Package ‘fsbrain’

April 15, 2020

Type Package
Title Managing and Visualizing Brain Surface Data
Version 0.1.0
Maintainer Tim Schäfer <ts+code@rcmd.org>
Description Provides high-level access to ‘FreeSurfer’ [http://freesurfer.net/] neuroimaging data on the level of subjects and groups. Load morphometry data, surfaces and brain parcelations based on atlases. Mask data using labels, load data for specific atlas regions only, and visualize the results directly in ‘R’.
License MIT + file LICENSE
Encoding UTF-8
LazyData true
URL https://github.com/dfsp-spirit/fsbrain
BugReports https://github.com/dfsp-spirit/fsbrain/issues
Imports reshape, freesurferformats (>= 0.1.7), pkgfilecache (>= 0.1.1), dplyr, rgl, squash, fields, data.table
Suggests knitr, rmarkdown, testthat (>= 2.1.0), magick, misc3d
VignetteBuilder knitr
RoxygenNote 7.0.0
NeedsCompilation no
Author Tim Schäfer [aut, cre] (<https://orcid.org/0000-0002-3683-8070>)
Repository CRAN
Date/Publication 2020-04-15 21:00:07 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphablend</td>
<td>4</td>
</tr>
<tr>
<td>annot.outline</td>
<td>5</td>
</tr>
<tr>
<td>apply.label.to.morphdata</td>
<td>6</td>
</tr>
<tr>
<td>apply.labeldata.to.morphdata</td>
<td>7</td>
</tr>
</tbody>
</table>
apply.transform .................................................. 7
arrange.brainview.images .................................... 8
brainviews ......................................................... 9
clip.data .......................................................... 10
collayer.bg ....................................................... 10
collayer.bg.atlas ............................................... 11
collayer.bg.meancurv ......................................... 12
collayer.bg.sulc ............................................... 13
collayer.from.annot ............................................ 14
collayer.from.annotdata ....................................... 15
collayer.from.mask.data ....................................... 16
collayer.from.morphlike.data ................................ 16
collayers.merge ............................................... 17
coloredmesh.from.mask ....................................... 18
coloredmesh.from.morph.native ................................ 19
coloredmesh.plot.colorbar.separate ......................... 20
coloredmeshes.from.color .................................... 22
colorlist.brain.clusters ....................................... 23
combine.colorbar.with.brainview.animation ................. 23
combine.colorbar.with.brainview.image ..................... 24
cube3D.tris ...................................................... 25
cubes3D.tris ..................................................... 26
delete_all_optional_data ...................................... 27
desaturate ....................................................... 27
download_fsaverage ............................................. 28
download_optional_data ....................................... 29
face.edges ....................................................... 29
find.freesurferhome .......................................... 30
find.subjectsdir.of ............................................ 30
flip3D ........................................................... 31
fs.coloredmesh ................................................. 32
fup ............................................................... 32
get.atlas.region.names ....................................... 33
get.view.angle.names ......................................... 34
getIn ............................................................. 34
get_optional_data_filepath .................................... 35
group.agg.atlas.native ........................................ 36
group.agg.atlas.standard ...................................... 37
group.annot ..................................................... 39
group.concat.measures.native ................................ 40
group.concat.measures.standard ............................. 41
group.label ...................................................... 43
group.label.from.annot ....................................... 44
group.morph.agg.native ....................................... 45
group.morph.agg.standard .................................... 47
group.morph.native ............................................ 48
group.morph.standard ......................................... 49
group.morphlike.data ......................................... 51
topics documented:

