.surface(sjd, 'fsaverage3',
    surface = "white", hemi = "lh");
  p = geodesic.path(surface, 5, c(10, 20));
  vis.subject.morph.native(sjd, 'fsaverage3', 'thickness', views='si');
  vis.paths.along.verts(surface$vertices, p$paths, color=c("red", "yellow"));

## End(Not run)
```

`get.atlas.region.names`
Determine atlas region names from a subject.

Description

Determine atlas region names from a subject. WARNING: Not all subjects have all regions of an atlas. You should use an average subject like fsaverage to get all regions.

Usage

```
get.atlas.region.names(
  atlas,
  template_subjects_dir = NULL,
  template_subject = "fsaverage",
  hemi = "lh"
)
```

Arguments

`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

`template_subjects_dir`,
: string. The directory containing the dir of the template_subject. E.g., the path to FREESURFER_HOME/subjects. If NULL, env vars will be searched for candidates, and the function will fail if they are not set correctly. Defaults to NULL.

`template_subject`,
: string. The subject identifier. Defaults to 'fsaverage'.

`hemi`,
: string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. Defaults to 'lh'. Should not matter much, unless you do not have the file for one of the hemis for some reason.

Value

vector of strings, the region names.

See Also

Other atlas functions: `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
 fsbrain::download_optional_data();
 subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
 atlas_regions = get.atlas.region.names('aparc',
 template_subjects_dir=subjects_dir, template_subject='subject1');

## End(Not run)
```

---

get.rglstyle *Get the default visualization style parameters as a named list.*

Description

Run `material3d` without arguments to see valid style keywords to create new styles.

Usage

```
get.rglstyle(style)
```

Arguments

`style`	string. A style name. Available styles are one of: "default", "shiny", "semitransparent", "glass", "edges".

Value

a style, resolved to a parameter list compatible with `material3d`.

See Also

`shade3d` can use the returned style

`get.view.angle.names` *Get list of valid view angle names.*

Description

The returned strings are used as constants to identify a view of type ‘sd_<angle>‘. They can be used to construct entries for the parameter ‘views‘ of functions like `vis.subject.morph.native`, or directly as parameter ’view_angles’ for functions like `vislayout.from.coloredmeshes`.

Usage

```
get.view.angle.names(angle_set = "all", add_sd_prefix = TRUE)
```

Arguments

`angle_set`	string, which view subset to return. Available subsets are: ’all’ (or alias ’t9’): for all 9 angles. ’t4’: for the t4 views. ’medial’: the 2 medial views, one for each hemi. ’lateral’: the 2 lateral views, one for each hemi. ’lh’: medial and laterial for the left hemisphere. ’rh’: medial and laterial for the right hemisphere.
`add_sd_prefix`	logical, whether the prefix ’sd_’ should be added to the string. This will construct full view names. If set to false, only the substring after the prefix ’sd_’ will be returned. This is used internally only and should not be needed in general.

Value

vector of character strings, all valid view angle strings.

`getIn` *Retrieve values from nested named lists*

Description

Retrieve values from nested named lists

Usage

```
getIn(named_list, listkeys, default = NULL)
```

Arguments

`named_list`	a named list
`listkeys`	vector of character strings, the nested names of the lists
`default`	the default value to return in case the requested value is ‘NULL‘.

Value

the value at the path through the lists, or ‘NULL‘ (or the ’default’) if no such path exists.

Examples

```
data = list("regions"=list("frontal"=list("thickness"=2.3, "area"=2345)));
getIn(data, c("regions", "frontal", "thickness"));       # 2.3
getIn(data, c("regions", "frontal", "nosuchentry"));     # NULL
getIn(data, c("regions", "nosuchregion", "thickness"));  # NULL
getIn(data, c("regions", "nosuchregion", "thickness"), default=14);  # 14
```

`get_optional_data_filepath`

Access a single file from the package cache by its file name.

Description

Access a single file from the package cache by its file name.

Usage

```
get_optional_data_filepath(filename, mustWork = TRUE)
```

Arguments

`filename,`	string. The filename of the file in the package cache.
`mustWork,`	logical. Whether an error should be created if the file does not exist. If mustWork=FALSE and the file does not exist, the empty string is returned.

Value

string. The full path to the file in the package cache or the empty string if there is no such file available. Use this in your application code to open the file.

---

group.agg.atlas.native

Aggregate native space morphometry data over brain atlas regions and subjects for a group of subjects.

Description

Aggregate native space morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas for all subjects.

Usage

```
group.agg.atlas.native(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  agg_fun = mean,
  cache_file = NULL
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh', 'split', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used. If 'split', the data for hte two hemispheres will go into seprate columns, with column names having 'lh_' and 'rh_' prefixes.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`agg_fun,`	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`cache_file,`	string or NULL. If given, it is interpreted as path of a file, and the data will be cached in the file cache_file in RData format. If the file does not exist yet, the function will run and cache the data in the file. If the file exists, the function will load the data from the file instead of running. The filename should end in

'.RData', but that is not enforced or checked in any way. WARNING: If cached data is returned, all parameters passed to this function (with the exception of 'cache_file') are ignored! Whether the cached data is for another subjects_list or hemi is NOT checked! You have to ensure this yourself, by using different filenames. Defaults to NULL.

Value

dataframe with aggregated values for all regions and subjects, with n columns and m rows, where n is the number of subjects and m is the number of regions.

See Also

Other aggregation functions: `group.agg.atlas.standard()`, `group.morph.agg.standard.vertex()`, `subject.atlas.agg()`

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   agg = group.agg.atlas.native(subjects_dir, c('subject1', 'subject2'),
    'thickness', 'lh', 'aparc');
   # Visualize the mean values. Could use any subject, typically
   # one would use fsaverage. Here we use subject1:
   agg$subject = NULL;   # remove non-numeric column.
   vis.region.values.on.subject(subjects_dir, 'subject1', 'aparc',
    lh_region_value_list=colMeans(agg), rh_region_value_list=NULL);

## End(Not run)
```

---

group.agg.atlas.standard

Aggregate standard space morphometry data over brain atlas regions and subjects for a group of subjects.

Description

Aggregate standard space morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas for all subjects.

Usage

```
group.agg.atlas.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  cache_file = NULL
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`fwhm,`	string. The smoothing setting which was applied when mapping data to the template subject. Usually one of '0', '5', '10', '15', '20', or '25'.
`agg_fun,`	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`template_subject,`	string. The template subject name. Defaults to 'fsaverage'. Must have its data in subjects_dir.
`cache_file,`	string or NULL. If given, it is interpreted as path of a file, and the data will be cached in the file cache_file in RData format. If the file does not exist yet, the function will run and cache the data in the file. If the file exists, the function will load the data from the file instead of running. The filename should end in '.RData', but that is not enforced or checked in any way. WARNING: If cached data is returned, all parameters passed to this function (with the exception of 'cache_file') are ignored! Whether the cached data is for another subjects_list or hemi is NOT checked! You have to ensure this yourself, by using different filenames. Defaults to NULL.

Value

dataframe with aggregated values for all regions and subjects, with n columns and m rows, where n is the number of subjects and m is the number of regions.

See Also

Other aggregation functions: `group.agg.atlas.native()`, `group.morph.agg.standard.vertex()`, `subject.atlas.agg()`

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   agg = group.agg.atlas.standard(subjects_dir, c('subject1', 'subject2'),
    'thickness', 'lh', 'aparc', fwhm='10');
   # Visualize the mean values. Could use any subject, typically
   #  one would use fsaverage. Here we use subject1:
   agg$subject = NULL;   # remove non-numeric column.
   vis.region.values.on.subject(subjects_dir, 'subject1', 'aparc',
    lh_region_value_list=colMeans(agg), rh_region_value_list=NULL);

## End(Not run)
```

`group.annot` *Load annotations for a group of subjects.*

Description

Load a brain surface annotation, i.e., a cortical parcellation based on an atlas, for a group of subjects.

Usage

```
group.annot(subjects_dir, subjects_list, hemi, atlas)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	vector of strings. The subject identifiers.
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

Value

list of annotations, as returned by freesurferformats::read.fs.annot(). If hemi is 'both', the annotations are the results of merging over the hemispheres for each subject.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
   annotations = group.annot(subjects_dir, subjects_list, "lh", "aparc");

## End(Not run)
```

`group.concat.measures.native`

Concatenate native space data for a group of subjects.

Description

A measure is something like 'thickness' or 'area'. This function concatenates the native space data for all subjects into a single long vector for each measure. A dataframe is then created, in which each column is one such vector. This can be used to compute the correlation between measures on vertex level, for example.

Usage

```
group.concat.measures.native(
  subjects_dir,
  subjects_list,
  measures,
  hemi,
  cortex_only = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measures,`	vector of strings. Names of the vertex-wise morhometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded. The data of each measure will be one column in the resulting dataframe.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`cortex_only`	logical, whether to set non-cortex data to NA

Value

dataframe with concatenated vertex values. Each column contains the values for one measure, concatenated for all subjects. WARNING: This dataframe can get large if you have many subjects.

See Also

Other concatination functions: `group.concat.measures.standard()`

Examples

```
## Not run: 
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c('subject1', 'subject2');
   cm = group.concat.measures.native(subjects_dir, subjects_list,
    c("thickness", "area"), "lh");

## End(Not run)
```

`group.concat.measures.standard`

Concatenate standard space data for a group of subjects.

Description

A measure is something like 'thickness' or 'area'. This function concatenates the standard space data for all subjects into a single long vector for each measure. A dataframe is then created, in which each column is one such vector. This can be used to compute the correlation between measures on vertex level, for example.

Usage

```
group.concat.measures.standard(
  subjects_dir,
  subjects_list,
  measures,
  hemi,
  fwhm_per_measure,
  cortex_only = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measures,`	vector of strings. Names of the vertex-wise morhometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded. The data of each measure will be one column in the resulting dataframe.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm_per_measure,`	vector of strings. The fwhm settings to use, for each measure. If this is a string instead of a vector of strings, the same fwhm will be used for all measures.
`cortex_only`	logical, whether to set non-cortex data to NA

Value

dataframe with concatenated vertex values. Each column contains the values for one measure, concatenated for all subjects. The column names are a concatination of the measure, "_fwhm", and the fwhm for that measure. WARNING: This dataframe can get large if you have many subjects.

See Also

Other concatination functions: `group.concat.measures.native()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c('subject1', 'subject2');
   cm = group.concat.measures.standard(subjects_dir, subjects_list,
    c("thickness", "area"), "lh", "10");

## End(Not run)
```

`group.label` *Retrieve label data for a group of subjects.*

Description

Load a label (like 'label/lh.cortex.label') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
group.label(
  subjects_dir,
  subjects_list,
  label,
  hemi,
  return_one_based_indices = TRUE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	vector of strings. The subject identifiers.
`label,`	string. Name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`return_one_based_indices,`	logical. Whether the indices should be 1-based. Indices are stored zero-based in the file, but R uses 1-based indices. Defaults to TRUE, which means that 1 will be added to all indices read from the file before returning them.

Value

named list of integer vectors with label data: Each name is a subject identifier from subjects_list, and the values are lists of the vertex indices in the respective label. See 'return_one_based_indices' for important information.

See Also

Other label data functions: `labeldata.from.mask()`, `mask.from.labeldata.for.hemi()`, `subject.label()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
   labels = group.label(subjects_dir, subjects_list, 'cortex.label', "lh");

## End(Not run)
```

`group.label.from.annot`

Extract a region from an atlas annotation as a label for a group of subjects.

Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to NaN or to extract only values from a certain region.

Usage

```
group.label.from.annot(
  subjects_dir,
  subjects_list,
  hemi,
  atlas,
  region,
  return_one_based_indices = TRUE,
  invert = FALSE,
  error_on_invalid_region = TRUE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	vector of string. The subject identifiers.
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`region,`	string. A valid region name for the annotation, i.e., one of the regions of the atlas.

`return_one_based_indices,`	logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.
`invert,`	logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
`error_on_invalid_region,`	logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

Value

named list of integer vectors with label data: for each subject, the list of vertex indices in the label.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot(` `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

`group.morph.agg.native` *Aggregate native space morphometry data over one hemisphere for a group of subjects.*

Description

Compute the mean (or other aggregates) over all vertices of a subject from native space morphometry data (like 'surf/lh.area'). Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
group.morph.agg.native(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  agg_fun = mean,
  cast = TRUE,
  format = "curv",
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`agg_fun`	function. An R function that aggregates data, typically `max`, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`cast`	Whether a separate 'hemi' column should exist.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. Foe example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.
`agg_fun_extra_params`	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the agg_fun.

Value

dataframe with aggregated values for all subjects, with 3 columns and n rows, where n is the number of subjects. The 3 columns are 'subject_id', 'hemi', and '<measure>' (e.g., "thickness"), the latter contains the aggregated data.

See Also

Other global aggregation functions: `group.morph.agg.standard()`, `group.multimorph.agg.native()`, `group.multimorph.agg.standard()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
   fulldata = group.morph.agg.native(subjects_dir, subjects_list, "thickness", "lh");
```

```
    head(fulldata);

## End(Not run)
```

`group.morph.agg.standard`

Aggregate standard space (fsaverage) morphometry data over one hemisphere for a group of subjects.

Description

Compute the mean (or other aggregates) over all vertices of a subject from standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh'). Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
group.morph.agg.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cast = TRUE,
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm`	string. Smoothing as string, e.g. '10' or '25'.
`agg_fun`	function. An R function that aggregates data, typically `max`, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.

`template_subject`	string. Template subject name, defaults to 'fsaverage'.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cast`	Whether a separate 'hemi' column should exist.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. Foe example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.
`agg_fun_extra_params`	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the agg_fun.

Value

dataframe with aggregated values for all subjects, with 2 columns and n rows, where n is the number of subjects. The 2 columns are 'subject_id' and '<hemi>.<measure>' (e.g., "lh.thickness"), the latter contains the aggregated data.

See Also

Other global aggregation functions: `group.morph.agg.native()`, `group.multimorph.agg.native()`, `group.multimorph.agg.standard()`

`group.morph.agg.standard.vertex`

Aggregate standard space morphometry data over subjects.

Description

Aggregate vertex-wise values over subjects, leading to one aggregated measure per vertex.

Usage

```
group.morph.agg.standard.vertex(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  fwhm,
```

```
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  agg_fun_extra_params = NULL,
  split_by_hemi = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm`	string. Smoothing as string, e.g. '10' or '25'.
`agg_fun`	function. An R function that aggregates data, typically `max`, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`template_subject`	string. Template subject name, defaults to 'fsaverage'.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. Foe example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.
`agg_fun_extra_params`	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the agg_fun.
`split_by_hemi`	logical, whether to return a hemilist

See Also

Other aggregation functions: `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `subject.atlas.agg()`

`group.morph.native` *Retrieve native space morphometry data for a group of subjects.*

Description

Load native space morphometry data (like 'surf/lh.area') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
group.morph.native(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  format = "curv",
  cortex_only = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	vector of strings. The subject identifiers.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`format,`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Defaults to FALSE.

Value

named list with native space morph data, the names are the subject identifiers from the subjects_list, and the values are morphometry data vectors (of different length, as each subject has a different vertex count in native space).

See Also

Other morphometry data functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `group.morph.standard()`, `subject.morph.native()`, `subject.morph.standard()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
   data = group.morph.native(subjects_dir, subjects_list, "thickness", "lh");

## End(Not run)
```

`group.morph.standard` *Retrieve standard space morphometry data for a group of subjects.*

Description

Load standard space morphometry data (like 'surf/lh.area') for a group of subjects from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
group.morph.standard(
  subjects_dir,
  subjects_list,
  measure,
  hemi = "both",
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  df = FALSE,
  df_t = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	vector of strings. The subject identifiers.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm,`	string. Smoothing as string, e.g. '10' or '25'.
`template_subject,`	string. Template subject name, defaults to 'fsaverage'.

`format,`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the template subject. Defaults to FALSE.
`df`	logical, whether to return a dataframe instead of the named list. The dataframe will have one subject per column, and *n* rows, where *n* is the number of vertices of the template subject surface.
`df_t`	logical, whether to return a transposed dataframe. Only one of df or df_t must be TRUE.

Value

named list with standard space morph data, the names are the subject identifiers from the subjects_list, and the values are morphometry data vectors (all with identical length, the data is mapped to a template subject).

See Also

Other morphometry data functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `group.morph.native()`, `subject.morph.native()`, `subject.morph.standard()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
  fulldata = group.morph.standard(subjects_dir, subjects_list, "thickness", "lh", fwhm='10');
   mean(fulldata$subject1);

   cortexdata = group.morph.standard(subjects_dir, subjects_list, "thickness",
    "lh", fwhm='10', cortex_only=FALSE);
   mean(cortexdata$subject1, na.rm=TRUE);

## End(Not run)
```

`group.morph.standard.sf`

Read combined data for a group from a single file.

Description

Read morphometry data for a group from a matrix in a single MGH or MGZ file.

Usage

```
group.morph.standard.sf(filepath, df = TRUE)
```

Arguments

`filepath`	character string, path to a file in MGH or MGZ format
`df`	logical, whether to return a data.frame, like `group.morph.standard`. If FALSE, the raw 4d matrix is returned.

Value

dataframe or 4d matrix, the morph data. See parameter 'df' for details.

Note

The file has typically been generated by running `mris_preproc` and/or `mri_surf2surf` on the command line, or written from R using `write.group.morph.standard.sf`. The file contains no information on the subject identifiers, you need to know the subjects and their order in the file. Same goes for the hemisphere.

See Also

`write.group.morph.standard.mf` to write the data to one file per hemi per subject instead. If you have created the input data file in FreeSurfer based on an FSGD file, you can read the subject identifiers from that FSGD file using `read.md.subjects.from.fsgd`.

`group.multimorph.agg.native`

Aggregate native space morphometry data for multiple measures over hemispheres for a group of subjects.

Description

Compute the mean (or other aggregates) over all vertices of a subject from native space morphometry data (like 'surf/lh.area'). You can specify several measures and hemispheres. Uses knowledge about the FreeSurfer directory structure to load the correct files.

Usage

```
group.multimorph.agg.native(
  subjects_dir,
  subjects_list,
  measures,
  hemis,
  agg_fun = mean,
  format = "curv",
  cast = TRUE,
```

```
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measures`	vector of strings. Names of the vertex-wise morphometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded.
`hemis`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`agg_fun`	function. An R function that aggregates data, typically `max`, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cast`	Whether a separate 'hemi' column should exist.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. Foe example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.
`agg_fun_extra_params`	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the agg_fun.

Value

dataframe with aggregated values over all measures and hemis for all subjects, with m columns and n rows, where n is the number of subjects. The m columns are 'subject_id' and '<hemi>.<measure>' (e.g., "lh.thickness") for all combinations of hemi and measure, the latter contains the aggregated data.

See Also

Other global aggregation functions: `group.morph.agg.native()`, `group.morph.agg.standard()`, `group.multimorph.agg.standard()`

Examples

```
## Not run:
    subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
    subjects_list = c("subject1", "subject2")
   data = group.multimorph.agg.native(subjects_dir, subjects_list, c("thickness", "area"),
     c("lh", "rh"), cast=FALSE, cortex_only=TRUE, agg_fun=mean,
     agg_fun_extra_params=list("na.rm"=TRUE));
    head(data);

## End(Not run)
```

`group.multimorph.agg.standard`

Aggregate standard space (fsaverage) morphometry data for multiple measures over hemispheres for a group of subjects.

Description

Compute the mean (or other aggregates) over all vertices of a subject from standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh'). You can specify several measures and hemispheres. Uses knowledge about the FreeSurfer directory structure to load the correct files.