- group.multimorph.agg.standard ........................................ 53
- hasIn ............................................................................. 54
- hemilist.derive.hemi ....................................................... 55
- hemilist.get.combined.data ............................................... 55
- hemilist.unwrap ............................................................... 56
- hemilist.wrap ................................................................. 56
- is.fs.coloredmesh ............................................................. 57
- is.fs.coloredvoxels ............................................................ 57
- is.hemilist ........................................................................ 58
- label.border ..................................................................... 58
- label.from.annotdata ......................................................... 59
- label.to.annot ................................................................. 60
- labeldata.from.mask .......................................................... 61
- list_optional_data .............................................................. 62
- mask.from.labeldata.for.hemi ............................................. 63
- mesh.vertex.neighbors ....................................................... 64
- print.fs.coloredmesh ......................................................... 65
- print.fs.coloredvoxels ....................................................... 66
- ras2vox_tkr .................................................................... 66
- read.colorcsv .................................................................. 67
- read.md.demographics ....................................................... 67
- read.md.subjects ............................................................... 68
- regions.to.ignore ............................................................. 69
- report.on.demographics ...................................................... 70
- rglvoxels ......................................................................... 71
- rotate3D ......................................................................... 72
- spread.values.over.annot .................................................. 73
- spread.values.over.hemi ..................................................... 74
- spread.values.over.subject ............................................... 75
- subject.annot ................................................................. 77
- subject.atlas.agg .............................................................. 78
- subject.filepath.morph.native ......................................... 79
- subject.filepath.morph.standard ....................................... 80
- subject.label ................................................................. 81
- subject.label.from.annot .................................................. 82
- subject.mask ................................................................. 84
- subject.morph.native ...................................................... 85
- subject.morph.standard .................................................... 86
- subject.surface .............................................................. 88
- subject.volume .............................................................. 89
- vis.color.on.subject ......................................................... 90
- vis.coloredmeshes ............................................................ 91
- vis.data.on.fsaverage ....................................................... 92
- vis.data.on.subject .......................................................... 94
- vis.labeldata.on.subject .................................................. 96
- vis.mask.on.subject .......................................................... 98
- vis.path.along.verts ........................................................ 100
- vis.region.values.on.subject ............................................. 101
Perform alpha blending for pairs of RGBA colors.

Description

Implements the *over* alpha blending operation.

Usage

```r
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
)

Arguments

annotdata  an annotation, as returned by functions like subject.annot.
surface_mesh  brain surface mesh, as returned by functions like subject.surface or 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.

Value

vector of colors, one color for each mesh vertex
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.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.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 coordinates 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 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

Combine several brainview images into a new figure.

Usage

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

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 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).

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 = list(),
rglactions = list(),
style = "default",
draw_colorbar = FALSE
)

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 'rgloptions' parameter for the colorbar to show up. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.

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 hemi list.
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.

Examples

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

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")
### collayer.bg.atlas

**collayer.bg.atlas**

Compute atlas or annotation surface color layer.

**Description**

Compute atlas or annotation surface color layer.

**Usage**

```r
collayer.bg.atlas(
  subjects_dir, 
  subject_id, 
  hemi = "both", 
  atlas = "aparc", 
  grayscale = !outline, 
  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, one of 'curv', 'sulc', or 'aparc'. If this is already a colorlayer in a hemilist, it will be returned as-is.
- `grayscale`, `outline`, and `outline_surface` are logical arguments that control the appearance of the color layer.
collayer.bg.meancurv

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)
)

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.data()`,
`collayers.merge()`
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**

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

**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.data()`, `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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subjects_dir</td>
<td>character string, the FreeSurfer SUBJECTS_DIR.</td>
</tr>
<tr>
<td>subject_id</td>
<td>character string, the subject identifier.</td>
</tr>
<tr>
<td>hemi</td>
<td>character string, one of 'lh', 'rh', or 'both'.</td>
</tr>
<tr>
<td>atlas</td>
<td>character string, the atlas name. E.g., &quot;aparc&quot;, &quot;aparc.2009s&quot;, or &quot;aparc.DKTatlas&quot;. Used to construct the name of the annotation file to be loaded.</td>
</tr>
</tbody>
</table>

**Value**

named hemi list, each entry is a vector of color strings, one color per surface vertex. The coloring represents the atlas data.
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.annotdata(), collayer.from.mask.data(), 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**

```r
collayer.from.mask.data(
  lh_data = NULL,
  rh_data = NULL,
  makecmap_options = list(colFn = squash::rainbow2)
)
```

**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

```r
collayer.from.morphlike.data(
    lh_morph_data = NULL,
    rh_morph_data = NULL,
    makecmap_options = list(colFn = squash::jet),
    return_map = 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_map`: logical, whether to add the colormap as entry `map` 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

```r
collayers.merge(collayers, opaque_background = "#FFFFFF")
```
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",
  colormap = NULL,
  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.
coloredmesh.from.morph.native

Description
Create a coloredmesh from native space morphometry data.

Usage
coloredmesh.from.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  surface = "white",
  colormap = NULL,
  clip = NULL,
  cortex_only = FALSE,
  make cmap_options = list(colFn = squash::jet)
)
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".
- **colormap**: a colormap function. See the squash package for some colormaps. Defaults to jet.
- **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'.

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 = TRUE,
  image.plot_extra_options = list(horizontal = TRUE),
  png_options = NULL,
  silent = FALSE
)

Arguments

coloredmeshes list of coloredmeshes. A coloredmesh is a named list as returned by the 'col-
oredmesh.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 leg-
end 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"="outfile.png","wid
Defaults to NULL, which will not save anything.
silent logical, whether to suppress messages. Defaults to 'FALSE'.

Value

named list with the following entries: "full_data": the combined data from all coloredmeshes (can
be NULL if they have no data). "colormap": the colormap function from the coloredmeshes (can
be NULL if they have none).

See Also

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

Examples

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]",

coloredmeshes.from.color

Create coloredmeshes for both hemis using pre-defined colors.

Description

Create coloredmeshes for both hemis using pre-defined colors.

Usage

```r
coloredmeshes.from.color(
  subjects_dir, subject_id, color_data, hemi,
  surface = "white", metadata = 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.
- `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'.
**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

---

**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
)
```
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 `image_composite`. Allows you to shift the location of the colorbar in the final image.

extend_brainview_img_height_by  
inger 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.

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**

```r
combine.colorbar.with.brainview.image(
  brainview_img,
  colorbar_img,
  output_img,
  offset = "+0+0",
  extend_brainview_img_height_by = 0L,
  silent = FALSE,
  allow_colorbar_shrink = TRUE
)
```
cube3D.tris

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 that including the file extension
- offset: offset string passed to 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.

See Also

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

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. Obviously, 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

```r
# 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

```r
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

```r
# 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.

---

**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 weights, 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_optional_data

Download optional data for this package if required.

Description

Ensure that the optional 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.

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.
find.subjectsdir.of

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

```r
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.

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

```r
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

A 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.

---

**flip3D**

*Flip a 3D array along an axis.*

Description

Flip the slice of an 3D array horizontally or vertically along an axis. This leads to an output array with identical dimensions.

Usage

```r
flip3D(volume, axis = 1L, how = "horizontally")
```

Arguments

- `volume`: a 3D image volume
- `axis`: positive integer in range 1L..3L or an axis name, the axis to use.
- `how`: character string, one of 'horizontally' or 'vertically'. How to flip the 2D slices. Note that flipping *horizontally* means that the image will be mirrored along the central *vertical* axis.

Value

A 3D image volume, flipped around the axis. The dimensions are identical to the dimensions of the input image.

See Also

Other volume math: `rotate3D()`
fs.coloredmesh constructor

Description

fs.coloredmesh constructor

Usage

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

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
- hemi: character string, one of ‘lh’ or ‘rh’
- render: logical, whether to render this mesh during visualization
- metadata: optional, named list containing metadata

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.

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.
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()
Examples

```r
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);
```

---

`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**

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

**Arguments**

- `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.
- `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 lateral
  for the left hemisphere. `rh`: medial and lateral for the right hemisphere.