Usage

```
group.multimorph.agg.standard(
  subjects_dir,
  subjects_list,
  measures,
  hemis,
  fwhm,
  agg_fun = mean,
  template_subject = "fsaverage",
  format = "mgh",
  cast = TRUE,
  cortex_only = FALSE,
  agg_fun_extra_params = NULL
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.

`measures,`	vector of strings. Names of the vertex-wise morphometry measures. E.g., c("area", "thickness"). Used to construct the names of the morphometry file to be loaded.
`hemis,`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm,`	string. Smoothing as string, e.g. '10' or '25'.
`agg_fun,`	function. An R function that aggregates data, typically `max`, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`template_subject,`	string. Template subject name, defaults to 'fsaverage'.
`format,`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cast,`	Whether a separate 'hemi' column should exist.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subjects. Also not that the aggregation function will need to be able to cope with NA values if you set this to TRUE. You can use 'agg_fun_extra_params' if needed to achieve that, depending on the function. Foe example, if you use the `mean` function, you could set `agg_fun_extra_params=list("na.rm"=TRUE)` to get the mean of the vertices which are not masked. Defaults to FALSE.
`agg_fun_extra_params`	named list, extra parameters to pass to the aggregation function 'agg_fun' besides the loaded morphometry data. This is useful if you have masked the data and need to ignore NA values in the agg_fun.

Value

dataframe with aggregated values over all measures and hemis for all subjects, with m columns and n rows, where n is the number of subjects. The m columns are 'subject_id' and '<hemi>.<measure>' (e.g., "lh.thickness") for all combinations of hemi and measure, the latter contains the aggregated data.

See Also

Other global aggregation functions: `group.morph.agg.native()`, `group.morph.agg.standard()`, `group.multimorph.agg.native()`

`group.surface` *Retrieve surface mesh data for a group of subjects.*

Description

Retrieve surface mesh data for a group of subjects.

Usage

```
group.surface(
  subjects_dir,
  subjects_list,
  surface,
  hemi = "both",
  force_hemilist = TRUE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	vector of strings. The subject identifiers.
`surface`	character string, the surface to load. Something like 'white' or 'pial'.
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the mesh files to be loaded.
`force_hemilist`	logical, whether to force the individual values inside the named return value list to be hemilists (even if the 'hemi' parameter is not set to 'both'). If this is FALSE, the inner values will contain the respective (lh or rh) surface only.

Value

named list of surfaces: Each name is a subject identifier from subjects_list, and the values are hemilists of 'fs.surface' instances.

Examples

```
## Not run: 
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subjects_list = c("subject1", "subject2");
   surfaces = group.surface(subjects_dir, subjects_list, 'white', "both");

## End(Not run)
```

`groupmorph.split.hemilist` *Split a per-vertex group data matrix for both hemispheres into a hemilist at given index.*

Description

Split a per-vertex group data matrix for both hemispheres into a hemilist at given index.

Usage

```
groupmorph.split.hemilist(data, numverts_lh)
```

Arguments

`data`	numerical matrix or dataframe of per-vertex data, with subjects in columns
`numverts_lh`	scalar positive integer, the number of vertices in the left hemisphere mesh (defining the index where to split).

Value

`hemilist` of the data, split at the index.

Examples

```
## Not run: 
    fsbrain::download_optional_data();
    fsbrain::download_fsaverage(TRUE);
    subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
    subjects_list = c("subject1", "subject2");
   data = group.morph.standard(subjects_dir, subjects_list, "thickness", "lh", fwhm='10');
    numverts_lh = subject.num.verts(subjects_dir, "fsaverage", hemi="lh");
    data_hemilist = groupmorph.split.hemilist(data, numverts_lh);

## End(Not run)
```

`hasIn` *Check for values in nested named lists*

Description

Check for values in nested named lists

Usage

```
hasIn(named_list, listkeys)
```

Arguments

`named_list`	a named list
`listkeys`	vector of character strings, the nested names of the lists

Value

whether a non-NULL value exists at the path

Examples

```
data = list("regions"=list("frontal"=list("thickness"=2.3, "area"=2345)));
hasIn(data, c("regions", "nosuchregion"));   # FALSE
```

`hemilist` *Create a hemilist from lh and rh data.*

Description

Simply runs `list('lh' = lh_data, 'rh' = rh_data)`: A hemilist (short for hemisphere list) is just a named list with entries 'lh' and/or 'rh', which may contain anything. Hemilists are used as parameters and return values in many `fsbrain` functions. The 'lh' and 'rh' keys typically contain surfaces or vectors of morphometry data.

Usage

```
hemilist(lh_data = NULL, rh_data = NULL)
```

Arguments

`lh_data`	something to wrap, typically some data for a hemisphere, e.g., a vector of morphometry data values.
`rh_data`	something to wrap, typically some data for a hemisphere, e.g., a vector of morphometry data values.

Value

named list, with the 'lh_data' in the 'lh' key and the 'rh_data' in the 'rh' key.

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.from.prefixed.list()`, `hemilist.get.combined.data` `hemilist.unwrap()`, `hemilist.wrap()`, `is.hemilist()`

Examples

```
lh_data = rnorm(163842, 5.0, 1.0);
rh_data = rnorm(163842, 5.0, 1.0);
hl = hemilist(lh_data, rh_data);
```

`hemilist.derive.hemi` *Derive 'hemi' string from the data in a hemilist*

Description

Derive 'hemi' string from the data in a hemilist

Usage

```
hemilist.derive.hemi(hemilist)
```

Arguments

`hemilist`	hemilist, an existing hemilist

Value

character string, one of 'lh', 'rh' or 'both'

Note

See `hemilist` for details.

See Also

Other hemilist functions: `hemilist.from.prefixed.list()`, `hemilist.get.combined.data()`, `hemilist.unwrap()`, `hemilist.wrap()`, `hemilist()`, `is.hemilist()`

`hemilist.from.prefixed.list` *Create a hemilist from a named list with keys prefixed with 'lh_' and 'rh_'.*

Description

A hemilist is a named list with entries 'lh' and/or 'rh', see `hemilist`.

Usage

```
hemilist.from.prefixed.list(
  named_list,
  report_ignored = TRUE,
  return_ignored = FALSE
)
```

Arguments

`named_list`	a named list, the keys must start with 'lh_' or 'rh_' to be assigned to the 'lh' and 'rh' entries of the returned hemilist. Other entries will be ignored.
`report_ignored`	logical, whether to print a message with the ignored entries, if any.
`return_ignored`	logical, whether to add a key 'ignored' to the returned hemilist, containing the ignored entries.

Value

a hemilist

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.get.combined.data()`, `hemilist.unwrap()`, `hemilist.wrap()`, `hemilist()`, `is.hemilist()`

`hemilist.get.combined.data` *Get combined data of hemi list*

Description

Get combined data of hemi list

Usage

```
hemilist.get.combined.data(hemi_list)
```

Arguments

`hemi_list`	named list, can have entries 'lh' and/or 'rh', see `hemilist`

Value

the data combined with `c`, or NULL if both entries are NULL.

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.from.prefixed.list()`, `hemilist.unwrap()`, `hemilist.wrap()`, `hemilist()`, `is.hemilist()`

`hemilist.unwrap` *Unwrap hemi data from a named hemi list.*

Description

Unwrap hemi data from a named hemi list.

Usage

```
hemilist.unwrap(hemi_list, hemi = NULL, allow_null_list = FALSE)
```

Arguments

`hemi_list`	named list, can have entries ’lh’ and/or ’rh’, see `hemilist`.
`hemi`	character string, the hemi data name to retrieve from the list. Can be NULL if the list only has a single entry.
`allow_null_list`	logical, whether to silently return NULL instead of raising an error if ’hemi_list’ is NULL

Value

the data

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.from.prefixed.list()`, `hemilist.get.combined.data` `hemilist.wrap()`, `hemilist()`, `is.hemilist()`

`hemilist.wrap` *Wrap data into a named hemi list.*

Description

Wrap data into a named hemi list.

Usage

```
hemilist.wrap(data, hemi, hemilist = NULL)
```

Arguments

`data`	something to wrap, typically some data for a hemisphere, e.g., a vector of morphometry data values. If NULL, the name will not be created.
`hemi`	character string, one of 'lh' or 'rh'. The name to use for the data in the returned list.
`hemilist`	optional `hemilist`, an existing hemilist to add the entry to. If left at the default value 'NULL', a new list will be created.

Value

a `hemilist`: a named list, with the 'data' in the name given by parameter 'hemi'

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.from.prefixed.list()`, `hemilist.get.combined.data` `hemilist.unwrap()`, `hemilist()`, `is.hemilist()`

`highlight.points.spheres`

Draw small 3D spheres at given points.

Description

Draw small 3D spheres at given points.

Usage

```
highlight.points.spheres(coords, color = "#FF0000", radius = 1)
```

Arguments

`coords`	double vector or nx3 double matrix, the xyz point coordinates.
`color`	the sphere color, like `'#FF0000'` or `"red"`.
`radius`	double, the sphere radius

See Also

Other 3d utility functions: `highlight.vertices.spheres()`, `vertex.coords()`

`highlight.vertices.on.subject` *Highlight vertices given by index on a subject's meshes by coloring faces.*

Description

Highlight vertices given by index on a subject's meshes by coloring faces.

Usage

```
highlight.vertices.on.subject(
  subjects_dir,
  vis_subject_id,
  verts_lh = NULL,
  verts_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  color_bg = "#FEFEFE",
  color_verts_lh = "#FF0000",
  color_verts_rh = "#FF4500",
  k = 0L
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`verts_lh`	integer vector, the indices of left hemisphere vertices.
`verts_rh`	integer vector, the indices of right hemisphere vertices.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.

`color_bg`	background color.
`color_verts_lh`	vector of colors to visualize on the left hemisphere surface. Length must match number of vertices in 'verts_lh', or be a single color.
`color_verts_rh`	vector of colors to visualize on the right hemisphere surface. Length must match number of vertices in 'verts_rh', or be a single color.
`k`	integer, radius to extend neighborhood (for better visibility).

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other surface visualization functions: `highlight.vertices.on.subject.spheres()`, `vis.color.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   highlight.vertices.on.subject(subjects_dir, 'subject1',
     verts_lh=c(5000, 100000), verts_rh=c(300, 66666), views="si");

## End(Not run)
```

`highlight.vertices.on.subject.spheres`

Highlight vertices given by index on a subject's meshes by coloring faces.

Description

Highlight vertices given by index on a subject's meshes by coloring faces.

Usage

```
highlight.vertices.on.subject.spheres(
  subjects_dir,
  vis_subject_id,
  vertices,
  surface = "white",
```

```
  patch_size = 25,
  show_patch = TRUE,
  style = "glass2",
  export_img = NULL,
  sphere_colors = c("#FF0000"),
  sphere_radius = 3,
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`vertices`	positive integer vector, the vertex indices over both hemispheres. Alternative to using verts_lh and verts_rh parameters, only one of them must be used at once.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`patch_size`	double, geodesic radius in which to draw a patch on the mesh around the verts. Pass `NULL` to disable.
`show_patch`	logical (or a vector with one logical value per entry in 'vertices'), whether to show colored geodesic patches at the highlighted vertices.
`style`	character string or rgl rendering style, see `get.rglstyle`.
`export_img`	character string, the path to the output image if you want to export a high-quality image, NULL if you want live visualization instead.
`sphere_colors`	the sphere colors like '#FF0000', can be a single one for all or one per sphere
`sphere_radius`	double, a single radius for all spheres
`...`	extra parameters passed on to `vis.data.on.subject`. Use this to set a custom colormap etc.

Value

list of coloredmeshes. The coloredmeshes used for the visualization. If export_img is set, the export return value is returned instead.

Note

If no patches are visualized, the color used for the brain can be set with `options("fsbrain.brain_na_color"="#FF0000")`.

See Also

Other visualization functions: `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other surface visualization functions: `highlight.vertices.on.subject()`, `vis.color.on.subject()`

Examples

```
## Not run:
   fsbrain::download_fsaverage(T);
   subjects_dir = fsaverage.path();
   mkco = list('colFn'=viridis::viridis, 'n'=300);
   # Ex.1: highlight with patches and custom colormap:
   highlight.vertices.on.subject.spheres(subjects_dir, 'fsaverage',
     vertices=c(300, 5000, 100000), makecmap_options = mkco);
   # Ex.2: show patches on some (red) vertices, not on blue ones:
   highlight.vertices.on.subject.spheres(subjects_dir, 'fsaverage',
     vertices=c(300, 5000, 100000, 300000), show_patch = c(T,F,T,F),
     sphere_colors = c("red", "blue", "red", "blue"));

## End(Not run)
```

`highlight.vertices.spheres`

Draw small 3D spheres at given brain mesh vertices. Supports full brain (2 meshes) as well.

Description

Draw small 3D spheres at given brain mesh vertices. Supports full brain (2 meshes) as well.

Usage

```
highlight.vertices.spheres(surface, vertices, ...)
```

Arguments

`surface`	an fs.surface instance, see `subject.surface` function. Can also be a hemilist of surfaces, in which case the vertices can be indices over both meshes (in range `1..(nv(lh)+nv(rh))`).
`vertices`	vector of positive integers, the vertex indices. Values which are outside of the valid indices for the surface will be silently ignored, making it easier to work with the two hemispheres.
`...`	Parameters passed to `highlight.points.spheres`.

Note

This function will draw into the current window and add to the scene, so it can be called after visualizing a mesh. See the example.

See Also

Other 3d utility functions: `highlight.points.spheres()`, `vertex.coords()`

Examples

```
## Not run:
lh_surf = subject.surface('~/data/study1', 'subject1',
 surface = "white", hemi = "lh");
vis.fs.surface(lh_surf, style="semitransparent");
highlight.vertices.spheres(lh_surf,
  vertices = c(3225L, 4300L, 5500L),
  color = c("green", "blue", "red"));

## End(Not run)
```

`images.dimmax` *Compute max width and height of magick images.*

Description

Compute max width and height of magick images.

Usage

```
images.dimmax(images)
```

Arguments

`images`	a vector/stack of magick images. See `magick::image_blank` or other methods to get one.

Value

named list with entries 'width' and 'height'

`is.fs.coloredmesh` *Check whether object is an fs.coloredmesh (S3)*

Description

Check whether object is an fs.coloredmesh (S3)

Usage

```
is.fs.coloredmesh(x)
```

Arguments

`x`	any 'R' object

Value

TRUE if its argument is a coloredmesh (that is, has "fs.coloredmesh" amongst its classes) and FALSE otherwise.

`is.fs.coloredvoxels` *Check whether object is an fs.coloredvoxels instance (S3)*

Description

Check whether object is an fs.coloredvoxels instance (S3)

Usage

```
is.fs.coloredvoxels(x)
```

Arguments

`x`	any ‘R‘ object

Value

TRUE if its argument is a fs.coloredvoxels instance (that is, has "fs.coloredvoxels" among its classes) and FALSE otherwise.

`is.fsbrain` *Check whether object is an fsbrain (S3)*

Description

Check whether object is an fsbrain (S3)

Usage

```
is.fsbrain(x)
```

Arguments

`x`	any ‘R‘ object

Value

TRUE if its argument is an fsbrain (that is, has "fsbrain" amongst its classes) and FALSE otherwise.

`is.hemilist` *Check whether x is a hemilist*

Description

A hemilist is a named list with entries 'lh' and/or 'rh', see `hemilist`.

Usage

```
is.hemilist(x)
```

Arguments

`x`	any R object

Value

whether 'x' is a hemilist

See Also

Other hemilist functions: `hemilist.derive.hemi()`, `hemilist.from.prefixed.list()`, `hemilist.get.combined.data` `hemilist.unwrap()`, `hemilist.wrap()`, `hemilist()`

`label.border` *Compute border of a label.*

Description

Compute the border of a label (i.e., a subset of the vertices of a mesh). The border thickness can be specified. Useful to draw the outline of a region, e.g., a significant cluster on the surface or a part of a ROI from a brain parcellation.

Usage

```
label.border(
  surface_mesh,
  label,
  inner_only = TRUE,
  expand_inwards = 0L,
  derive = FALSE
)
```

Arguments

`surface_mesh`	surface mesh, as loaded by `subject.surface` or `read.fs.surface`.
`label`	instance of class ‘fs.label‘ or an integer vector, the vertex indices. This function only makes sense if they form a patch on the surface, but that is not checked.
`inner_only`	logical, whether only faces consisting only of label_vertices should be considered to be label faces. If FALSE, faces containing at least one label vertex will be used. Defaults to TRUE. Leave this alone if in doubt, especially if you want to draw several label borders which are directly adjacent on the surface.
`expand_inwards`	integer, border thickness extension. If given, once the border has been computed, it is extended by the given graph distance. It is guaranteed that the border only extends inwards, i.e., it will never extend to vertices which are not part of the label.
`derive`	logical, whether the returned result should also include the border edges and faces in addition to the border vertices. Takes longer if requested, defaults to FALSE.

Value

the border as a list with the following entries: ‘vertices‘: integer vector, the vertex indices of the border. Iff the parameter ‘derive‘ is TRUE, the following two additional fields are included: ‘edges‘: integer matrix of size (n, 2) for n edges. Each row defines an edge by its start and target vertex. ‘faces‘: integer vector, the face indices of the border.

See Also

Other surface mesh functions: `face.edges()`, `mesh.vertex.included.faces()`, `mesh.vertex.neighbors()`, `subject.surface()`, `vis.path.along.verts()`

`label.colFn` *A simple colormap function for binary colors.*

Description

Useful for plotting labels.

Usage

```
label.colFn(n = 2L, col_a = "#228B22", col_b = "#FFFFFF")
```

Arguments

`n`	positive integer, the number of colors. Must be 1 or 2 for this function.
`col_a`	color string, the foreground color
`col_b`	color string, the background color

Value

vector of 'n' RGB colorstrings

`label.colFn.inv` *A simple colormap function for binary colors.*

Description

Useful for plotting labels.

Usage

```
label.colFn.inv(n = 2L, col_a = "#228B22", col_b = "#FFFFFF")
```

Arguments

`n`	positive integer, the number of colors. Must be 1 or 2 for this function.
`col_a`	color string, the foreground color
`col_b`	color string, the background color

Value

vector of 'n' RGB colorstrings

`label.from.annotdata` *Extract a region from an annotation as a label.*

Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to ‘NaN‘ or to extract only values from a certain region.

Usage

```
label.from.annotdata(
  annotdata,
  region,
  return_one_based_indices = TRUE,
  invert = FALSE,
  error_on_invalid_region = TRUE
)
```

Arguments

`annotdata,`	annotation. An annotation for one hemisphere, as returned by `subject.annot`. This must be the loaded data, not a path to a file.
`region,`	string. A valid region name for the annotation, i.e., one of the regions of the atlas used to create the annotation.
`return_one_based_indices,`	logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.
`invert,`	logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
`error_on_invalid_region,`	logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

Value

integer vector with label data: the list of vertex indices in the label. See 'return_one_based_indices' for important information.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot`
`spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

`label.to.annot` *Merge several labels into an annotation*

Description

Merge several labels and a colortable into an annotation.

Usage

```
label.to.annot(
  label_vertices_by_region,
  num_vertices_in_surface,
  colortable_df = NULL,
  index_of_unknown_region = 1L
)
```

Arguments

`label_vertices_by_region`	named list of integer vectors, the keys are strings which define region names, and the values are integer vectors: the vertex indices of the region.
`num_vertices_in_surface`	integer, total number of vertices in the surface mesh
`colortable_df`	NULL or dataframe, a colortable. It must contain the columns 'struct_name', 'r', 'g', 'b', and 'a'. All other columns will be derived if missing. The entries in 'struct_name' must match keys from the 'label_vertices_by_region' parameter. There must be one more row in here than there are labels. This row identifies the 'unknown' region (see also parameter 'index_of_unknown_region'). If NULL, a colortable will be auto-generated.
`index_of_unknown_region`	positive integer, the index of the row in 'colortable_df' that defines the 'unknown' or 'background' region to which all vertices will be assigned which are *not* part of any of the given labels.

Value

an annotation, see `read.fs.annot` for details.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
# Create two labels. Real-word labels would have more vertices, of course.
label1 = c(46666, 467777);
label2 = c(99888, 99889);
label_vertices = list("region1"=label1, "region2"=label2);
colortable_df = data.frame("struct_index"=seq(0, 2),
 "struct_name"=c("unknown", "region1", "region2"),
 "r"=c(255L, 255L, 0L), "g"=c(255L, 0L, 255L), "b"=c(255L, 0L, 0L), "a"=c(0L, 0L, 0L));
annot = label.to.annot(label_vertices, 100000, colortable_df);
```

`labeldata.from.mask` *Create labeldata from a mask.*

Description

Create labeldata from a mask. This function is trivial and only calls `which` after performing basic sanity checks.