**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**

```r
getIn(named_list, listkeys)
```
**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.

**Examples**

```r
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
```
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

\[
\text{group.agg.atlas.native(}
\begin{align*}
\text{subjects_dir}, \\
\text{subjects_list}, \\
\text{measure}, \\
\text{hemi}, \\
\text{atlas}, \\
\text{agg_fun} = \text{mean}, \\
\text{cache_file} = \text{NULL}
\end{align*}
\]

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.
- `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(), 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()

Examples

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);

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

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,
group.agg.atlas.standard

```r
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.
group.annot

See Also

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

Examples

```r
## 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)
```

```
---

## 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

```r
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()

Examples

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");
group.concat.measures.standard

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. 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 concatenation functions: group.concat.measures.standard()

Examples

```r
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");
```

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

```r
group.concat.measures.standard(
  subjects_dir,
  subjects_list,
  measures,
```
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 morphology measures. E.g., c("area", "thickness"). Used to construct the names of the morphology 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 morphology 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 concatenation of the measure, "_fwhm", and the fwhm for that measure. WARNING: This dataframe can get large if you have many subjects.

See Also

Other concatenation functions: group.concat.measures.native()

Examples

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", "rh");
Cross-references

- `labeldata.from.mask`
- `mask.from.labeldata.for.hemi`
- `subject.label`
Examples

```r
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");
```

---

**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

```r
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()
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. For 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

```r
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");
```
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

```r
group.morph.agg.standard(
  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 = mean,  # 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 = "fsaverage",  # string. The template subject to use for constructing the file names.
  format = "mgh",  # string. The format of the file.
  cast = TRUE,  # logical. Whether to cast the data to a numeric type.
  cortex_only = FALSE,  # logical. Whether to only aggregate data from the cortical surface.
  agg_fun_extra_params = NULL)  # list. Extra parameters to pass to the aggregation function.
```

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 note 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. For 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.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
)
group.morph.standard

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

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");

---

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

```r
group.morph.standard(  
  subjects_dir,  
  subjects_list,  
  measure,  
  hemi,  
  fwhm = "10",  
  template_subject = "fsaverage",  
  format = "mgh",  
  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.
- `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.

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

```r
## 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.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**

```r
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`. 
group.multimorph.agg.native

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 logical, 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. For 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

```r
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);
```
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. For 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()

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

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

hemilist.derive.hemi

Derive 'hemi' string from the data in a hemilist

Description

Derive 'hemi' string from the data in a hemilist

Usage

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

Arguments

- `hemilist`: hemilist, an existing hemilist

Value

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

hemilist.get.combined.data

Get combined data of hemi list

Description

Get combined data of hemi list

Usage

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

Arguments

- `hemi_list`: named list, can have entries 'lh' and/or 'rh'

Value

the data combined with `c`, or NULL if both entries are NULL.
**hemilist.unwrap**  
*Unwrap hemi data from a named hemi list.*

**Description**  
Unwrap hemi data from a named hemi list.

**Usage**

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

**Arguments**

- **hemi_list**: named list, can have entries 'lh' and/or 'rh'
- **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

---

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

**Description**  
Wrap data into a named hemi list.

**Usage**

```r
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**

named list, with the 'data' in the name given by parameter 'hemi'
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" 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’.

**Usage**
is.hemilist(x)

**Arguments**
- **x**  
  any R object

**Value**
whether ‘x’ is a 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.
**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**

```r
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()`

---

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

**Description**

Merge several labels and a colortable into an annotation.

**Usage**

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

---

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.
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

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);

---

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**

```r
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()
**print.fs.coloredvoxels**

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

**Description**

Print description of fs.coloredvoxels (S3).