Usage

```
labeldata.from.mask(mask, invert = FALSE)
```

Arguments

`mask`	a logical vector
`invert`	Whether to report the inverse the mask before determining the indices. Defaults to FALSE.

Value

labeldata. The list of indices which are TRUE in the mask (or the ones which FALSE if 'invert' is TRUE).

See Also

Other label data functions: `group.label()`, `mask.from.labeldata.for.hemi()`, `subject.label()`

`limit_fun` *Get data limiting function.*

Description

Get data limiting function to use in rglactions as 'trans_fun' to transform data. This is typically used to limit the colorbar in a plot to a certain range. This is similar to `clip.data` or `clip_fun`, but uses absolute values instead of percentiles to clip.

Usage

```
limit_fun(vmin, vmax)
```

Arguments

`vmin`	numerical scalar, the lower border. Data values below this will be set to vmin in the return value.
`vmax`	numerical scalar, the upper border. Data values above this will be set to vmax in the return value.

Value

a function that takes as argument the data, and clips it to the requested range. I.e., values outside the range will be set to the closest border value ('vmin' or 'vmax'). Designed to be used as `rglactions$trans_fun` in vis functions, to limit the colorbar and data range.

See Also

`rglactions`

Examples

```
rglactions = list("trans_fun"=limit_fun(2,3));
```

`limit_fun_na` *Get data limiting function to NA.*

Description

Get data limiting function to use in `rglactions` as 'trans_fun' to transform data. This is typically used to limit the colorbar in a plot to a certain range. This is similar to `clip.data`, but uses absolute values instead of percentiles to clip.

Usage

```
limit_fun_na(vmin, vmax)
```

Arguments

`vmin`	numerical scalar, the lower border. Data values below this will be set to 'NA' in the return value.
`vmax`	numerical scalar, the upper border. Data values above this will be set to 'NA' in the return value.

Value

a function that takes as argument the data, and clips it to the requested range. I.e., values outside the range will be set to 'NA'. Designed to be used as `rglactions$trans_fun` in vis functions, to limit the colorbar and data range.

Note

This is useful for thresholding stuff like t-value maps. All values outside the range will be displayed as the background color.

See Also

`limit_fun_na_inside` which will set the values inside the range to 'NA'.

Examples

```
rglactions = list("trans_fun"=limit_fun_na(2,3));
```

`limit_fun_na_inside` *Get data limiting function, setting values inside range to NA.*

Description

Get data limiting function to use in `rglactions` as 'trans_fun' to transform data.

Usage

```
limit_fun_na_inside(vmin, vmax)
```

Arguments

`vmin`	numerical scalar, the lower border. Data values between this and vmax will be set to 'NA' in the return value.
`vmax`	numerical scalar, the upper border. See 'vmin'.

Value

a function that takes as argument the data, and clips it to the requested range. I.e., values inside the range will be set to 'NA'. Designed to be used as `rglactions$trans_fun` in vis functions.

Note

This is useful for thresholding data plotted with a background. All values inside the range will set to NA and be transparent, and thus be displayed as the background color.

See Also

`limit_fun_na` which will set the values outside the range to 'NA'.

Examples

```
rglactions = list("trans_fun"=limit_fun_na_inside(2,3));
```

`list_optional_data` *Get file names available in package cache.*

Description

Get file names of optional data files which are available in the local package cache. You can access these files with get_optional_data_file().

Usage

```
list_optional_data()
```

Value

vector of strings. The file names available, relative to the package cache.

`mask.from.labeldata.for.hemi`

Create a binary mask from labels.

Description

Create a binary mask for the data of a single hemisphere from one or more labels. A label contains the vertex indices which are part of it, but often having a mask in more convenient.

Usage

```
mask.from.labeldata.for.hemi(
  labels,
  num_vertices_in_hemi,
  invert_labels = FALSE,
  existing_mask = NULL
)
```

Arguments

`labels`	list of labels. A label is just a vector of vertex indices. It can be created manually, but is typically loaded from a label file using `subject.label`.
`num_vertices_in_hemi`	integer. The number of vertices of the surface for which the mask is created. This must be for a single hemisphere.
`invert_labels`	logical, whether to invert the label data.
`existing_mask`	an existing mask to modify or NULL. If it is NULL, a new mask will be created before applying any labels, and the values set during initialization of this new mask are the negation of the 'invert_label' parameter. Defaults to NULL.

Value

logical vector. The mask. It contains a logical value for each vertex. By default, the vertex indices from the labels are FALSE and the rest are TRUE, but this can be changed with the parameter 'invert_labels'.

See Also

Other label data functions: `group.label()`, `labeldata.from.mask()`, `subject.label()`

Other mask functions: `coloredmesh.from.mask()`, `vis.mask.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();

  # Define the data to use:
  subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
  subject_id = 'subject1';
  surface = 'white';
  hemi = 'both';
  atlas = 'aparc';
  region = 'bankssts';

  # Create a mask from a region of an annotation:
  lh_annot = subject.annot(subjects_dir, subject_id, 'lh', atlas);
  rh_annot = subject.annot(subjects_dir, subject_id, 'rh', atlas);
  lh_label = label.from.annotdata(lh_annot, region);
  rh_label = label.from.annotdata(rh_annot, region);
  lh_mask = mask.from.labeldata.for.hemi(lh_label, length(lh_annot$vertices));
  rh_mask = mask.from.labeldata.for.hemi(rh_label, length(rh_annot$vertices));

  # Edit the mask: add the vertices from another region to it:
  region2 = 'medialorbitofrontal';
  lh_label2 = label.from.annotdata(lh_annot, region2);
  rh_label2 = label.from.annotdata(rh_annot, region2);
  lh_mask2 = mask.from.labeldata.for.hemi(lh_label2, length(lh_annot$vertices),
   existing_mask = lh_mask);
  rh_mask2 = mask.from.labeldata.for.hemi(rh_label2, length(rh_annot$vertices),
   existing_mask = rh_mask);

## End(Not run)
```

`mesh.vertex.neighbors` *Compute neighborhood of a vertex*

Description

Given a set of query vertex indices and a mesh *m*, compute all vertices which are adjacent to the query vertices in the mesh. A vertex *u* is *adjacent* to another vertex *v* iff there exists an edge *e = (u, v)* in *m*. While you could call this function repeatedly with the old output as its new input to extend the neighborhood, you should maybe use a proper graph library for this.

Usage

```
mesh.vertex.neighbors(
  surface,
  source_vertices,
  k = 1L,
  restrict_to_vertices = NULL
)
```

Arguments

`surface`	a surface as returned by functions like `subject.surface` or `read.fs.surface`.
`source_vertices`	Vector of source vertex indices.
`k`	positive integer, how often to repeat the procedure and grow the neighborhood, using the output ‘vertices‘ as the ‘source_vertices‘ for the next iteration. Warning: settings this to high values will be very slow for large meshes.
`restrict_to_vertices`	integer vector of vertex indices. If given, the neighborhood growth will be limited to the given vertex indices. Defaults to NULL, which means the neighborhood is not restricted.

Value

the neighborhood as a list with two entries: "faces": integer vector, the face indices of all faces the source_vertices are a part of. "vertices": integer vector, the unique vertex indices of all vertices of the faces in the 'faces' property. These vertex indices include the indices of the source_vertices themselves.

See Also

Other surface mesh functions: `face.edges()`, `label.border()`, `mesh.vertex.included.faces()`, `subject.surface()`, `vis.path.along.verts()`

`mkco.cluster` *Return recommended 'makecmap_options' for diverging cluster data.*

Description

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap_options' to the vis.* functions, e.g., `vis.subject.morph.native`.

Usage

```
mkco.cluster()
```

Value

named list, visualization settings to be used as 'makecmap_options' for diverging data.

Note

This uses a cyan blue red yellow colormap, which is popular for displaying clusters in neuroscience.

`mkco.div` *Return recommended 'makecmap_options' for diverging data.*

Description

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap_options' to the vis.* functions, e.g., `vis.subject.morph.native`.

Usage

```
mkco.div()
```

Value

named list, visualization settings to be used as 'makecmap_options' for diverging data.

`mkco.heat` *Return recommended 'makecmap_options' for sequential data with heatmap style.*

Description

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap_options' to the vis.* functions, e.g., `vis.subject.morph.native`.

Usage

```
mkco.heat()
```

Value

named list, visualization settings to be used as 'makecmap_options' for sequential data with heatmap style.

`mkco.seq` *Return recommended 'makecmap_options' for sequential data.*

Description

This function returns recommended visualization settings (a colormap function and suitable other settings) for the given type of data. The return value is meant to be passed as parameter 'makecmap_options' to the vis.* functions, e.g., `vis.subject.morph.native`.

Usage

```
mkco.seq()
```

Value

named list, visualization settings to be used as 'makecmap_options' for sequential data.

`numverts.lh` *Determine vertex count of left hemi from hemilist of surfaces or the count itself.*

Description

Determine vertex count of left hemi from hemilist of surfaces or the count itself.

Usage

```
numverts.lh(surfaces)
```

Arguments

`surfaces`	hemilist of surfaces, or a single integer, which will be interpreted as the number of vertices of the left hemisphere surface.

Value

integer, the number of vertices.

`numverts.rh` *Determine vertex count of right hemi from hemilist of surfaces or the count itself.*

Description

Determine vertex count of right hemi from hemilist of surfaces or the count itself.

Usage

```
numverts.rh(surfaces)
```

Arguments

`surfaces`	hemilist of surfaces, or a single integer, which will be interpreted as the number of vertices of the right hemisphere surface.

Value

integer, the number of vertices.

`principal.curvatures` *Computes principal curvatures according to 2 definitions from raw k1 and k2 values.*

Description

Computes principal curvatures according to 2 definitions from raw k1 and k2 values.

Usage

```
principal.curvatures(k1_raw, k2_raw)
```

Arguments

`k1_raw`	numerical vector, one of the two principal curvatures, one value per vertex
`k2_raw`	numerical vector, the other one of the two principal curvatures, one value per vertex

Value

a named ’principal_curvatures’ list, with entries ’principal_curvature_k1’: larger value of k1_raw, k2_raw. ’principal_curvature_k2’: smaller value of k1_raw, k2_raw. ’principal_curvature_k_major’: larger value of abs(k1_raw), abs(k2_raw). ’principal_curvature_k_minor’: smaller value of abs(k1_raw), abs(k2_raw).

Note

To obtain k1_raw and k2_raw, use `surface.curvatures` to compute it from a mesh, or load the FreeSurfer files `'surf/?h.white.max'` and `'surf/?h.white.min'`.

`print.fs.coloredmesh` *Print description of a brain coloredmesh (S3).*

Description

Print description of a brain coloredmesh (S3).

Usage

```
## S3 method for class 'fs.coloredmesh'
print(x, ...)
```

Arguments

`x`	brain surface with class ‘fs.coloredmesh‘.
`...`	further arguments passed to or from other methods

`print.fs.coloredvoxels` *Print description of fs.coloredvoxels (S3).*

Description

Print description of fs.coloredvoxels (S3).

Usage

```
## S3 method for class 'fs.coloredvoxels'
print(x, ...)
```

Arguments

`x`	brain voxel tris with class ‘fs.coloredvoxels‘.
`...`	further arguments passed to or from other methods

`print.fsbrain` *Print description of an fsbrain (S3).*

Description

Print description of an fsbrain (S3).

Usage

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

Arguments

`x`	fsbrain instance with class ‘fsbrain‘.
`...`	further arguments passed to or from other methods

`qc.for.group` *Perform data quality check based on computed region stats.*

Description

Determine subjects that potentially failed segmentation, based on region-wise morphometry data. The stats can be computed from any kind of data, but something like area or volume most likely works best. The stats are based on the mean of the region values, so the measure should at least roughly follow a normal distribution.

Usage

```
qc.for.group(subjects_dir, subjects_list, measure, atlas, hemi = "both", ...)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`atlas`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`hemi`	string, one of 'lh', 'rh', 'split', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded. If set to 'both', combined data for 'lh' and 'rh' will be used. If 'split', the data for hte two hemispheres will go into seprate columns, with column names having 'lh_' and 'rh_' prefixes.
`...`	parameters passed to `qc.from.regionwise.df`.

Value

qc result as a hemilist, each entry contains a named list as returned by `qc.from.regionwise.df`.

See Also

Other quality check functions: `qc.from.regionwise.df()`, `qc.from.segstats.table()`

`qc.from.regionwise.df` *Perform data quality check based on a dataframe containing aggregated region-wise data.*

Description

Determine subjects that potentially failed segmentation, based on region-wise data. The data can be anything, but there must be one numerical value per subject per region.

Usage

```
qc.from.regionwise.df(
  rdf,
  z_threshold = 2.8,
  verbosity = 0L,
  num_bad_regions_allowed = 1L
)
```

Arguments

`rdf`	data.frame, the region data. The first column must contain the subject identifier, all other columns should contain numerical data for a single region. (Each row represents a subject.) This can be produced by calling `group.agg.atlas.native` or by parsing a text file produced by the FreeSurfer tool 'aparcstats2table' (see `fsbrain:::qc.from.segstats.table` for parsing code).
`z_threshold`	numerical, the cutoff value for considering a subject an outlier (in standard deviations).
`verbosity`	integer, controls the output to stdout. 0=off, 1=normal, 2=verbose.
`num_bad_regions_allowed`	integer, the number of regions in which subjects are allowed to be outliers without being reported as potentially failed segmentation

Value

named list with entries: 'failed_subjects': vector of character strings, the subject identifiers which potentially failed segmentation. 'mean_dists_z': distance to mean, in standard deviations, per subject per region. 'num_outlier_subjects_per_region': number of outlier subjects by region. 'metadata': named list of metadata, e.g., hemi, atlas and measure used to compute these QC results.

See Also

Other quality check functions: `qc.for.group()`, `qc.from.segstats.table()`

`qc.from.segstats.tables`

Perform data quality check based on a segstats table.

Description

Determine subjects that potentially failed segmentation, based on segstats table data. The input table file must be a segmentation or parcellation table, generated by running the FreeSurfer tools 'aparcstats2table' or 'asegstats2table' for your subjects.

Usage

```
qc.from.segstats.tables(filepath_lh, filepath_rh, ...)
```

Arguments

`filepath_lh`	path to left hemisphere input file, a tab-separated file generated by running the FreeSurfer tools 'aparcstats2table' or 'asegstats2table'. The command line in the system shell would be something like 'aparcstats2table_bin –subjectsfile $subjects_file –meas $measure –hemi $hemi -t $aparc_output_table'.
`filepath_rh`	path to equivalent right hemisphere input file.
`...`	parameters passed to `qc.from.regionwise.df`.

Value

qc result as a hemilist, each entry contains a named list as returned by `qc.from.regionwise.df`.

`qc.vis.failcount.by.region`

Visualize the number of outlier subjects per region in your dataset.

Description

The function helps you to see which regions are affected the most by QC issues: for each region, it plots the number of subjects which are outliers in the region.

Usage

```
qc.vis.failcount.by.region(
  qc_res,
  atlas,
  subjects_dir = fsaverage.path(),
  subject_id = "fsaverage",
  ...
)
```

Arguments

`qc_res`	hemilist of QC results, as returned by functions like `qc.for.group` or `qc.from.segstats.tables`.
`atlas`	string. The brain atlas to use. E.g., 'aparc' or 'aparc.a2009s'.
`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`...`	extra parameters passed to `vis.region.values.on.subject`. E.g., to change to interactive view, get a colorbar and better resolution, try: `draw_colorbar=T`, `rgloptions = rglo()`, `views='si'`.

Note

You can visualize this on any subject you like, 'fsaverage' is a typical choice. The atlas must be the one used during the QC step.

`qdec.table.skeleton` *Generate skeleton dataframe for FreeSurfer QDEC long file from subjects list.*

Description

Generate skeleton dataframe for FreeSurfer QDEC long file from subjects list.

Usage

```
qdec.table.skeleton(
  subjects_list,
  isi = rep(0.8, length(subjects_list)),
  isi_name = "years",
  timepoint_names = c("_MR1", "_MR2")
)
```

Arguments

`subjects_list`	vector of character strings, the Freesurfer subject IDs (cross-sectional names, without any suffixes like `_MR1`, `long`, etc.)
`isi`	numerical vector, the inter-scan interval for the subjects, in a unit of your choice. Typically in years.
`isi_name`	character string, the name for the isi columns. Defaults to "years".
`timepoint_names`	vector of character strings, the timepoint names. These are mandatory for QDEC, so there should be very little reason to change them. Leave along unless you know what you are doing.

Value

data.frame with 3 columns named fsid and fsid-base and 'isi_name', a data.frame to use with the `demographics.to.qdec.table.dat` function.

See Also

The function `demographics.to.qdec.table.dat` to write the result to a QDEC file.

Examples

```
dem = data.frame("ID"=paste("subject", seq(5), sep=""),
  "age"=sample.int(20, 5)+10L, "isi"=rnorm(5, 2.0, 0.1)); #sample data.
qdec.table.skeleton(dem$ID, dem$isi);
```

`ras2vox_tkr` *The FreeSurfer default ras2vox_tkr matrix.*

Description

Applying this matrix to a FreeSurfer surface RAS coordinate (from a surface file like 'lh.white') will give you the voxel index (CRS) in a conformed FreeSurfer volume. The returned matrix is the inverse of the 'vox2ras_tkr' matrix.

Usage

```
ras2vox_tkr()
```

Value

numeric 4x4 matrix, the FreeSurfer ras2vox_tkr matrix.

See Also

Other surface and volume coordinates: `vox2ras_tkr()`

Examples

```
 # Compute the FreeSurfer CRS voxel index of surface RAS coordinate (0.0, 0.0, 0.0):
 ras2vox_tkr() %*% c(0, 0, 0, 1);
# Show that the voxel at surface RAS corrds (0.0, 0.0, 0.0) is the one with CRS (128, 128, 128):
 ras2vox_tkr() %*% c(0.0, 0.0, 0.0, 1);
```

`read.colorcsv` *Read colors from CSV file.*

Description

Read colors from CSV file.

Usage

```
read.colorcsv(filepath)
```

Arguments

`filepath`	character string, path to a CSV file containing colors

Value

vector of hex color strings

`read.md.demographics` *Read demographics file*

Description

Load a list of subjects and metadata from a demographics file, i.e., a tab-separated file containing an arbitrary number of columns, one of which must be the subject id.

Usage

```
read.md.demographics(
  demographics_file,
  column_names = NULL,
  header = FALSE,
  scale_and_center = FALSE,
  sep = "",
  report = FALSE,
  stringsAsFactors = TRUE,
  group_column_name = NULL
)
```

Arguments

`demographics_file,`	string. The path to the file.
`column_names,`	vector of strings. The column names to set in the returned dataframe. The length must match the number of columns in the file.
`header,`	logical. Whether the file starts with a header line.
`scale_and_center,`	logical. Whether to center and scale the data. Defaults to FALSE.
`sep,`	string. Separator passed to `read.table`, defaults to tabulator.
`report,`	logical. Whether to write an overview, i.e., some descriptive statistics for each column, to STDOUT. Defaults to FALSE. See `report.on.demographics`.
`stringsAsFactors,`	logical. Whether to convert strings in the input data to factors. Defaults to TRUE.
`group_column_name,`	string or NULL. If given, the column name of the group column. It must be a factor column with 2 levels. Enables group-comparison tests. Defaults to NULL.

Value

a dataframe. The data in the file. String columns will be returned as factors, which you may want to adapt afterwards for the subject identifier column.

See Also

Other metadata functions: `demographics.to.fsgd.file()`, `read.md.subjects()`, `report.on.demographics()`

Examples

```
demographics_file =
system.file("extdata", "demographics.tsv", package = "fsbrain", mustWork = TRUE);
column_names = c("subject_id", "group", "age");
demographics = read.md.demographics(demographics_file,
header = TRUE, column_names = column_names, report = FALSE);
```

`read.md.subjects` *Read subjects file*

Description

Load a list of subjects from a subjects file, i.e., a simple text file containing one subject name per line.

Usage

```
read.md.subjects(subjects_file, header)
```

Arguments

`subjects_file`	character string, the path to the subjects file.
`header`	logical, whether the file starts with a header line.

Value

vector of strings, the subject identifiers.

See Also

Other metadata functions: `demographics.to.fsgd.file()`, `read.md.demographics()`, `report.on.demographics()`

Examples

```
subjects_file = system.file("extdata", "subjects.txt", package = "fsbrain", mustWork = TRUE);
 subjects_list = read.md.subjects(subjects_file, header = FALSE);
```

`read.md.subjects.from.fsgd`

Read subjects list from an FSGD file.

Description

Read subjects list from an FSGD file.

Usage

```
read.md.subjects.from.fsgd(filepath)
```

Arguments

`filepath`	character string, path to a FreeSurfer Group Descriptor (FSGD) file.

Value

vector of character strings, the subject identifiers

Note

This is not a parser for all data in an FSGD file.

See Also

`demographics.to.fsgd.file`

`regions.to.ignore` *Give suggestions for regions to ignore for an atlas.*

Description

Give suggestions for regions to ignore for an atlas. These are regions for which many subjects do not have any vertices in them, or the Medial Wall and Unknown regions.

Usage

```
regions.to.ignore(atlas)
```

Arguments

`atlas,`	string. The name of an atlas. Supported strings are 'aparc' and 'aparc.a2009s'.

Value

vector of strings, the region names.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
aparc_regions_ign = regions.to.ignore('aparc');
aparc_a2009s_regions_ign = regions.to.ignore('aparc.a2009s');
```

`report.on.demographics` *Print a demographics report*

Description

Print a demographics report

Usage

```
report.on.demographics(
  demographics_df,
  group_column_name = NULL,
  paired = FALSE
)
```

Arguments

`demographics_df`	a demographics data.frame, as returned by `read.md.demographics`.
`group_column_name,`	string or NULL. If given, the column name of the group column. It must be a factor column with 2 levels. Enables group-comparison tests. Defaults to ‘NULL‘.
`paired`	Whether the data of the two groups if paired (repeated measurements). Only relevant if group_column_name is given and tests for group differences are included in the report. Defaults to ‘FALSE‘.

Value

vector of character strings, the lines of the demographics report.

See Also

Other metadata functions: `demographics.to.fsgd.file()`, `read.md.demographics()`, `read.md.subjects()`

`rglactions` *Create rglactions list, suitable to be passed as parameter to vis functions.*

Description

Create rglactions list, suitable to be passed as parameter to vis functions.

Usage

```
rglactions()
```

Value

named list, an example ‘rlgactions‘ instance that will save a screenshot of the plot produced by the vis function in the current working directory (see `getwd`), under the name ’fsbrain_out.png’.

Note

List of all available rglactions: (1) ‘snapshot_png=filepath‘ takes a screenshot in PNG format and saves it in at ‘filepath‘. (2) ‘trans_fun=function‘ uses the transformation function trans_fun to the data before mapping data values to colors and plotting. Popular transformation functions are `limit_fun`, `limit_fun_na`, and `clip_fun`. (3) ‘text=arglist‘ calls `text3d` with the given args after plotting. (4) ‘snapshot_vec=filepath‘ takes a screenshot in vector format and saves it in at ‘filepath‘. You also need to set the format via ‘snapshot_vec_format‘, valid entries are one of "ps", "eps", "tex", "pdf", "svg", "pgf" (default is ’eps’). This is experimental and may take a while.

Examples

```
rgla_screenie = list('snapshot_png'='fsbain_out.png');
rgla_screenie = rglactions();   # same as above
rgla_vec_scr = list('snapshot_vec'="~/fsbrain.pdf",
  "snapshot_vec_format"="pdf");
rgla_clamp = list('trans_fun'=clip.data); # old style
rgla_clamp = list('trans_fun'=clip_fun(0.05, 0.95)); # new style
rgla_clamp = list('trans_fun'=clip_fun());            # equivalent.
rgla_limit = list('trans_fun'=limit_fun(2,5));
rgla_ls = list('trans_fun'=limit_fun_na(2,5), 'snapshot_png'='~/fig1.png');
```

`rglo` *Get rgloptions and consider global options.*

Description

This function retrieves the global rgloptions defined in `getOption('fsbrain.rgloptions')`, or, if this is not set, returns the value from `rglot`.

Usage

```
rglo()
```

Value

named list, usable as 'rgloptions' parameter for vis functions like `vis.subject.morph.native`.

Note

You can set the default size for all fsbrain figures to 1200x1200 pixels like this: `options("fsbrain.rgloptions"=list("wi`

`rglot` *Get rgloptions for testing.*

Description

This function defines the figure size that is used during the unit tests. Currently `list('windowRect' = c(50, 50, 800, 800)`.

Usage

```
rglot()
```

Value

named list, usable as 'rgloptions' parameter for vis functions like `vis.subject.morph.native`.

`rglvoxels` *Draw 3D boxes at locations using rgl.*

Description

Draw 3D boxes at all given coordinates using rgl, analogous to `spheres3d`. Constructs the coordinates for triangles making up the boxes, then uses `triangles3d` to render them.

Usage

```
rglvoxels(centers, r = 1, voxelcol = NULL, do_show = TRUE, ...)
```

Arguments

`centers`	numerical matrix with 3 columns. Each column represents the x, y, z coordinates of a center at which to create a cube.
`r`	numerical vector or scalar, the cube edge length. This is the length of the axis-parallel edges of the cube. The vector must have length 1 (same edge length for all cubes), or the length must be identical to the number of rows in parameter ‘centers‘.
`voxelcol`	vector of rgb color strings for the individual voxels. Its length must be identical to `nrow(centers)` if given.
`do_show`	logical, whether to visualize the result in the current rgl scene
`...`	material properties, passed to `triangles3d`. Example: `color = "#0000ff", lit=FALSE`.

Value

list of ‘fs.coloredvoxels‘ instances, invisible. The function is called for the side effect of visualizing the data, and usually you can ignore the return value.

Examples

```
## Not run:
   # Plot a 3D cloud of 500 red voxels:
   centers = matrix(rnorm(500*3)*100, ncol=3);
   rglvoxels(centers, voxelcol="red");

## End(Not run)
```

`scale01` *Scale given values to range 0..1.*

Description

Scale given values to range 0..1.

Usage

```
scale01(x, ...)
```

Arguments

`x`	the numeric data
`...`	the numeric data

Value

the scaled data

`shape.descriptor.names` *Get all shape descriptor names.*

Description

Get all shape descriptor names.

Usage

```
shape.descriptor.names()
```

Value

vector of character strings, the names

`shape.descriptors` *Computes geometric curvature-based descriptors.*

Description

Computes geometric curvature-based descriptors.

Usage

```
shape.descriptors(pc, descriptors = shape.descriptor.names())
```

Arguments

`pc`	a 'principal_curvatures' data list, see `principal.curvatures` for details.
`descriptors`	vector of character strings, the descriptors you want. See `shape.descriptor.names` for all available names.

Value

dataframe of descriptor values, each columns contains one descriptor.

References

Shimony et al. (2016). Comparison of cortical folding measures for evaluation of developing human brain. Neuroimage, 125, 780-790.

`shift.hemis.apart` *Shift hemispheres apart.*

Description

Modify mesh coordinates of a hemilist of coloredmeshes to introduce a gap between the two hemispheres.

Usage

```
shift.hemis.apart(
  coloredmeshes_hl,
  shift_by = NULL,
  axis = 1L,
  hemi_order_on_axis = "lr",
  min_dist = 0
)
```

Arguments

`coloredmeshes_hl`	hemilist of coloredmeshes
`shift_by`	numerical vector of length 2, the amount by which to shift the hemis. The first value is for the left hemi, the second for the right hemi (values can be negative). Pass ‘NULL‘ to determine the shift automatically from the mesh coordinates, and adapt ’hemi_order_on_axis’ to define how that happens.
`axis`	positive integer, one of 1L, 2L or 3L. The axis on which to shift (x,y,z).
`hemi_order_on_axis`	character string, one of ’auto’, ’auto_flipped’, ’lr’ or ’rl’. Defines how to determine the order of the hemis on the axes. This is ignored unless ’shift_by’ is ‘NULL‘. The ’auto’ setting assumes that the hemisphere with the smaller minimal vertex coordinate (on the given axis) comes first. Note that if the overlap (or shift) is extreme, this may not hold anymore. Therefore, the default value is ’lr’, which works for FreeSurfer data. The ’auto_flipped’ setting will always return the inverse of ’auto’, so if ’auto’ did not work, ’auto_flipped’ will.
`min_dist`	numerical scalar, the minimal distance of the hemispheres. Ignored unless ’shift_by’ is ‘NULL‘.

Value

hemilist of coloredmeshes, the shifted meshes

`sjd.demo` *Download optional demo data if needed and return its path.*

Description

This is a wrapper around `download_optional_data()` and `get_optional_data_filepath("subjects_dir")`. It will download the optional fsbrain demo data unless it already exists locally.

Usage

```
sjd.demo(accept_freesurfer_license = FALSE)
```

Arguments

`accept_freesurfer_license`	logical, whether you want to also download fsaverage and fsaverage3, and accept the FreeSurfer license for fsaverage and fsaverage3, available at https://surfer.nmr.mgh.harvard.edu/fs Defaults to FALSE. If FALSE, only the demo data from fsbrain itself (’subject1’) will be downloaded.

Value

character string, the path to the ’subjects_dir’ directory within the downloaded optional data directory.

Note

This function will stop if the data cannot be accessed, i.e., the 'subjects_dir' does not exist after trying to download the data.

`sjld` *Get subjects list from subjects.txt file in dir.*

Description

Get subjects list from subjects.txt file in dir.

Usage

```
sjld(subjects_dir)
```

Arguments

`subjects_dir`	character string, existing subjects dir with a subjects.txt file containing one subject per line and no header line.

Value

named list with entries: 'd', the query subjects_dir (repeated from the parameter), 'l', vector of character strings, the subjects_list read from the file, 'f', the subjects_file.

Note

This function stops if the file does not exist or cannot be read.

`spread.values.over.annot`
Spread a single value for a region to all region vertices.

Description

Given an annotation and a list of values (one per brain region), return data that has the values for each region mapped to all region vertices.

Usage

```
spread.values.over.annot(
  annot,
  region_value_list,
  value_for_unlisted_regions = NaN,
  warn_on_unmatched_list_regions = FALSE,
  warn_on_unmatched_atlas_regions = FALSE
)
```

Arguments

`annot,`	annotation. The result of calling fs.read.annot.
`region_value_list,`	named list of strings. Each name must be a region name from the annotation, and the value must be the value to spread to all region vertices.
`value_for_unlisted_regions,`	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.
`warn_on_unmatched_list_regions,`	logical. Whether to print a warning when a region occurs in the region_value_list that is not part of the given atlas (and the value assigned to this region is thus ignored in the output file and data). Defaults to FALSE.
`warn_on_unmatched_atlas_regions,`	logical. Whether to print a warning when a region occurs in the atlas that is not part of the given region_value_list (and thus the vertices of the region will be assigned the value 'value_for_unlisted_regions' in the output file and data). Defaults to FALSE.

Value

named list with following entries: "spread_data": a vector of length n, where n is the number of vertices in the annotation. One could write this to an MGH or curv file for visualization. "regions_not_in_annot": list of regions which are not in the annotation, but in the region_value_list. Their values were ignored.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.hemi()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
   region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
   morph_like_data =
   spread.values.over.annot(annot, region_value_list, value_for_unlisted_regions=0.0);

## End(Not run)
```

`spread.values.over.hemi` *Spread the values in the region_value_list and return them for one hemisphere.*

Description

Given an atlas and a list that contains one value for each atlas region, create morphometry data in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

Usage

```
spread.values.over.hemi(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region_value_list,
  value_for_unlisted_regions = NA,
  silent = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id,`	string. The subject identifier
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`region_value_list,`	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list or vector, but then the length must exactly match the number of regions in the atlas, and the order must match the annotation file of the subject and hemisphere. Use with care, and keep in mind that some subjects do not have all regions.
`value_for_unlisted_regions,`	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.
`silent`	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).

Value

numeric vector containing the data.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.subject()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
   morph_like_data =
   spread.values.over.hemi(subjects_dir, 'subject1', 'lh', 'aparc', region_value_list);

## End(Not run)
```

---

spread.values.over.subject

Spread the values in the region_value_list and return them for one hemisphere.

Description

Given an atlas and a list that contains one value for each atlas region, create morphometry data in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

Usage

```
spread.values.over.subject(
  subjects_dir,
  subject_id,
  atlas,
  lh_region_value_list,
  rh_region_value_list,
  value_for_unlisted_regions = NaN,
  silent = FALSE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id,`	string. The subject identifier

`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`lh_region_value_list,`	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the left hemisphere.
`rh_region_value_list,`	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the right hemisphere.
`value_for_unlisted_regions,`	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.
`silent`	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).

Value

named list with entries 'lh' and 'rh'. Each value is a numeric vector containing the data for the respective hemisphere.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
   rh_region_value_list = list("bankssts"=0.5);
   morph_like_data =
   spread.values.over.subject(subjects_dir, 'subject1', 'aparc',
   lh_region_value_list, rh_region_value_list);

## End(Not run)
```

`subject.annot` *Load an annotation for a subject.*

Description

Load a brain surface annotation, i.e., a cortical parcellation based on an atlas, for a subject.

Usage

```
subject.annot(subjects_dir, subject_id, hemi, atlas)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`atlas`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.

Value

the annotation, as returned by `read.fs.annot`. It is a named list, enties are: "vertices" vector of n vertex indices, starting with 0. "label_codes": vector of n integers, each entry is a color code, i.e., a value from the 5th column in the table structure included in the "colortable" entry (see below). "label_names": the n brain structure names for the vertices, already retrieved from the colortable using the code. "hex_colors_rgb": Vector of hex color for each vertex. The "colortable" is another named list with 3 entries: "num_entries": int, number of brain structures. "struct_names": vector of strings, the brain structure names. "table": numeric matrix with num_entries rows and 5 colums. The 5 columns are: 1 = color red channel, 2=color blue channel, 3=color green channel, 4=color alpha channel, 5=unique color code. "colortable_df": The same information as a dataframe. Contains the extra columns "hex_color_string_rgb" and "hex_color_string_rgba" that hold the color as an RGB(A) hex string, like "#rrggbbaa".

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject`, `subject.atlas.agg()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   annot_lh = subject.annot(subjects_dir, "subject1", "lh", "aparc");

## End(Not run)
```

`subject.annot.border` *Compute annot border vertices.*

Description

Compute annot border vertices.

Usage

```
subject.annot.border(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  surface = "white",
  expand_inwards = 0L,
  limit_to_regions = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`atlas`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`expand_inwards`	integer, additional thickness of the borders. Increases computation time, defaults to 0L.
`limit_to_regions`	vector of character strings or NULL, a list of regions for which to draw the outline (see `get.atlas.region.names`). If NULL, all regions will be used. If (and only if) this parameter is used, the 'outline_color' parameter can be a vector of color strings, one color per region.

Value

hemilist of integer vectors, the vertices in the border

`subject.atlas.agg` *Aggregate morphometry data over brain atlas regions for a subject.*

Description

Aggregate morphometry data over brain atlas regions, e.g., compute the mean thickness value over all regions in an atlas.

Usage

```
subject.atlas.agg(
  vertex_morph_data,
  vertex_label_names,
  agg_fun = base::mean,
  requested_label_names = c()
)
```

Arguments

`vertex_morph_data,`	numeric vector. The morphometry data, one value per vertex. The morphometry data are typically loaded from an MGZ or curv format file with the read.fs.curv or read.fs.mgh functions.
`vertex_label_names,`	string vector. The region names for the vertices, one string per vertex. The region names are typically loaded from an annotation file with the read.fs.annot function.
`agg_fun,`	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to `base::mean`.
`requested_label_names,`	string vector. The label (or region) names that you want to occur in the output. If not specified, all region names which occur in the data are used. If given, and one of the requested names does NOT occur in the data, it will occur in the output with aggregation value NaN. If given, and one of the names from the data does NOT occur in the requested list, it will NOT occur in the output. So if you specify this, the output dataframe will contain a row for a region if and only if it is in the requested list.

Value

dataframe with aggregated values for all regions, with 2 columns and n rows, where n is the number of effective regions. The columns are: "region": string, contains the region name. "aggregated": numeric, contains the result of applying agg_fun to the morphometry data in that region.

See Also

Other aggregation functions: `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.morph.agg.standard`

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject` `subject.annot()`, `subject.label.from.annot()`, `subject.lobes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   morph_data = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
   annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
   agg = subject.atlas.agg(morph_data, annot$label_names);

## End(Not run)
```

`subject.filepath.morph.native`

Construct filepath of native space morphometry data file.

Description

Construct filepath of native space morphometry data file.

Usage

```
subject.filepath.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  format = "curv",
  warn_if_nonexistent = FALSE,
  error_if_nonexistent = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

`hemi`	string, one of 'lh' or 'rh'. The hemisphere name.
`format,`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.
`warn_if_nonexistent`	logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.
`error_if_nonexistent`	logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

Value

string, the file path.

`subject.filepath.morph.standard` *Construct filepath of standard space morphometry data file.*

Description

Construct filepath of standard space morphometry data file.

Usage

```
subject.filepath.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "auto",
  warn_if_nonexistent = FALSE,
  error_if_nonexistent = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier. Can be a vector.
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi`	string, one of 'lh' or 'rh'. The hemisphere name.
`fwhm`	string. Smoothing as string, e.g. '10' or '25'. Defaults to '10'.
`template_subject`	string. Template subject name, defaults to 'fsaverage'.

`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`warn_if_nonexistent`	logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.
`error_if_nonexistent`	logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

Value

string, the file path. (Or a vector if 'subject_id' is a vector.)

`subject.label` *Retrieve label data for a single subject.*

Description

Load a label (like 'label/lh.cortex.label') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  return_one_based_indices = TRUE,
  full = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`label`	string. Name of the label file, without the hemi part. You can include the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'. You can also pass just the label (e.g., 'cortex'): if the string does not end with the suffix '.label', that suffix gets added auomatically.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded. For 'both', see the information on the return value.
`return_one_based_indices`	logical. Whether the indices should be 1-based. Indices are stored zero-based in the file, but R uses 1-based indices. Defaults to TRUE, which means that 1 will be added to all indices read from the file before returning them.

`full`	logical, whether to return the full label structure instead of only the vertex indices.

Value

integer vector with label data: the list of vertex indices in the label. See 'return_one_based_indices' for important information. If parameter ‘hemi‘ is set to ‘both‘, a named list with entries ‘lh‘ and ‘rh‘ is returned, and the values of are the respective labels.

See Also

Other label data functions: `group.label()`, `labeldata.from.mask()`, `mask.from.labeldata.for.hemi()`

Examples

```
## Not run: 
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   cortex_lh = subject.label(subjects_dir, "subject1", "cortex.label", "lh");

## End(Not run)
```

---

subject.label.from.annot

Extract a region from an atlas annotation as a label for a subject.

Description

The returned label can be used to mask morphometry data, e.g., to set the values of a certain region to NaN or to extract only values from a certain region.