**Usage**

```r
## 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

---

**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**

```r
ras2vox_tkr()
```

**Value**

numeric 4x4 matrix, the FreeSurfer ras2vox_tkr matrix.

**See Also**

Other surface and volume coordinates: `vox2ras_tkr()`

**Examples**

```r
# 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**

```r
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**

```r
read.md.demographics(
  demographics_file,
  column_names,
  header = TRUE,
  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 utils::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 [fsbrain::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: read.md.subjects()

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,
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.
regions.to.ignore

Usage

read.md.subjects(subjects_file, header = FALSE)

Arguments

subjects_file,
string. The path to the file.
header, logical. Whether the file starts with a header line. Defaults to FALSE.

Value

a vector of strings. The subject IDs.

See Also

Other metadata functions: read.md.demographics()

Examples

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

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

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 in-
    cluded in the report. Defaults to FALSE.

Value

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

Draw 3D boxes at locations using rgl.

Description

Draw 3D boxes at all given coordinates using rgl, analogous to \texttt{rgl.spheres}. Constructs the coordinates for triangles making up the boxes, then uses \texttt{triangles3d} to render them.

Usage

\begin{verbatim}
rglvoxels(centers, r = 1, voxelcol = NULL, ...)
\end{verbatim}

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>centers</td>
<td>numerical matrix with 3 columns. Each column represents the x, y, z coordinates of a center at which to create a cube.</td>
</tr>
<tr>
<td>r</td>
<td>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'.</td>
</tr>
<tr>
<td>voxelcol</td>
<td>vector of rgb color strings for the individual voxels. Its length must be identical to nrow(centers) if given.</td>
</tr>
<tr>
<td>...</td>
<td>material properties, passed to \texttt{triangles3d}. Example: color = &quot;#0000ff&quot;,lit=FALSE.</td>
</tr>
</tbody>
</table>

Value

list of \texttt{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

\begin{verbatim}
# Plot a 3D cloud of 500 red voxels:
centers = matrix(rnorm(500*3)*100, ncol=3);
rglvoxels(centers, voxelcol="red");
\end{verbatim}
rotate3D  

Rotate a 3D array in 90 degree steps.

Description

Rotate a 3D array in 90 degree steps along an axis. This leads to an array with different dimensions.

Usage

rotate3D(volume, axis = 1L, degrees = 90L)

Arguments

- **volume**: a 3D image volume
- **axis**: positive integer in range 1L..3L or an axis name, the axis to use.
- **degrees**: integer, must be a (positive or negative) multiple of 90L.

Value

a 3D image volume, rotated around the axis. The dimensions may or may not be different from the input image, depending on the rotation angle.

See Also

Other volume math: `flip3D()`

Examples

```r
# Load data
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain') / 255;
# Show a lightbox along the third axis. Note that the orientation in the
# visualization is not very intuitive: the brain lies on the side.
volvis.lightbox(brain, axis=3);
# Rotate the whole brain volume by 90 degrees clockwise along
# the third axis to fix the orientation:
volvis.lightbox(rotate3D(brain, axis=3, degrees = 90), axis=3);
```
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.
spread.values.over.hemi

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()

Examples

```r
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);
```

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

```r
spread.values.over.hemi(
  subjects_dir,
  subject_id,
  hemi,
  atlas,
  region_value_list,
  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.
spread.values.over.subject

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.

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
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()

Examples

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);

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,
spread.values.over.subject

    lh_region_value_list,
    rh_region_value_list,
    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

  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.

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()

Examples

  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);
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, entries 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 columns. 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.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 = 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 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.

Examples

fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot_lh = subject.annot(subjects_dir, "subject1", "lh", "aparc");
subject.filepath.morph.native

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()

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()

Examples

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);
subject.filepath.morph.standard

Construct filepath of standard space morphometry data file.

**Description**

Construct filepath of standard space morphometry data file.

**Usage**

```r
subject.filepath.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  warn_if_nonexistent = FALSE,
  error_if_nonexistent = FALSE
)
```
subject.label

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.
- **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.

Examples

```r
filepath_lh_thickness_std =
subject.filepath.morph.standard("/media/ext/data/study1", "subject1", "thickness", "lh", fwhm="25");
```

---

**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

```r
subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  return_one_based_indices = TRUE
)
```
subject.label.from.annot

Extract a region from an atlas annotation as a label for a subject.

**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'. You can also pass just the label (e.g., 'cortex'): if the string does not end with the suffix '.label', that suffix gets added automatically.
- **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.

**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**

```r
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cortex_lh = subject.label(subjects_dir, "subject1", "cortex.label", "lh");
```
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.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.
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

```r
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.