Usage

```
subject.label.from.annot(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region,
  return_one_based_indices = TRUE,
  invert = FALSE,
  error_on_invalid_region = TRUE
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id,`	string. The subject identifier.
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`region,`	string. A valid region name for the annotation, i.e., one of the regions of the atlas.
`return_one_based_indices,`	logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.
`invert,`	logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
`error_on_invalid_region,`	logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

Value

integer vector with label data: the list of vertex indices in the label.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject`
`subject.annot()`, `subject.atlas.agg()`, `subject.lobes()`

`subject.lobes` *Load labels representing brain lobes.*

Description

This gives you labels that represent brain lobes for a subject. The lobe definition is based on the Desikan-Killiany atlas (Desikan *et al.*, 2010) as suggested on the FreeSurfer website at https://surfer.nmr.mgh.harvard.edu/fswiki/CorticalParcellation.

Usage

```
subject.lobes(
  subjects_dir,
  subject_id,
  hemi = "both",
  include_cingulate = TRUE,
  as_annot = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
`include_cingulate`	logical, whether to include the vertices of the cingulate in the lobes
`as_annot`	return a hemilist of annotations instead of the return value described in the *value* section

Value

hemilist of integer vectors, the vectors represent vertex indices of the hemispheres, and each vertex is assigned one of the following values: '0'=no_lobe, '1'=frontal, '2'=parietal, '3'=temporal, '4'=occipital.

See Also

Other atlas functions: `get.atlas.region.names()`, `group.agg.atlas.native()`, `group.agg.atlas.standard()`, `group.annot()`, `group.label.from.annot()`, `label.from.annotdata()`, `label.to.annot()`, `regions.to.ignore()`, `spread.values.over.annot()`, `spread.values.over.hemi()`, `spread.values.over.subject`
`subject.annot()`, `subject.atlas.agg()`, `subject.label.from.annot()`

Other label functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `subject.mask()`, `vis.labeldata.on.subject()`, `vis.subject.label()`

`subject.mask` *Compute a mask for a subject.*

Description

Compute a binary vertex mask for the surface vertices of a subject. By defaults, the medial wall is masked.

Usage

```
subject.mask(
  subjects_dir,
  subject_id,
  hemi = "both",
  from_label = "cortex",
  surf_num_verts = "white",
  invert_mask = TRUE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`from_label`	string, the label file to use. Defaults to 'cortex', which will result in a mask of the medial wall versus cortex vertices.
`surf_num_verts`	string or integer. If an integer, interpreted as the number of vertices in the respective surface (lh or rh). If a character string, interpreted as a surface name, (e.g.,'white' or 'pial'), and the respective surface will be loaded to determine the number of vertices in it. If parameter 'hemi' is set to 'both' and you supply the vertex count as an integer, this can be a vector of length 2 if the surfaces have different vertex counts (the first entry for 'lh', the second for 'rh').
`invert_mask`	logical, whether to invert the mask. E.g., when the mask is loaded from the cortex labels, if this is set to FALSE, the cortex would be masked (set to 0 in the final mask). If you want **everything but the cortex** to be masked (set to 0), you should set this to 'TRUE'. Defaults to 'TRUE'.

Value

the mask, a logical vector with the length of the vertices in the surface. If parameter 'hemi' is set to 'both', a named list with entries 'lh' and 'rh' is returned, and the values of are the respective masks.

See Also

Other label functions: `apply.label.to.morphdata`(), `apply.labeldata.to.morphdata`(), `subject.lobes`(), `vis.labeldata.on.subject`(), `vis.subject.label`()

Examples

```
## Not run:
   # Generate a binary mask of the medial wall. Wall vertices will
   #  be set to 0, cortex vertices will be set to 1.
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   mask = subject.mask(subjects_dir, "subject1");
```

```
    # Print some information on the mask:
    #cat(sprintf("lh: %d verts, %d in cortex, %d medial wall.\n", length(mask$lh),
    # sum(mask$lh), (length(mask$lh)- sum(mask$lh))))
    # Output: lh: 149244 verts, 140891 in cortex, 8353 medial wall.
    # Now visualize the mask to illustrate that it is correct:
    vis.mask.on.subject(subjects_dir, "subject1", mask$lh, mask$rh);

## End(Not run)
```

`subject.morph.native` *Retrieve native space morphometry data for a single subject.*

Description

Load native space morphometry data (like 'surf/lh.area') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
subject.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  format = "curv",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
`split_by_hemi`	logical, whether the returned data should be encapsulated in a named list, where the names are from 'lh' and 'rh', and the values are the respective data.

Value

vector with native space morph data, as returned by `read.fs.morph`.

See Also

Other morphometry data functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `group.morph.native()`, `group.morph.standard()`, `subject.morph.standard()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

   # Load the full data:
   thickness_lh = subject.morph.native(subjects_dir, "subject1", "thickness", "lh");
   mean(thickness_lh);  # prints 2.437466

   # Load the data again, but this time exclude the medial wall:
   thickness_lh_cortex = subject.morph.native(subjects_dir, "subject1", "thickness",
    "lh", cortex_only=TRUE);
   mean(thickness_lh_cortex, na.rm=TRUE);     # prints 2.544132
   vis.data.on.subject(subjects_dir, "subject1", thickness_lh_cortex, NULL);

## End(Not run)
```

`subject.morph.standard`

Retrieve standard space morphometry data for a single subject.

Description

Load standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
subject.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`measure`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm`	string. Smoothing as string, e.g. '10' or '25'.
`template_subject`	string. Template subject name, defaults to 'fsaverage'.
`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the template subject. Defaults to FALSE.
`split_by_hemi`	logical, whether the returned data should be encapsulated in a named list, where the names are from 'lh' and 'rh', and the values are the respective data.

Value

vector with standard space morph data

See Also

Other morphometry data functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `group.morph.native()`, `group.morph.standard()`, `subject.morph.native()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
  thickness_lh = subject.morph.standard(subjects_dir, "subject1", "thickness", "lh", fwhm='10');

## End(Not run)
```

`subject.num.verts` *Get subjects vertex count.*

Description

Determine vertex counts for the brain meshes of a subject.

Usage

```
subject.num.verts(
  subjects_dir,
  subject_id,
  surface = "white",
  hemi = "both",
  do_sum = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`surface`	string. The surface name. E.g., "white", or "pial". Used to construct the name of the surface file to be loaded.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
`do_sum`	logical, whether to return the sum of the vertex counts for lh and rh. Ignored unless 'hemi' is 'both'. If set, a single scalar will be returned.

Value

hemilist of integers, the vertex count. If hemi is 'both' and 'do_sum' is ‘FALSE‘, a hemilist of integers is returned. Otherwise, a single integer.

`subject.surface` *Load a surface for a subject.*

Description

Load a brain surface mesh for a subject.

Usage

```
subject.surface(
  subjects_dir,
  subject_id,
  surface = "white",
  hemi = "both",
  force_hemilist = FALSE,
  as_tm = FALSE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`surface`	string. The surface name. E.g., "white", or "pial". Used to construct the name of the surface file to be loaded.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
`force_hemilist`	logical, whether to return a hemilist even if the 'hemi' parameter is not set to 'both'
`as_tm`	logical, whether to return an `rgl::tmesh3d` instead of an `fs.surface` instance by applying the `fs.surface.to.tmesh3d` function.

Value

the ‘fs.surface‘ instance, as returned by `read.fs.surface`. If parameter ‘hemi‘ is set to ‘both‘, a named list with entries ‘lh‘ and ‘rh‘ is returned, and the values of are the respective surfaces. The mesh data structure used in ‘fs.surface‘ is a *face index set*.

See Also

Other surface mesh functions: `face.edges()`, `label.border()`, `mesh.vertex.included.faces()`, `mesh.vertex.neighbors()`, `vis.path.along.verts()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   lh_white = subject.surface(subjects_dir, "subject1", "white", "lh");

## End(Not run)
```

`subject.volume` *Read a brain volume.*

Description

Load a brain volume (like ‘mri/brain.mgz‘) for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
subject.volume(
  subjects_dir,
  subject_id,
  volume,
  format = "auto",
  drop_empty_dims = TRUE,
  with_header = FALSE,
  mri_subdir = NULL
)
```

Arguments

`subjects_dir`	character string, the FreeSurfer ‘SUBJECTS_DIR‘, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	character string, the subject identifier.
`volume`	character string, name of the volume file without file extension. Examples: ‘brain‘ or ‘aseg‘.
`format`	string. One of ’mgh’, ’mgz’, ’AUTO’. If left at the default value ’AUTO’, the function will look for files with extensions ’mgh’ and ’mgz’ (in that order) and use the first one that exists.
`drop_empty_dims`	logical, whether to drop empty dimensions of the returned data. Passed to `read.fs.mgh`.
`with_header`	logical. Whether to return the header as well. If TRUE, return a named list with entries "data" and "header". The latter is another named list which contains the header data. These header entries exist: "dtype": int, one of: 0=MRI_UCHAR; 1=MRI_INT; 3=MRI_FLOAT; 4=MRI_SHORT. "voldim": integer vector. The volume (=data) dimensions. E.g., c(256, 256, 256, 1). These header entries may exist: "vox2ras_matrix" (exists if "ras_good_flag" is 1), "mr_params" (exists if "has_mr_params" is 1). Passed to `read.fs.mgh`.
`mri_subdir`	character string or NULL, the subdir to use within the ‘mri‘ directory. Defaults to ‘NULL‘, which means to read directly from the ‘mri‘ dir. You could use this to read volumes from the ‘mri/orig/‘ directory by setting it to ‘orig‘.

Value

numerical array, the voxel data. If ‘with_header‘, the full volume datastructure (see above).

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   brain = subject.volume(subjects_dir, 'subject1', 'brain', with_header = TRUE);
  # Use the vox2ras matrix from the header to compute RAS coordinates at CRS voxel (0, 0, 0):
   brain$header$vox2ras_matrix %*% c(0,0,0,1);

## End(Not run)
```

`surface.curvatures` *Compute the k1 and k2 principal curvatures of a mesh.*

Description

Compute the k1 and k2 principal curvatures of a mesh.

Usage

```
surface.curvatures(surface)
```

Arguments

`surface`	an fs.surface instance, as returned by `subject.surface`.

Value

named list, the entries 'K1' and 'K2' contain the principal curvatures.

Note

Require the optional dependency package 'Rvcg'.

`tmesh3d.to.fs.surface` *Get an fs.surface brain mesh from an rgl tmesh3d instance.*

Description

Get an fs.surface brain mesh from an rgl tmesh3d instance.

Usage

```
tmesh3d.to.fs.surface(tmesh)
```

Arguments

`tmesh`	a tmesh3d instance, see `rgl::tmesh3d` for details.

Value

an fs.surface instance, as returned by `subject.surface` or `freesurferformats::read.fs.surface`.

`vdata.split.by.hemi` *Split morph data vector at hemisphere boundary.*

Description

Given a single vector of per-vertex data for a mesh, split it at the hemi boundary. This is achieved by loading the respective surface and checking the number of vertices for the 2 hemispheres.

Usage

```
vdata.split.by.hemi(
  subjects_dir,
  subject_id,
  vdata,
  surface = "white",
  expand = TRUE
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier
`vdata`	numerical vector of data for both hemispheres, one value per vertex
`surface`	the surface to load to determine the vertex counts
`expand`	logical, whether to allow input of length 1, and expand (repeat) it to the length of the hemispheres.

Value

a hemilist, each entry contains the data part of the respective hemisphere.

Note

Instead of calling this function to split the data, you could use the 'split_by_hemi' parameter of `subject.morph.native`.

`vertex.coords` *Return coordinates for vertices, supporting entire brain via hemilist.*

Description

Return coordinates for vertices, supporting entire brain via hemilist.

Usage

```
vertex.coords(surface, vertices)
```

Arguments

`surface`	an fs.surface instance, see `subject.surface` function. Can also be a hemilist of surfaces, in which case the vertices must be indices over both meshes (in range `1..(nv(lh)+nv(rh))`). If a hemilist, both entries must be surfaces (non-NULL).
`vertices`	vector of positive integers, the vertex indices. Values which are outside of the valid indices for the surface will be silently ignored, making it easier to work with the two hemispheres.

Value

double nx3 matrix of vertex coordinates.

See Also

Other 3d utility functions: `highlight.points.spheres()`, `highlight.vertices.spheres()`

`vertex.hemis` *Return the proper hemi string ('lh' or 'rh') for each vertex.*

Description

Return the proper hemi string ('lh' or 'rh') for each vertex.

Usage

```
vertex.hemis(surface, vertices)
```

Arguments

`surface`	hemilist of surfaces or a single integer which will be interpreted as the vertex count of the left hemisphere.
`vertices`	vector of positive integers, the query vertex indices. Can be in range `1..(nv(lh)+nv(rh))`, i.e., across the whole brain.

Value

vector of character strings, each string is 'lh' or 'rh'.

Note

It is not checked in any way whether the vertex indices are out of bounds on the upper side (higher than the highest rh vertex index).

Examples

```
vertex.hemis(100L, vertices=c(99L, 100L, 101L));
```

`vis.color.on.subject` *Visualize pre-defined vertex colors on a subject.*

Description

Visualize pre-defined vertex colors on a subject.

Usage

```
vis.color.on.subject(
  subjects_dir,
  vis_subject_id,
  color_lh = NULL,
  color_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  color_both = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`color_lh`	vector of colors to visualize on the left hemisphere surface. Length must match number of vertices in hemi surface, or be a single color.
`color_rh`	vector of colors to visualize on the right hemisphere surface. Length must match number of vertices in hemi surface, or be a single color.

`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`color_both`	vector of colors to visualize on the left and right hemispheres. Alternative to 'color_lh' and 'color_rh'. Length must match sum of vertices in both hemis. Can also be a hemilist.
`style`	character string or rgl rendering style, see `get.rglstyle`.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other surface visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   color_lh = '#ff0000';
   num_verts_subject1_rh = 153333;
   color_rh = rep('#333333', num_verts_subject1_rh);
   color_rh[1:30000] = '#00ff00';
   color_rh[30001:60000] = '#ff0000';
   color_rh[60001:90000] = '#0000ff';
   color_rh[90001:120000] = '#ffff00';
   color_rh[120001:150000] = '#00ffff';
   vis.color.on.subject(subjects_dir, 'subject1', color_lh, color_rh);

## End(Not run)
```

`vis.coloredmeshes` *Visualize a list of colored meshes in a single scene.*

Description

Visualize a list of colored meshes in a single scene.

Usage

```
vis.coloredmeshes(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE
)
```

Arguments

`coloredmeshes`	list of coloredmesh. A coloredmesh is a named list as returned by the coloredmesh.from.* functions. It has the entries 'mesh' of type tmesh3d, a 'col', which is a color specification for such a mesh.
`background`	string, background color passed to rgl::bg3d()
`skip_all_na`	logical, whether to skip (i.e., not render) meshes in the list that have the property 'render' set to FALSE. Defaults to TRUE. Practically, this means that a hemisphere for which the data was not given is not rendered, instead of being rendered in a single color.
`style`	a named list of style parameters or a string specifying an available style by name (e.g., 'shiny'). Defaults to 'default', the default style.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000));`
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. Defaults to the empty list.
`draw_colorbar`	logical. Whether to draw a colorbar. WARNING: Will only show up if there is enough space in the plot area and does not resize properly. Defaults to FALSE. See `coloredmesh.plot.colorbar.separate` for an alternative.

Value

the list of visualized coloredmeshes

Note

To change or adapt the colorbar, you should use the makecmap_options parameter when constructing them in a vis function. See the example.

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   cm = vis.subject.morph.native(subjects_dir, 'subject1', 'thickness',
    makecmap_options=list('n'=100, 'colFn'=viridis::viridis));
   # You could mess with the meshes here.
   vis.coloredmeshes(cm);

## End(Not run)
```

`vis.coloredmeshes.rotating` *Visualize a list of colored meshes in a single scene and rotate them, movie-style.*

Description

Visualize a list of colored meshes in a single scene and rotate them, movie-style.

Usage

```
vis.coloredmeshes.rotating(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  x = 0,
  y = 0,
  z = 1,
  rpm = 6,
  duration = 10,
  rgloptions = rglo(),
  rglactions = list()
)
```

Arguments

`coloredmeshes`	list of coloredmesh. A coloredmesh is a named list as returned by the coloredmesh.from.* functions. It has the entries 'mesh' of type tmesh3d, a 'col', which is a color specification for such a mesh.
`background`	string, background color passed to rgl::bg3d()
`skip_all_na`	logical, whether to skip (i.e., not render) meshes in the list that have the property 'rendner' set to FALSE. Defaults to TRUE. Practically, this means that a hemisphere for which the data was not given is not rendered, instead of being rendered in a single color.

`style`	a named list of style parameters or a string specifying an available style by name (e.g., 'shiny'). Defaults to 'default', the default style.
`x`	rotation x axis value, passed to `spin3d`. Defaults to 0.
`y`	rotation y axis value, passed to `spin3d`. Defaults to 1.
`z`	rotation z axis value, passed to `spin3d`. Defaults to 0.
`rpm`	rotation rpm value, passed to `spin3d`. Defaults to 15.
`duration`	rotation duration value, passed to `spin3d`. Defaults to 20.
`rgloptions`	option list passed to `par3d`. Example: rgloptions = list("windowRect"=c(50,50,1000,1000));
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. Defaults to the empty list.

Value

the list of visualized coloredmeshes

`vis.colortable.legend` *Create a separate legend plot for a colortable or an annotation.*

Description

This plots a legend for a colortable or an atlas (annotation), showing the region names and their assigned colors. This function creates a new plot.

Usage

```
vis.colortable.legend(colortable, ncols = 1L, plot_struct_index = TRUE)
```

Arguments

`colortable`	dataframe, a colortable as returned by `read.fs.colortable` or the inner 'colortable_df' returned by `subject.annot`. One can also pass an annotation (*fs.annot* instance).
`ncols`	positive integer, the number of columns to use when plotting
`plot_struct_index`	logical, whether to plot the region index from tge 'struct_index' field. If there is no such field, this is silently ignored.

Note

This function plots one or more legends (see `legend`). You may have to adapt the device size before calling this function if you inted to plot a large colortable.

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
   vis.colortable.legend(annot$colortable_df, ncols=3);

## End(Not run)
```

`vis.data.on.fsaverage` *Visualize arbitrary data on the fsaverage template subject, if available.*

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain. This function tries to automatically retrieve the subjects dir of the fsaverage template subject by checking the environment variables SUBJECTS_DIR and FREESURFER_HOME for the subject. The subject is required for its surfaces, which are not shipped with this package for licensing reasons.

Usage

```
vis.data.on.fsaverage(
  subjects_dir = NULL,
  vis_subject_id = "fsaverage",
  morph_data_lh = NULL,
  morph_data_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL,
  morph_data_both = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Defaults to 'fsaverage'.

`morph_data_lh`	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
`morph_data_rh`	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.
`morph_data_both`	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, 'morph_data_lh' and 'morph_data_rh' must be NULL.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other morphometry visualization functions: `vis.data.on.subject()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`

`vis.data.on.group.native`

Visualize native space data on a group of subjects.

Description

Plot surface data on the native space surfaces of a group of subjects and combine the tiles into a single large image.

Usage

```
vis.data.on.group.native(
  subjects_dir,
  subject_id,
  morph_data_both,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_morph.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	vector of character strings, the subject identifiers
`morph_data_both`	named list of numerical vectors, the morph data for both hemispheres of all subjects. Can be loaded with `group.morph.native`.
`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.

`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`...`	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'morph_data_both' parameter. The surfaces are loaded in the order of the 'subject_id' parameter, so the order in both must match.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi`
`'range'=c(0, 4)))`.

See Also

Other group visualization functions: `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.coloredmeshes`
`vis.group.morph.native()`, `vis.group.morph.standard()`

`vis.data.on.group.standard`

Visualize standard space data for a group on template.

Description

Plot standard space data for a group of subjects onto a template brain and combine the tiles into a single large image.

Usage

```
vis.data.on.group.standard(
  subjects_dir,
  vis_subject_id,
  morph_data_both,
  captions = NULL,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_morph.png",
  num_per_row = 5L,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

`subjects_dir`	character string, the path to the SUBJECTS_DIR containing the template subject
`vis_subject_id`	character string, the template subject name. A typical choice is 'fsaverage'.
`morph_data_both`	named list of numerical vectors, 4D array or dataframe, the morph data for both hemispheres of all subjects. Can be loaded with `group.morph.standard` or `group.morph.standard.sf`.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`...`	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subject data are plotted row-wise, in the order in which they appear in the 'morph_data_both' parameter.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi` `'range'=c(0, 4)))`.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.group.annot()`, `vis.group.coloredmeshes()` `vis.group.morph.native()`, `vis.group.morph.standard()`

`vis.data.on.subject` *Visualize arbitrary data on the surface of any subject.*

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.data.on.subject(
  subjects_dir,
  vis_subject_id,
  morph_data_lh = NULL,
  morph_data_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL,
  morph_data_both = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`morph_data_lh`	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
`morph_data_rh`	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.

`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.
`morph_data_both`	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, 'morph_data_lh' and 'morph_data_rh' must be NULL.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other morphometry visualization functions: `vis.data.on.fsaverage()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
   morph_data_rh = NULL;
   vis.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);

## End(Not run)
```

`vis.dti.trk` *Visualize DTI tracks from Diffusion Toolkit/TrackVis TRK format file.*

Description

Visualize DTI tracks from Diffusion Toolkit/TrackVis TRK format file.

Usage

```
vis.dti.trk(
  trk,
  filter_tracks = list(min_length = 15, min_segment_count = 6),
  color_by_orientation = FALSE
)
```

Arguments

`trk`	character string, the path to a TRK file that should be loaded. Alternatively, a loaded `trk` instance as returned by `freesurferformats::read.dti.trk`.
`filter_tracks`	optional, named list of filters. Can contain fields `min_length` and `min_segment_count`. Set the whole thing to `NULL` or an entry to 0 for no filtering.
`color_by_orientation`	logical, whether to color the tracks by orientation. Slower, but may make the resulting visualization easier to interprete.

Value

The (loaded or received) `trk` instance. Note that this function is typically called for the side effect of visualization.

Note

The current simple implementation is very slow if the number of tracks becomes large (several thousand tracks).

Examples

```
## Not run:
# Create the following file with Diffusion Toolkit from your DTI data:
trk = freesurferformats::read.dti.trk("~/data/tim_only/tim/DICOM/dti.trk");
vis.dti.trk(trk);

## End(Not run)
```

`vis.export.from.coloredmeshes`

Export high-quality brainview image with a colorbar.

Description

This function serves as an easy (but slightly inflexible) way to export a high-quality, tight-layout, colorbar figure to disk. If no colorbar is required, one can use `vislayout.from.coloredmeshes` instead.

Usage

```
vis.export.from.coloredmeshes(
  coloredmeshes,
  colorbar_legend = NULL,
  img_only = TRUE,
  horizontal = TRUE,
  silent = TRUE,
  quality = 1L,
  output_img = "fsbrain_arranged.png",
  image.plot_extra_options = NULL,
  large_legend = TRUE,
  view_angles = get.view.angle.names(angle_set = "t4"),
  style = "default",
  grid_like = TRUE,
  background_color = "white",
  transparency_color = NULL,
  ...
)
```

Arguments

`coloredmeshes`	list of coloredmesh. A coloredmesh is a named list as returned by the ‘coloredmesh.from*‘ functions (like `coloredmesh.from.morph.native`). It has the entries ’mesh’ of type tmesh3d, a ’col’, which is a color specification for such a mesh. The ‘vis*‘ functions (like `vis.subject.morph.native`) all return a list of coloredmeshes.

`colorbar_legend`	character string or NULL, the title for the colorbar.
`img_only`	logical, whether to return only the resulting image
`horizontal`	logical, whether to plot the colorbar horizontally (TRUE) or vertically (FALSE). Pass 'NULL' to force no colorbar at all.
`silent`	logical, whether to suppress messages
`quality`	integer, an arbitrary quality. This is the resolution per tile before trimming, divided by 1000, in pixels. Example: 1L means 1000x1000 pixels per tile before trimming. Currently supported values: `1L..2L`. Note that the resolution you can get is also limited by your screen resolution.
`output_img`	string, path to the output file. Defaults to "fsbrain_arranged.png"
`image.plot_extra_options`	named list, custom options for fields::image.plot. Overwrites those derived from the quality setting. If in doubt, leave this alone.
`large_legend`	logical, whether to plot extra large legend text, affects the font size of the colorbar_legend and the tick labels.
`view_angles`	list of strings. See `get.view.angle.names` for all valid strings.
`style`	the rendering style, see `material3d` or use a predefined style like 'default' or 'shiny'.
`grid_like`	logical, passed to `vislayout.from.coloredmeshes`.
`background_color`	hex color string (like '#FFFFFF'), the color to use for the background. Ignored if 'transparency_color' is not NULL. To get a transparent background, use 'transparency_color' instead of this parameter. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
`transparency_color`	hex color string (like '#FFFFFF'), the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try '#FFFFFF' (white) or '#000000' (black) if in doubt. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
`...`	extra arguments passed to `vislayout.from.coloredmeshes`.

Value

magick image instance or named list, depending on the value of 'img_only'. If the latter, the list contains the fields 'rev_vl', 'rev_cb', and 'rev_ex', which are the return values of the functions `vislayout.from.coloredmeshes`, `coloredmesh.plot.colorbar.separate`, and `combine.colorbar.with.brainview.`, respectively.

Note

Note that your screen resolution has to be high enough to generate the final image in the requested resolution, see the 'fsbrain FAQ' vignette for details and solutions if you run into trouble.

See Also

This function should not be used anymore, it will be deprecated soon. Please use the `export` function instead.

Examples

```
## Not run:
    rand_data = rnorm(327684, 5, 1.5);
    cm = vis.data.on.fsaverage(morph_data_both=rand_data,
      rglactions=list('no_vis'=T));
    vis.export.from.coloredmeshes(cm, colorbar_legend='Random data',
      output_img="~/fsbrain_arranged.png");

## End(Not run)
```

`vis.fs.surface` *Visualize fs.surface mesh*

Description

Render a mesh. All mesh formats supported by the *freesurferformats* package are supported, including OFF, PLY, OBJ, STL, and many more.

Usage

```
vis.fs.surface(
  fs_surface,
  col = "white",
  per_vertex_data = NULL,
  hemi = "lh",
  makecmap_options = mkco.seq(),
  ...
)
```

Arguments

`fs_surface`	an fs.surface instance, as returned by function like `subject.surface` or `read.fs.surface`. If a character string, it is assumed to be the full path of a surface file, and the respective file is loaded with `read.fs.surface`. If parameter 'hemi' is 'both', this must be a hemilist. A single `rgl::tmesh` is also fine.
`col`	vector of colors, the per-vertex-colors. Defaults to white. Must be a single color or one color per vertex. If parameter 'hemi' is 'both', this must be a hemilist.
`per_vertex_data`	numerical vector, per-vertex data. If given, takes precedence over 'col'. Used to color the mesh using the colormap options in parameter 'makecmap_options'. If a character string, it is assumed to be the full path of a morphometry data file,

	and the respective file is loaded with read.fs.morph. If parameter 'hemi' is 'both', this must be a hemilist.
`hemi`	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.
`makecmap_options`	named list of parameters to pass to makecmap. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`...`	extra parameters to pass to vis.coloredmeshes.

Value

see `vis.coloredmeshes`

Note

This function can be used to visualize arbitrary triangular meshes in R. Despite its name, it is not limited to brain surface meshes.

`vis.group.annot` *Plot atlas annotations for a group of subjects.*

Description

Plot atlas annotations for a group of subjects and combine them into a single large image.

Usage

```
vis.group.annot(
  subjects_dir,
  subject_id,
  atlas,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_annot.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	vector of character strings, the subject identifiers
`atlas`	vector of character strings, the atlas names. Example: `c('aparc', 'aparc.a2009s')`

`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`...`	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id' and 'atlas'.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.data.on.group.standard()`, `vis.group.coloredmeshes()`, `vis.group.morph.native()`, `vis.group.morph.standard()`

---

vis.group.coloredmeshes

Plot coloredmeshes for a group of subjects.

Description

Plot coloredmeshes for a group of subjects into a single image.

Usage

```
vis.group.coloredmeshes(
  coloredmeshes,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_annot.png",
  num_per_row = 5L,
  captions = NULL,
  background_color = "white"
)
```

Arguments

`coloredmeshes`	a list of coloredmeshes lists, each entry in the outer list contains the hemilist of coloredmeshes (lefgt and right hemisphere mesh) for one subject.
`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
`background_color`	color for image background (transparency is not supported).

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

This is a mid-level function, end users may want to call high-level functions like `vis.group.annot` instead.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.morph.native()`, `vis.group.morph.standard()`

---

vis.group.morph.native

Plot native space morphometry data for a group of subjects.

Description

Plot native space morphometry data for a group of subjects and combine them into a single large image.

Usage

```
vis.group.morph.native(
  subjects_dir,
  subject_id,
  measure,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_morph.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	vector of character strings, the subject identifiers
`measure`	vector of character strings, the morphometry measures, e.g., `c('thickness', 'area')`
`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`...`	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id' and 'measure'.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi` `'range'=c(0, 4)))`.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.coloredmeshes()`, `vis.group.morph.standard()`

---

vis.group.morph.standard

Plot standard space morphometry data for a group of subjects.

Description

Plot standard space morphometry data for a group of subjects and combine them into a single large image.

Usage

```
vis.group.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  fwhm = "10",
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_morph.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	vector of character strings, the subject identifiers
`measure`	vector of character strings, the morphometry measures, e.g., `c('thickness', 'area')`
`fwhm`	vector of character strings, the smoothing kernel FWHM strings, e.g., `c('0', '10', '15')`
`view_angles`	see `get.view.angle.names`.
`output_img`	character string, the file path for the output image. Should end with '.png'.
`num_per_row`	positive integer, the number of tiles per row.
`captions`	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`...`	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id', 'measure' and 'fwhm'.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi` `'range'=c(0, 4)))`.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.coloredmeshes()`, `vis.group.morph.native()`

`vis.labeldata.on.subject`

Visualize a label on the surface of a subject.

Description

Visualizes a label. Note that a label is just a set of vertices, and that you can use this function to visualize sets of vertices, e.g., to see where on the mesh a certain vertex lies. It may be helpful the visualize the vertex with its neighbors, because otherwise it may be too small to spot. Use the function [fsbrain::mesh.vertex.neighbors] to get them. It is advisable to set the view to the interactive 'si' mode and use the 'inflated' surface to identify single vertices.

Usage

```
vis.labeldata.on.subject(
  subjects_dir,
  vis_subject_id,
  lh_labeldata,
  rh_labeldata,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = label.colFn.inv),
  style = "default",
  ...
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`lh_labeldata`	integer vector of vertex indices for the left hemisphere
`rh_labeldata`	integer vector of vertex indices for the right hemisphere
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.

`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
`...`	extra arguments to pass to `coloredmesh.from.label`.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other label functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `subject.lobes()`, `subject.mask()`, `vis.subject.label()`

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();

  # Define the data to use:
  subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
  lh_labeldata = c(1000, 1001, 1002);   # only the vertices, will be tiny.
  subject_id = 'subject1';
```

```
  surface = 'white'; # Should use 'inflated', but we do not currently
                     # ship it for the example subject to reduce download size.

  # For the right hemi, extend them to neighborhood for better visibility:
  rh_labeldata = c(500, 5000);
  rh_surface = subject.surface(subjects_dir, subject_id, surface, 'rh');
  rh_labeldata_neighborhood = mesh.vertex.neighbors(rh_surface, rh_labeldata);
  vis.labeldata.on.subject(subjects_dir, subject_id, lh_labeldata,
   rh_labeldata_neighborhood$vertices, surface=surface, views=c('si'));

## End(Not run)
```

`vis.mask.on.subject` *Visualize a vertex mask on the surface of a subject.*

Description

A mask is a logical vector that contains one value per vertex. You can create it manually, or use functions like `mask.from.labeldata.for.hemi` to create and modify it. Check the example for this function.

Usage

```
vis.mask.on.subject(
  subjects_dir,
  vis_subject_id,
  mask_lh,
  mask_rh,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = label.colFn.inv),
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`mask_lh`	logical vector or NULL, the mask to visualize on the left hemisphere surface. Must have the same length as the lh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.

`mask_rh`	logical vector or NULL, the mask to visualize on the right hemisphere surface. Must have the same length as the rh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other mask functions: `coloredmesh.from.mask()`, `mask.from.labeldata.for.hemi()`

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Examples

```
## Not run:
   fsbrain::download_optional_data();

  # Define the data to use:
  subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
  subject_id = 'subject1';
  surface = 'white';
  hemi = 'both';
  atlas = 'aparc';
  region = 'bankssts';

  # Create a mask from a region of an annotation:
  lh_annot = subject.annot(subjects_dir, subject_id, 'lh', atlas);
  rh_annot = subject.annot(subjects_dir, subject_id, 'rh', atlas);
  lh_label = label.from.annotdata(lh_annot, region);
  rh_label = label.from.annotdata(rh_annot, region);
  lh_mask = mask.from.labeldata.for.hemi(lh_label, length(lh_annot$vertices));
  rh_mask = mask.from.labeldata.for.hemi(rh_label, length(rh_annot$vertices));

  # Edit the mask: add the vertices from another region to it:
  region2 = 'medialorbitofrontal';
  lh_label2 = label.from.annotdata(lh_annot, region2);
  rh_label2 = label.from.annotdata(rh_annot, region2);
  lh_mask2 = mask.from.labeldata.for.hemi(lh_label2, length(lh_annot$vertices),
   existing_mask = lh_mask);
  rh_mask2 = mask.from.labeldata.for.hemi(rh_label2, length(rh_annot$vertices),
   existing_mask = rh_mask);
  # Visualize the mask:
  vis.mask.on.subject(subjects_dir, subject_id, lh_mask2, rh_mask2);

## End(Not run)
```

`vis.path.along.verts` *Draw a 3D line from vertex to vertex*

Description

To get a nice path along the surface, pass the vertex indices along a geodesic path. Note: You can first open an interactive brain view (‘views='si'‘) with a vis* function like `vis.subject.morph.native`, then run this function to draw into the active plot.

Usage

```
vis.path.along.verts(
  surface_vertices,
  path_vertex_indices = NULL,
  do_vis = TRUE,
```

```
  color = "#FF0000",
  no_material = FALSE
)
```

Arguments

`surface_vertices`	float matrix of size (n, 3), the surface vertex coordinates, as returned as part of `subject.surface` or `read.fs.surface`, in the member "vertices". Can also be a `freesurferformats::fs.surface` or `rgl::tmesh3d` instance, in which case the coordinates are extracted automatically.
`path_vertex_indices`	vector of vertex indices, the path. You will need to have it computed already. (This function does **not** compute geodesic paths, see `geodesic.path` for that. You can use it to visualize such a path though.) If omitted, the vertex coordinates will be traversed in their given order to create the path.
`do_vis`	logical, whether to actually draw the path.
`color`	a color string, like '#FF0000' to color the path.
`no_material`	logical, whether to use set the custom rendering material properties for path visualization using `rgl::material3d` before plotting. If you set this to FALSE, no material will be set and you should set it yourself before calling this function, otherwise the looks of the path are undefined (dependent on the default material on your system, or the last material call). Setting this to TRUE also means that the 'color' argument is ignored of course, as the color is part of the material.

Value

n x 3 matrix, the coordinates of the path, with appropriate ones duplicated for rgl pair-wise segments3d rendering.

See Also

`vis.paths` if you need to draw many paths, `geodesic.path` to compute a geodesic path.

Other surface mesh functions: `face.edges()`, `label.border()`, `mesh.vertex.included.faces()`, `mesh.vertex.neighbors()`, `subject.surface()`

Examples

```
## Not run:
  sjd = fsaverage.path(TRUE);
  surface = subject.surface(sjd, 'fsaverage3',
    surface = "white", hemi = "lh");
  p = geodesic.path(surface, 5, c(10, 20));
  vis.subject.morph.native(sjd, 'fsaverage3', views='si');
  vis.path.along.verts(surface$vertices, p[[1]]);

## End(Not run)
```

`vis.paths` *Visualize many paths.*

Description

Visualize many paths.

Usage

```
vis.paths(coords_list, path_color = "#FF0000")
```

Arguments

`coords_list`	list of `m` matrices, each `n` x 3 matrix must contain the 3D coords for one path.
`path_color`	a color value, the color in which to plot the paths.

Note

This function is a lot faster than calling `vis.path.along.verts` many times and having it draw each time.

`vis.paths.along.verts` *Visualize several paths in different colors.*

Description

Visualize several paths in different colors.

Usage

```
vis.paths.along.verts(
  surface_vertices,
  paths,
  color = viridis::viridis(length(paths))
)
```

Arguments

`surface_vertices`	float matrix of size (n, 3), the surface vertex coordinates, as returned as part of `subject.surface` or `read.fs.surface`, in the member "vertices". Can also be a `freesurferformats::fs.surface` or `rgl::tmesh3d` instance, in which case the coordinates are extracted automatically.
`paths`	list of positive integer vectors, the vertex indices of the paths
`color`	a color string, like '#FF0000' to color the path.

`vis.region.values.on.subject` *Visualize arbitrary data, one value per atlas region, on the surface of any subject (including template subjects).*

Description

This function can be used for rendering a single value (color) for all vertices of an atlas region. The typical usecase is the visualization of results of atlas-based analyses, e.g., p-value, means or other aggregated values over all vertices of a region.

Usage

```
vis.region.values.on.subject(
  subjects_dir,
  subject_id,
  atlas,
  lh_region_value_list,
  rh_region_value_list,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  value_for_unlisted_regions = NA,
  draw_colorbar = FALSE,
  makecmap_options = mkco.heat(),
  bg = NULL,
  silent = FALSE,
  style = "default",
  border = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`atlas`	string. The brain atlas to use. E.g., 'aparc' or 'aparc.a2009s'.
`lh_region_value_list`	named list. A list for the left hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list, but then the its length must exactly match the number of atlas regions. The order of values must also match the order of regions in the annotation, of course. The resulting mapping will be printed so you can check it (unless 'silent' is set).

`rh_region_value_list`	named list. A list for the right hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`value_for_unlisted_regions`	numerical scalar or 'NA', the value to assign to regions which do not occur in the region_value_lists. Defaults to 'NA'.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.
`silent`	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
`border`	logical, whether to add a black border around the regions. Alternatively, the parameter can be given as a named list with entries 'color' and 'expand_inwards', where the latter defines the borders thickness. E.g., `border = list('color'='#FF0000', 'expand_inwards'=2L)`. Border computation is slow, sorry.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard`(), `vis.subject.pre`(), `vis.symmetric.data.on.subject`(), `vislayout.from.coloredmeshes`()

Other region-based visualization functions: `vis.subject.annot`()

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   atlas = 'aparc';   # Desikan atlas
   # For the left hemisphere, we just assign a subset of the
   # atlas regions. The others will get the default value.
  lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7, "postcentral"=0.8, "lingual"=0.6);
   # For the right hemisphere, we retrieve the full list of regions for
   # the atlas, and assign random values to all of them.
  atlas_region_names = get.atlas.region.names(atlas, template_subjects_dir = subjects_dir,
    template_subject='subject1');
   rh_region_value_list = rnorm(length(atlas_region_names), 3.0, 1.0);
   names(rh_region_value_list) = atlas_region_names;
   vis.region.values.on.subject(subjects_dir, 'subject1', atlas,
    lh_region_value_list, rh_region_value_list);

## End(Not run)
```

`vis.seg.legend` *Plot legend for a brain volume segmentation based on colorLUT.*

Description

Plot legend for a brain volume segmentation based on colorLUT.

Usage

```
vis.seg.legend(colortable, segvol, ...)
```

Arguments

`colortable`	a colortable data.frame, or a character string, which will be treated as a filename and loaded with `read.fs.colortable`. Typically `FS_HOME/FreeSurferColorLUT.txt`.
`segvol`	optional 3D or 4D array of integer data, the brain segmentation. Or a character string, which will be treated as a filename and loaded with `read.fs.volume`. If given, only colortable entries which actually occur in the volume data are plotted. If `NULL`, all entries are plotted, which may be a lot.
`...`	passed on to vis.colortable.legend

Examples

```
## Not run:
ct = file.path(fs.home(), "FreeSurferColorLUT.txt");
seg = file.path(fs.home(), "subjects", "fsaverage", "mri", "aseg.mgz");
vis.seg.legend(ct, seg);


## End(Not run)
```

`vis.subject.annot` *Visualize an annotation for a subject.*

Description

Creates a surface mesh, loads the colors from the annotation, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.subject.annot(
  subjects_dir,
  subject_id,
  atlas,
  hemi = "both",
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  outline = FALSE,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`atlas`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded. Can also be a hemilist of already loaded annotations.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".

`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`outline`	logical, whether to draw an outline only instead of filling the regions. Defaults to 'FALSE'. Instead of passing 'TRUE', one can also pass a list of extra parameters to pass to `annot.outline`, e.g., `outline=list('outline_color'='#000000')`. Using this increases computation time dramatically, sorry for the performance.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other region-based visualization functions: `vis.region.values.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   vis.subject.annot(subjects_dir, 'subject1', 'aparc', 'both');

## End(Not run)
```

`vis.subject.label` *Visualize a binary label for a subject.*

Description

Visualize a label for a subject. A label is just a logical vector with one entry for each vertex in the mesh. Each vertex may additionally be associated with a scalar value, but this function ignored that.

Usage

```
vis.subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = label.colFn.inv, col.na = "#FFFFFF00"),
  map_to_NA = 0L,
  bg = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`label`	string. Name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

`map_to_NA`	the value or value range that should **not** be considered part of the label, and should thus be plotted as background color. Only used if 'bg' is not 'NULL'. If a single value, only excatly this value is used (typically 0). If two values, they are interpreted as a range, and a values between them are mapped to NA. If you prefer to map the data to NA yourself before using this function, pass 'NULL'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other label functions: `apply.label.to.morphdata()`, `apply.labeldata.to.morphdata()`, `subject.lobes()`, `subject.mask()`, `vis.labeldata.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   subject_id = 'subject1';
   surface = 'white';
   hemi = 'both';
   label = 'cortex.label';
   vis.subject.label(subjects_dir, subject_id, label, hemi, views="si");

## End(Not run)
```

`vis.subject.morph.native`

Visualize native space morphometry data for a subject.

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.subject.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi = "both",
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  cortex_only = FALSE,
  style = "default",
  makecmap_options = mkco.seq(),
  bg = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	string. The subject identifier.
`measure`	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.

`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

Other morphometry visualization functions: `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   vis.subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh', views=c("t9"));

## End(Not run)
```

`vis.subject.morph.standard`
Visualize native space morphometry data for a subject or a group.

Description

Renders standard space morphometry data for a single subject, or the group mean for a group of subjects. The default template subject is fsaverage.

Usage

```
vis.subject.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi = "both",
  fwhm = "10",
  surface = "white",
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  cortex_only = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id`	character string or vector of character strings, the subject or subjects. For a single subjects, its data will be plotted. If a group of subjects is given instead, at each vertex the mean value over all the subjects will be plotted.
`measure`	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`fwhm`	string, smoothing setting (full width at half maximum of the kernel). The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".

`template_subject`	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to 'fsaverage'.
`template_subjects_dir`	The template subjects dir. If NULL, the value of the parameter 'subjects_dir' is used. If you have FreeSurfer installed and configured, and are using the standard fsaverage subject, try passing the result of calling 'file.path(Sys.getenv('FREESURFER_HOME'), 'subjects')'.
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`cortex_only`	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are *not* part of the cortex (as defined by the label file 'label/?h.cortex.label') should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
`bg`	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see `collayer.bg` for valid strings.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vislayout.from.colored`

Other morphometry visualization functions: `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.subject.morph.native()`, `vis.symmetric.data.on.subject()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   fsaverage_dir = file.path(Sys.getenv('FREESURFER_HOME'), 'subjects');
   if(dir.exists(fsaverage_dir)) {
       vis.subject.morph.standard(subjects_dir, 'subject1', 'thickness', 'lh',
       '10', template_subjects_dir=fsaverage_dir);
   }
   # The last command will load the file
   #  *<subjects_dir>/subject1/surf/lh.thickness.fwhm10.fsaverage.mgh* and
   #  visualize the data on *$FREESURFER_HOME/subjects/fsaverage/surf/lh.white*.

## End(Not run)
```

`vis.subject.pre` *Visualize pre-loaded data.*

Description

Visualize pre-loaded data.

Usage

```
vis.subject.pre(
  surfaces,
  pervertex_data,
  hemi = "both",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  style = "default",
  makecmap_options = mkco.seq()
)
```

Arguments

`surfaces`	a `hemilist` of surfaces loaded with a function like `freesurferformats::read.fs.surface`.
`pervertex_data`	a `hemilist` of per-vertex data for the surfaces, i.e., a list of numeric vectors. E.g., loaded from a moorphometry data file with a function like `freesurferformats::read.fs.morph`.

`hemi`	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to 'FALSE'. See `coloredmesh.plot.colorbar.separate` for an alternative.
`style`	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

---

vis.symmetric.data.on.subject

Visualize clusters or activation data on the surface of any subject.

Description

This function is intended to plot symmetric data around zero (like positive and negative activation data, signed p-values, etc.) on a subject's surface. It is a thin wrapper around `vis.data.on.subject`.

Usage

```
vis.symmetric.data.on.subject(
  subjects_dir,
  vis_subject_id,
```

```
  morph_data_lh = NULL,
  morph_data_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = TRUE,
 makecmap_options = list(colFn = cm.cbry(), symm = TRUE, col.na = "#FFFFFF00", n = 200),
  map_to_NA = c(0),
  bg = NULL,
  morph_data_both = NULL,
  style = "default"
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`vis_subject_id`	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
`morph_data_lh`	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
`morph_data_rh`	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
`surface`	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
`views`	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))`. See `rglactions`.
`draw_colorbar`	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You

	may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to ‘FALSE‘. See `coloredmesh.plot.colorbar.separate` for an alternative.
`makecmap_options`	named list of parameters to pass to `makecmap`. Must not include the unnamed first parameter, which is derived from ’measure’. Should include at least a colormap function as name ’colFn’.
`map_to_NA`	the value or value range that should **not** be considered a cluster, and should thus be plotted as background color. These values will be set to NA, leading to transparcent rendering, so the background will be visible instead. If a single value, only exactly this value is used (typically 0). If two values, they are interpreted as a range, and a values between them are mapped to NA. If you prefer to map the data to NA yourself before using this function or do not want to use a , pass ‘NULL‘.
`bg`	a background definition. Can be a surface color layer or a character string like ’curv_light’ to select a pre-defined layer, see `collayer.bg` for valid strings.
`morph_data_both`	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, ’morph_data_lh’ and ’morph_data_rh’ must be NULL.
`style`	character string, a rendering style, e.g., ’default’, ’shiny’ or ’semitransparent’.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres()`, `highlight.vertices.on.subject()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.subject.annot()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vislayout.from.coloredmes`

Other morphometry visualization functions: `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`

Examples

```
## Not run:
    fsbrain::download_optional_data();
    subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
    morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
    morph_data_rh = NULL;
   vis.symmetric.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);

## End(Not run)
```

`vislayout.from.coloredmeshes`

Visualize coloredmeshes from several angles and combine the images into a new figure.

Description

Create a tight layout view of coloredmeshes from several angles. Creates separate ‘sd_<angle>‘ images, then crops and finally merges them into a single output image with image magick. The ‘coloredmeshes‘ to pass to this function are usually obtained by running any ‘vis*‘ function (like `vis.subject.morph.native`, `vis.subject.morph.standard`, `vis.subject.label`, `vis.subject.annot`, and others). That means you can use this function to visualize all kinds of data, e.g., morphometry data in native and standard space, labels, and brain atlases.

Usage

```
vislayout.from.coloredmeshes(
  coloredmeshes,
  view_angles = get.view.angle.names(angle_set = "t4"),
  rgloptions = rglo(),
  rglactions = list(),
  style = "default",
  output_img = "fsbrain_arranged.png",
  silent = FALSE,
  grid_like = TRUE,
  background_color = "white",
  transparency_color = NULL
)
```

Arguments

`coloredmeshes,`	list of coloredmesh. A coloredmesh is a named list as returned by the ‘coloredmesh.from*‘ functions (like `coloredmesh.from.morph.native`). It has the entries ’mesh’ of type tmesh3d, a ’col’, which is a color specification for such a mesh. The ‘vis*‘ functions (like `vis.subject.morph.native`) all return a list of coloredmeshes.
`view_angles`	list of strings. See `get.view.angle.names` for all valid strings.
`rgloptions`	option list passed to `par3d`. Example: `rgloptions = list("windowRect"=c(50,50,1000,1000))`.
`rglactions`	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action.
`style`	character string, a rendering style, e.g., ’default’, ’shiny’ or ’semitransparent’. Alternatively, a named list of style parameters (see `material3d`), e.g., `list("shininess"=50, specular="black", alpha=0.5)`. Use the magic word ’from_mesh’ to use the ’style’ field of each coloredmesh instead of a single, global style. In that case, you will have to make sure your meshes have such a field, if not, the style ’default’ is used as a fallback for those which don’t.

`output_img`	string, path to the output file. Defaults to "fsbrain_arranged.png"
`silent`	logical, whether to suppress all messages
`grid_like`	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally. Passed to `arrange.brainview.images`.
`background_color`	hex color string (like '#FFFFFF'), the color to use for the background. Ignored if 'transparency_color' is not NULL. To get a transparent background, use 'transparency_color' instead of this parameter. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
`transparency_color`	hex color string (like '#FFFFFF'), the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try '#FFFFFF' (white) or '#000000' (black) if in doubt. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!

Value

named list, see `arrange.brainview.images` for details

See Also

Other visualization functions: `highlight.vertices.on.subject.spheres`(), `highlight.vertices.on.subject`(), `vis.color.on.subject`(), `vis.data.on.fsaverage`(), `vis.data.on.subject`(), `vis.labeldata.on.subject`(), `vis.mask.on.subject`(), `vis.region.values.on.subject`(), `vis.subject.annot`(), `vis.subject.label`(), `vis.subject.morph.native`(), `vis.subject.morph.standard`(), `vis.subject.pre`(), `vis.symmetric.data.on.subj`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   # Use any vis function to get coloredmeshes. You can visualize morphometry,
   #  labels, an atlas, whatever. You can suppress the view unless you need it.
   coloredmeshes = vis.subject.morph.native(subjects_dir, "subject1", "thickness",
    cortex_only=TRUE, rglactions=list("clip_data"=c(0.05, 0.95)),
    views=NULL);
   # The meshes contain the surface, data, and color information and can be
   #  visualized. You could adapt the rendering style while doing so:
   vislayout.from.coloredmeshes(coloredmeshes, style='shiny');
   # You could change the rendering style on a per-mesh basis.
   coloredmeshes[[1]]$style = list("shininess"=50, alpha=0.5);
   vislayout.from.coloredmeshes(coloredmeshes, style='from_mesh');

## End(Not run)
```

`vol.boundary.box` *Compute 3D bounding box of a volume.*

Description

Compute the axis-aligned foreground bounding box of a 3D volume, i.e. the inner foreground area that must be retained if you want to remove all background from the corners of the volume. The foreground is determined by thresholding, such that all values greater than 0 are considered foreground. See `vol.boundary.mask` for details.

Usage

```
vol.boundary.box(volume, threshold = 0L, apply = FALSE)
```

Arguments

`volume`	a 3D image volume
`threshold`	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value will be considered ‘foreground‘ voxels.
`apply`	logical, whether to directly apply the bounding box and return the resulting volume instead.

Value

named list with 2 entries: ‘from‘ is an integer vector of length 3, defining the minimal (x,y,z) foreground indices. ‘to‘ is an integer vector of length 3, defining the maximal (x,y,z) foreground indices.

See Also

Other volume utility: `vol.imagestack()`, `vol.merge()`, `vol.overlay.colors.from.activation()`, `vol.planes()`, `vol.slice()`

`vol.boundary.box.apply` *Apply a boundary box to a volume, returning the inner volume part*

Description

Apply a boundary box to a volume, returning the inner volume part

Usage

```
vol.boundary.box.apply(volume, bbox)
```

Arguments

`volume`	a 3D image volume
`bbox`	the boundary box

Value

a 3D image volume, the inner volume part, resulting from the application of the boundary box

---

`vol.hull` *Retain only the outer hull voxels of the foreground.*

Description

Filters the *foreground* voxel in the volume by keeping only an outer border of voxels, and setting the inner core voxels to 'NA'. This is a utility function for voxel-based visualization. The goal is to remove the inner voxels, which will not be visible anyways, and thus to dramatically reduce the number of triangles that will need to be computed for the mesh.

Usage

```
vol.hull(volume, thickness = 1L, axes = c(2L))
```

Arguments

`volume`	numeric 3d array, must contain foreground voxel and background voxels. The latter must have value 'NA'. This function assumes that a solid foreground object surrounded by background exists in the volume.
`thickness`	integer, the width of the border in voxels, i.e., how many of the voxels in each upright column to keep at the top and at the bottom.
`axes`	integer vector, the axes to use. Valid values in the vector are 1L, 2L and 3L. You will have to use all 3 axes if you do not want any holes in the object. (Obvisouly, having noise around the object can still lead to holes.)

Value

numeric 3d array, a filtered version of the input. It contains at least as many 'NA' voxels as the input. If the function had any effect, it contains a lot more 'NA' values. The other values and the volume dimensions are left unchanged.

`vol.imagestack` *Turn volume into an ImageMagick image stack.*

Description

Create an image from each slice along the axis, then stack those into an ImageMagick image stack.

Usage

```
vol.imagestack(volume, axis = 1L, intensity_scale = 255)
```

Arguments

`volume`	a 3D image volume. Can be numeric, or something that can be read directly by `magick::image_read` in 2D matrices (slices along the axis), e.g., a 3D array of color strings. If a 2D matrix is passed, the resulting stack will contain a single image.
`axis`	positive integer in range 1L..3L or an axis name, the axis to use.
`intensity_scale`	integer, value by which to scale the intensities in the volume to the range '[0, 1]'. Only used for numeric volumes. Set to NULL for data that can be read directly by `magick::image_read`, and to 1 for intensity data that requires no scaling. Defaults to 255, which is suitable for 8 bit image data.

Value

a vectorized ImageMagick image, containing one subimage per slice. This can be interpreted as an animation or whatever.

See Also

Other volume utility: `vol.boundary.box()`, `vol.merge()`, `vol.overlay.colors.from.activation()`, `vol.planes()`, `vol.slice()`

`vol.intensity.to.color` *Convert integer intensity image to RGB color string form.*

Description

Convert a gray-scale image defined by intensity values in range '[0, 1]' to an image with identical dimensions that contains an R color string (like '#222222') at each position. The color strings are computed from the intensities, by taking the intensity value as the value for all three RGB channels. I.e., the output is still gray-scale, but defined in RGB space. To make it clear, this function does **not** apply a colormap. It only changes the representation of the data, not the resulting colors.

Usage

```
vol.intensity.to.color(volume, scale = NULL)
```

Arguments

volume	numeric array, typically a 3D image with intensities in range '[0, 1]'. This function now also supports numeric matrices (2D images, slices) and numeric vectors (1D).
scale	numeric or character string, a scaling to apply to the values. Defaults to NULL, which means *no scaling* and requires the values in 'volume' to be in rage '[0, 1]'. You can pass a number like 255 or the string 'normalize' to scale based on the data. You can pass the string 'normalize_if_needed' to scale only if the data is *outside* the range '[0, 1]', so that data in range '[0.3, 0.5]' would **not** be rescaled to '[0, 1]'.

Value

array (or matrix, or vector) of RGB color strings. All of them will represent gray values.

Examples

```
vol.intensity.to.color(c(0.0, 0.5, 1.0));
# output: "#000000" "#808080" "#FFFFFF"
vol.intensity.to.color(c(20, 186, 240), scale="normalize");
vol.intensity.to.color(c(20, 186, 240), scale=255);
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize");
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize_if_needed");
```

`vol.mask.from.segmentation`

Extract subset from a volume by value.

Description

Extract subset from a volume by value, set all other voxel values to 'NA'. Typically used to extract a brain structure (like corpus callosum) from a volume segmentation (like the 'mri/aseg.mgz' file of a subject). You should consider passing the volume and the include values as integers.

Usage

```
vol.mask.from.segmentation(volume, include_values)
```

Arguments

volume	numeric 3D array
include_values	numerical vector, the intensity values which qualify a voxel to be part of the result (without being set to NA)

Value

numerical array with same dimensions as the input volume. All values which are not part of ‘include_values‘ replaced with ‘NA‘.

`vol.merge` *Merge background volume and overlay to new colors.*

Description

Merge background volume and overlay to new colors.

Usage

```
vol.merge(
  volume,
  overlay_colors,
  bbox_threshold = 0L,
  forced_overlay_color = NULL
)
```

Arguments

`volume`	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range ‘[0, 1]‘.
`overlay_colors`	3D array of color strings, values which are not part of the overlay (and should display background in the result) must have ‘NA‘ instead of a color string. Must have same dimensions as the ‘volume‘.
`bbox_threshold`	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value in the background ‘volume‘ will be considered ‘foreground‘ voxels. Background-only slices at the borders of the volume will be discarded (in the merged, final image). Pass ‘NULL‘ to use the full image without applying any bounding box.
`forced_overlay_color`	NULL or an rgb color string, like ’#FF0000’ for red. If NULL, the activation colors will be used as foreground colors. Otherwise, the given color will be for all foreground vertices.

Value

3D array of color strings, the merged colors

See Also

Other volume utility: `vol.boundary.box()`, `vol.imagestack()`, `vol.overlay.colors.from.activation()`, `vol.planes()`, `vol.slice()`

`vol.overlay.colors.from.activation`

Generate colors for a 3D volume, based on the activation data and a colormap.

Description

Applies the colormap function to the data, then sets the alpha value (transparency) to full in all areas without any activation. Feel free to clip data or whatever before passing it, so that all your no-activation data has the same value.

Usage

```
vol.overlay.colors.from.activation(
  volume,
  colormap_fn = squash::blueorange,
  no_act_source_value = 0
)
```

Arguments

`volume`	a 3D array, the activation data (or p-values, effect sizes, or whatever)
`colormap_fn`	function, a colormap function
`no_act_source_value`	numerical scalar, the value from the data in 'volume' that means no activation. The output colors for this value will be set to 'NA'. Set to NULL to not change anything.

Value

a 3D matrix of color strings, with the same dimensions as the input volume

See Also

Other volume utility: `vol.boundary.box()`, `vol.imagestack()`, `vol.merge()`, `vol.planes()`, `vol.slice()`

`vol.overlay.colors.from.colortable` *Compute voxel colors based on colortable.*

Description

Use the intensity values of the voxels in volume and lookup the respective colors in a colortable.

Usage

```
vol.overlay.colors.from.colortable(
  volume,
  colortable,
  ignored_struct_indices = c(),
  ignored_struct_names = c("unknown", "Unknown")
)
```

Arguments

`volume`	numeric 3D array, the values should be integers present in the ‘struct_index‘ column of the colortable. All other values will be assigned ‘NA‘ as a color.
`colortable`	a colortable, as returned by `read.fs.colortable`, or a character string representing a path to a colortable file.
`ignored_struct_indices`	integer vector, ‘struct_index‘ entries in the colortable that should be ignored
`ignored_struct_names`	vector of character strings, ‘struct_name‘ entries in the colortable that should be ignored. Can be combined with ‘ignored_struct_indices‘.

Value

character string 3D array, the colors. Voxels in the volume which were not matched by the colortable are set to ‘NA‘ in it.

`vol.planes` *Translate names and indices of planes.*

Description

Translate names and indices of 3D image planes. The names only make sense if the data in the volume is in the default FreeSurfer conformed orientation.

Usage

```
vol.planes(plane = NULL)
```

Arguments

`plane`	NULL, a plane index, or a plane name.

Value

if ‘plane‘ is NULL, all available planes and their indices as a named list. If ‘plane‘ is an integer (a plane index), its name. If ‘plane‘ is an characters string (a plane name), its index.

See Also

Other volume utility: `vol.boundary.box()`, `vol.imagestack()`, `vol.merge()`, `vol.overlay.colors.from.activation` `vol.slice()`

`vol.slice` *Extract a slice of a 3D image stack.*

Description

Extracts one or more 2D slices from a 3D image (or a frame of a 4D image). To display the result, you can use `volvis.lightbox`.

Usage

```
vol.slice(
  volume,
  slice_index = NULL,
  frame = 1L,
  axis = 1L,
  rotation = 0L,
  flip = NULL
)
```

Arguments

`volume`	a 3D or 4D image volume. Note that empty dimensions will be dropped before any processing, and the remaining volume must have 3 or 4 dimensions.
`slice_index`	positive integer or vector of positive integers, the index into the slices (for the axis). A *slice* in the sense of this function is any 2D image plane extracted from the 3D volume (no matter the axis). If NULL, the slice in the middle of the volume is used. One can pass the magic character string 'all' to use all slice indices along the axis.
`frame`	positive integer, optional. The frame (time point) to use, only relevant for 4D volumes. The last (i.e. 4th) dimension is assumed to be the time dimension in that case.
`axis`	positive integer, the axis to use when indexing the slices. Defaults to 1.

`rotation`	integer, rotation in degrees. Defaults to 0 (no ratation). Must be a multiple of 90L if given.
`flip`	NULL or one of the character strings 'vertically' or 'horizontally'. Note that flipping *horizontally* means that the image will be mirrored along the central *vertical* axis. If ‘NULL‘ is passed, nothing is flipped. Flipping occurs after rotation.

Value

slice data. If ‘slice_index‘ is a scalar, a numerical 2D matrix (a 2D image from the stack). Otherwise, a numerical 3D array that contains the selected 2D images.

See Also

Other volume utility: `vol.boundary.box()`, `vol.imagestack()`, `vol.merge()`, `vol.overlay.colors.from.activation`
`vol.planes()`

`vol.vox.from.crs` *Compute R voxel index for FreeSurfer CRS voxel index.*

Description

Performs a vox2vos transform from FreeSurfer to R indices.

Usage

```
vol.vox.from.crs(fs_crs, add_affine = FALSE)
```

Arguments

`fs_crs`	integer vector of length 3, Freesurfer indices for column, row, and slice (CRS).
`add_affine`	logical, whether to add 1 to the output vector as the 4th value

Value

the R indices into the volume data for the given FreeSurfer CRS indices

Examples

```
# Get voxel intensity data on the command line, based
#  on the FreeSUrfer (zero-based) CRS voxel indices:
#  `mri_info --voxel 127 100 100 ~/data/tim_only/tim/mri/brain.mgz`
# (the result is: 106.0)
#
# That should be identical to:
# our_crs = vol.vox.from.crs(c(127,100,100), add_affine = FALSE);
# brain$data[our_crs[1], our_crs[2], our_crs[3]];   # gives 106
```

`volvis.contour` *Visualize contour of a volume.*

Description

Compute a smoothed surface from the voxel intensities in the given volume and render it. Requires the ‘misc3d‘ package to be installed, which is an optional dependency.

Usage

```
volvis.contour(volume, level = 80, show = TRUE, frame = 1L, color = "white")
```

Arguments

`volume`	a 3D brain volume
`level`	numeric, intensity threshold for the data. Voxels with intensity value smaller than ‘level‘ will be ignored when creating the contour surface.
`show`	logical, whether to display the triangles. Defaults to ‘TRUE‘.
`frame`	integer, the frame to show in case of a 4D input volume. Can also be the character string ’all’ to draw the contents of all frames at once. Useful to plot white matter tracts from DTI data, where each tract is stored in a different frame.
`color`	the color to use when plotting. Can be a vector of colors when plotting all frames of a 4D image (one color per frame).

Value

the rendered triangles (a ‘Triangles3D‘ instance) with coordinates in surface RAS space if any, ‘NULL‘ otherwise. This will be a list if you pass a 4D volume and select ’all’ frames.

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   brain = subject.volume(subjects_dir, 'subject1', 'brain');
   # Plot all voxels of the brain:
   volvis.contour(brain);

## End(Not run)
```

`volvis.lb`

Show continuous 3D voxel/volume data as a lightbox, optionally with a background brain volume and colormap.

Description

This function is the main way to visualize 3D volume images that contain raw MRI scans or statistical results.

Usage

```
volvis.lb(
  volume,
  background = NULL,
  colFn = viridis::viridis,
  colortable = NULL,
  no_act_source_value = 0,
  bbox_threshold = NULL,
  bbox_of_volume = TRUE,
  ...
)
```

Arguments

`volume`	numerical 3D array of per-voxel data, typically activation data, a raw MRI image, or a segmentation to show. Can also be a filename if the file can be loaded as such a volume with `read.fs.volume`.
`background`	numerical 3D array or 3D array of color strings, the background volume. Typically a raw brain volume. Dimensions and space must match those of the 'volume' for an array. Can also be a single file name as a character string. Can also be a single color name, like '#FEFEFE' but the string then must start with '#' (color names like 'red' are not allowed, they would be treated as file names). If a color string, be sure to use the `...` parameter to set the same color as `background_color` for the tiles.
`colFn`	a colormap function, passed to `vol.overlay.colors.from.activation` and used as colormap for the 'volume' data. Pass NULL to derive gray-scale values from the raw data (only recommended with single-color backgrounds). Note that the colormap is not used for the the background data (if any), which will be shown in grayscale (unless it is a 3D array of color strings).
`colortable`	optional, only makes sense for categorical 'volume' data like segmentations. If not NULL, a colortable as returned by `read.fs.colortable`, or a character string representing a path to a colortable file (like `"FREESURFER_HOME/FreeSurferColorLUT.txt"]`).
`no_act_source_value`	numerical value, passed to `vol.overlay.colors.from.activation`. Specifies the value which is treated as transparent in the 'volume' parameter data (where you will see the background). If you need more control, e.g., you want to treat

	one or morge ranges of values as NA, you should load the 'volume' data first, modify it as needed, as pass it to this function afterwards. Set this parameter to `NULL` to disable it. Only for 'colFn', ignored if a 'colortable' is used.
`bbox_threshold`	numerical scalar, passed on to `vol.merge`. If set, voxels with intensities smaller than this threshold will be dropped at the outside of the image. If `bbox_of_volume` parameter is `TRUE` (the default), this threshold applies to the 'volume', otherwise to the 'background'. Set to `NULL` to disable bounding box and show the full image.
`bbox_of_volume`	logical, whether the bounding box is computed on the volume (foreground), which typically is what you want. Leave alone if in doubt.
`...`	extra parameters to be passed to `volvis.lightbox`, can be used to select specific slices, set the `backgroud_color` for the border between and around the image tiles, etc.

Note

This function should be preferred over manually calling `volvis.lightbox`.

See Also

Other volume visualization: `volvis.lightbox()`

Examples

```
## Not run:
volume = subject.volume(subjects_dir, subject_id, 'brain');
volvis.lb(volume);
volvis.lb("~/study1/subject1/mri/brain.mgz");
volvis.lb("~/study1/subject1/mri/brain.mgz", bbox_threshold = 1L);
volvis.lb("~/study1/subject1/mri/brain.mgz", background = "~/data/study1/subject1/mri/T1.mgz");
volvis.lb("~/study1/subject1/mri/brain.mgz", background = "#FEFEFE", background_color="#FEFEFE");
ct = file.path(find.freesurferhome(mustWork = T), "FreeSurferColorLUT.txt"); # ct = "color table"
volvis.lb("~/study1/subject1/mri/aseg.mgz", background="~/study1/subject1/mri/T1.mgz",
 colortable = ct, colFn=NULL, axis=2L);
volvis.lb("~/study1/subject1/mri/aseg.mgz", background = "~/study1/subject1/mri/T1.mgz",
 colortable = ct, colFn=NULL, bbox_threshold = 0);

## End(Not run)
```

`volvis.lightbox` *Draw a lightbox view from volume slices.*

Description

A lightbox is a single image that holds a set of subimages, arranged in a grid. The images can have a small border or spacing between them. Consecutive subimages will be appear the same row of the grid.

If overlay_colors are given, the volume will be used as the background, and it will only be visible where overlay_colors has transparency.

Usage

```
volvis.lightbox(
  volume,
  slices = -5,
  axis = 1L,
  per_row = 5L,
  per_col = NULL,
  border_geometry = "5x5",
  background_color = "#000000",
  arrange_single_image = FALSE
)
```

Arguments

`volume`	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range '[0, 1]'.
`slices`	slice index definition. If a vector of integers, interpreted as slice indices. If a single negative interger '-n', interpreted as every 'nth' slice, starting at slice 1. The character string 'all' or the value 'NULL' will be interpreted as *all slices*.
`axis`	positive integer in range 1L..3L, the axis to use.
`per_row`	positive integer, the number of subimages per row in the output image. If 'NULL', automatically computed from the number of slices and the 'per_col' parameter.
`per_col`	positive integer, the number of subimages per column in the output image. If 'NULL', automatically computed from the number of slices and the 'per_row' parameter.
`border_geometry`	string, a geometry string passed to `magick::image_border` to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
`background_color`	string, a valid ImageMagick color string such as "white" or "#000080". The color to use when extending images (e.g., when creating the border). Defaults to black.
`arrange_single_image`	logical, whether to apply the given arrangement (from parameters 'per_row' and 'per_column') even if a single slice (a 2D image) is passed as 'volume'. Defaults to FALSE, which prevents that background tiles are added to fill the row up to

‘per_row‘ images. This also prevents the border from getting added to a single image, so all you see is the raw image. Set to ‘TRUE‘ if you want to arrange even a single image in a row with a border.

Value

a magick image instance

Note

You should, in most cases, not call this function directly. Use `volvis.lb` instead, which has a more intuitive interface.

See Also

`volvis.lb`

Other volume visualization: `volvis.lb()`

`volvis.voxels` *Voxel-based visualization of volume mask at surface RAS positions.*

Description

Plots a 3D box at every *foreground* voxel in the given volume. All voxels which do not have their intensity value set to ‘NA‘ are considered *foreground* voxels. The locations at which to plot the voxels is computed from the voxel CRS indices using the FreeSurfer `vox2ras_tkr` matrix. This means that the position of the rendered data fits to the surface coordinates (in files like ‘surf/lh.white‘), and that you can call this function while an active surface rendering window is open (e.g., from calling `vis.subject.morph.native`), to superimpose the surface and volume data. **On coloring the voxels** (using *rgl materials*): Note that you can call this function several times for the active plot, and color the voxels differently by passing different material properties in each call. Alternatively, check the ‘voxelcol‘ parameter.

Usage

```
volvis.voxels(volume, render_every = 1, voxelcol = NULL, ...)
```

Arguments

`volume`	numeric 3d array, voxels which should not be plotted must have value ‘NA‘. Take care not to plot too many.
`render_every`	integer, how many to skip before rendering the next one (to improve performance and/or see deeper structures). Use higher values to see a less dense representation of your data that usually still allows you to see the general shape, but at lower computational burden. Set to 1 to render every (foreground) voxel.

`voxelcol`	character string or a *voxel coloring*. A *voxel coloring* can be specified in three ways: 1) the string 'from_intensity' will compute colors based on the intensity values of the foreground voxels in the volume, applying normalization of the intensity values if needed. 2) an array of RGB color strings: will be used to retrieve the colors for all foreground vertices, at their CRS indices. 3) A vector with length identical to the number of foreground voxels in the volume: will be applied directly. Obvisouly, you should not pass a color material parameter (see '...') when using this.
`...`	material properties, passed to `triangles3d`. Example: `color = "#0000ff"`, `lit=FALSE`.

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   brain = subject.volume(subjects_dir, 'subject1', 'brain');
   # Plot all voxels of the brain:
   brain[which(brain==0L, arr.ind = TRUE)] = NA;  # mark background
   brain = vol.hull(brain); # remove inner triangles
   volvis.voxels(brain);

## End(Not run)
```

`vox2ras_tkr` *The FreeSurfer default vox2ras_tkr matrix.*

Description

Applying this matrix to a FreeSurfer CRS index of a conformed volume will give you the RAS coordinates of the voxel in surface coordinates, i.e., in the coordinates used in surface file like 'lh.white'. The central voxel is 127,127,127 when using zero-based indices (or 128,128,128 when using one-based indices), meaning its surface RAS coordinates are 0.0, 0.0, 0.0. The returned matrix is the inverse of the 'ras2vox_tkr' matrix.

Usage

```
vox2ras_tkr()
```

Value

numeric 4x4 matrix, the FreeSurfer vox2ras_tkr matrix.

See Also

Other surface and volume coordinates: `ras2vox_tkr()`

Examples

```
# Compute surface RAS coordinate of voxel with CRS (0L, 0L, 0L):
vox2ras_tkr() %*% c(0, 0, 0, 1);
# Show that voxel with CRS (128,128,128) is at the
#  origin (0.0, 0.0, 0.0) of the surface RAS coordinate system:
(vox2ras_tkr() %*% c(128, 128, 128, 1))[1:3];
```

`write.group.morph.standard` *Write standard space group data to a standard FreeSurfer directory stucture.*

Description

Write standard space group data to a standard FreeSurfer directory stucture.

Usage

```
write.group.morph.standard(
  subjects_dir,
  subjects_list,
  data,
  measure_name,
  hemi = "both",
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  create_dirs = TRUE,
  template_lh_numverts = NULL
)
```

Arguments

`subjects_dir`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list`	vector of strings. The subject identifiers.
`data`	the data matrix
`measure_name`	character string, the data part of the generated file names, e.g., 'thickness' or 'area'.
`hemi`	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`fwhm`	string. Smoothing as string, e.g. '10' or '25'.
`template_subject`	string. Template subject name, defaults to 'fsaverage'.

`format`	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
`create_dirs`	logical, whether to create missing (sub) directories which occur in the 'filepaths'.
`template_lh_numverts`	positive integer, the vertex count of the left hemi of the template subject, only used if 'hemi' is 'both'. If hemi is both and this is unspecified (left at the default value `NULL`), the template subject needs to exist in the 'subjects_dir' to determine the vertex count of the left hemisphere, so that the data can be split into the `lh` and `rh` files at the correct index.

See Also

`write.group.morph.standard.sf` and `write.group.morph.standard.mf`

Examples

```
## Not run:
dm = matrix(rnorm(325684 * 6, 5.0, 1.2), ncol = 6);
subjects = paste("subject", seq(6), sep="");
write.group.morph.standard("/tmp/groupdata", subjects, dm,
  "rand", template_lh_numverts = 325684 / 2);

## End(Not run)
```

`write.group.morph.standard.mf` *Write per-vertex standard space data for a group of subjects to given file names.*

Description

Write per-vertex standard space data for a group of subjects to given file names.

Usage

```
write.group.morph.standard.mf(
  filepaths_hl,
  data_hl,
  format = "auto",
  create_dirs = TRUE
)
```

Arguments

`filepaths_hl`	`hemilist` of vectors of character strings, the full paths to the output files, including file names and extension.

`data_hl`	`hemilist` of numerical matrix or data.frame, the morph data for the hemispheres of all subjects. See `groupmorph.split.hemilist` to get this format if you have a full matrix or dataframe for both hemispheres.
`format`	character string, a valid format spec for `freesurferformats::write.fs.morph`, e.g., "auto" to derive from filename, "mgh", "mgz", "curv" or others.
`create_dirs`	logical, whether to create missing (sub) directories which occur in the 'filepaths'.

See Also

`write.group.morph.standard.sf` to write the data to a single stacked file instead.

`write.group.morph.standard.sf`

Reshape and write combined per-vertex data for a group to a single MGH file.

Description

Write morphometry data for a group into a single MGH or MGZ file. In neuroimaging, the first 3 dimensions in the resulting 4D volume file are space, and the 4th is the time/subject dimension.

Usage

```
write.group.morph.standard.sf(filepath, data)
```

Arguments

`filepath`	character string, path to the target file, should end with '.mgh' or '.mgz'.
`data`	numerical 2D matrix, with the rows identifying the subjects and the columns identifying the vertices.

Note

The file will contain no information on the subject identifiers. The data can be for one or both hemispheres. See `group.morph.standard.sf` to read the data back into R.

Examples

```
## Not run:
# create per-vertex data for 5 subjects.
mat = matrix(rnorm(5 * 163842, 3.0, 0.5), nrow=5, ncol = 163842);
fsbrain::write.group.morph.standard.sf("~/group_pvd.mgz", mat);

## End(Not run)
```

`write.region.aggregated`

Write data aggregated over regions to morphometry file for group.

Description

Given an atlas, a subjects list and a measure, aggregate the measure over each region (e.g., mean) and write an output morphometry file in which the value for all region vertices is set to the aggregated value.

Usage

```
write.region.aggregated(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  agg_fun = mean,
  outfile_morph_name = "",
  format = "mgz"
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subjects_list,`	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
`measure,`	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`agg_fun,`	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
`outfile_morph_name,`	string. The measure part of the output file name. E.g., 'agg_thickness' will write the file '<subject>/surf/<hemi>.agg_thickness.mgh'. Defaults to 'agg_<measure>'.
`format,`	string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

See Also

Other output functions: `write.region.values.fsaverage()`, `write.region.values()`

`write.region.values` *Write one value per atlas region for a subject.*

Description

Given an atlas and a list that contains one value for each atlas region, write a morphometry file in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

Usage

```
write.region.values(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region_value_list,
  outfile_morph_name,
  format = "mgz",
  do_write_file = TRUE,
  output_path = NULL,
  value_for_unlisted_regions = NaN
)
```

Arguments

`subjects_dir,`	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
`subject_id,`	string. The subject identifier
`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`region_value_list,`	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.
`outfile_morph_name,`	string. The measure part of the output file name. E.g., 'agg_thickness' will write the file '<subject>/surf/<hemi>.agg_thickness.mgh'.
`format,`	string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

`do_write_file`,	logical. Whether to write the data to a file on the disk. If FALSE, the data are only returned (and the outfile_morph_name parameter gets ignored). Default to TRUE.
`output_path`	string, path to the output directory. If omitted, defaults to the 'surf' directory of the subject (i.e., '<subjects_dir>/<subject_id>/surf/').
`value_for_unlisted_regions`,	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.

Value

a named list with the following entries: "data": a vector containing the data. "file_written": string, path to the file that was written, only exists if do_write = TRUE.

See Also

Other output functions: `write.region.aggregated()`, `write.region.values.fsaverage()`

Examples

```
## Not run:
   fsbrain::download_optional_data();
   subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
   region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
   write.region.values(subjects_dir, 'subject1', 'lh', 'aparc',
    region_value_list, 'pvalues.mgz', do_write_file = FALSE);

## End(Not run)
```

`write.region.values.fsaverage`

Write one value per atlas region for a template subject.

Description

Given an atlas and a list that contains one value for each atlas region, write a morphometry file in which all region vertices are assigned the value. Can be used to plot stuff like p-values or effect sizes onto brain regions.

Usage

```
write.region.values.fsaverage(
  hemi,
  atlas,
  region_value_list,
  output_file,
```

```
  template_subject = "fsaverage",
  template_subjects_dir = NULL,
  show_freeview_tip = FALSE,
  value_for_unlisted_regions = NaN
)
```

Arguments

`hemi,`	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
`atlas,`	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
`region_value_list,`	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.
`output_file,`	string or 'NULL'. Path of the output file, including file name and extension. The format is determined from the (absence of a) file extension. If NULL, no file will be written.
`template_subject`	string, template subject name. Defaults to 'fsaverage'.
`template_subjects_dir`	string, the path to the subjects directory containing the template subject directory. If this is 'NULL', the function will try to find it using the environment, see the function `find.subjectsdir.of` for details. Defaults to NULL.
`show_freeview_tip`	logical, whether to print the freeview command on howto use the overlay to the console. (Only happens if the output_file is not 'NULL'.)
`value_for_unlisted_regions,`	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.

Value

a named list with the following entries: "data": a vector containing the data. "file_written": string, path to the file that was written, only exists if do_write = TRUE.

See Also

Other output functions: `write.region.aggregated()`, `write.region.values()`

Index