See Also

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

Arguments

- `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

```r
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);
```
subject.morph.standard

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()
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, hemi)

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.

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

```r
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');

lh_white = subject.surface(subjects_dir, "subject1", "white", "lh");
```
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

```r
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
```

---

### vis.color.on.subject

**Visualize pre-defined vertex colors on a subject.**

**Description**

Visualize pre-defined vertex colors on a subject.

**Usage**

```r
vis.color.on.subject(
  subjects_dir,
  vis_subject_id,
  color_lh,
  color_rh,
  surface = "white",
  views = c("t4"),
  rgloptions = list(),
  rglactions = 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.
- **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.
- **color_rh** vector of colors to visualize on the right hemisphere surface.
- **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.
**vis.coloredmeshes**

Visualize a list of colored meshes in a single scene.

**Description**

Visualize a list of colored meshes in a single scene.

**Usage**

```r
vis.coloredmeshes(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE
)
```

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: `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()`

**Examples**

```r
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] = "#0000ff";
color_rh[60001:90000] = "#00ff00";
color_rh[90001:120000] = "#ffff00";
color_rh[120001:150000] = "#00ffff";
vis.color.on.subject(subjects_dir, "subject1", color_lh, color_rh);
```
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

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

```r
vis.data.on.fsaverage(
  subjects_dir = NULL,
  vis_subject_id = "fsaverage",
  morph_data_lh,  
morph_data_rh,  
surface = "white",  
colormap = NULL,  
views = c("t4"),  
rgloptions = list(),
)```
vis.data.on.fsaverage

rglactions = list(),
draw_colorbar = FALSE,
makecmap_options = list(colFn = squash::jet),
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.

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".

colormap a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.

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)).

glactions named list. A list in which the names are from a set of pre-defined actions. The values can be used to set parameters for the action.

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 'rgloptions' parameter for the colorbar to show up. 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, see collayer.bg for valid strings.
vis.data.on.subject

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: 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.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.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,
  morph_data_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = squash::jet),
  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.

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".

colormap a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.

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.

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 'rgloptions' parameter for the colorbar to show up. 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, see collayer.bg for valid strings.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: 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.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

```r
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);
```

---

**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 to 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**

```r
vis.labeldata.on.subject(
  subjects_dir,
  vis_subject_id,
  lh_labeldata,
  rh_labeldata,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE,
  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.
- `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.
vis.labeldata.on.subject

surface string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".

colormap a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.

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.

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.

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 'rgloptions' parameter for the colorbar to show up. 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'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

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

Other visualization functions: 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.symmetric.data.on.subject(), vislayout.from.coloredmeshes()

Examples

```r
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'));

---

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 [fsbrain::mask.from.labeldata.for.hemi] to create and modify it. Check the example for this function.

**Usage**

```r
vis.mask.on.subject(
   subjects_dir, 
   vis_subject_id, 
   mask_lh, 
   mask_rh, 
   surface = "white", 
   colormap = NULL, 
   views = c("t4"), 
   rgloptions = list(), 
   rglactions = list(), 
   draw_colorbar = FALSE, 
   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.

- `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".
**vis.mask.on.subject**

- **colormap**: a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.
- **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.
- **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 'rgloptions' parameter for the colorbar to show up. 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'.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other mask functions: `coloredmesh.from.mask()`, `mask.from.labeldata.for.hemi()`

Other visualization functions: `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.symmetric.data.on.subject()`, `vislayout.from.coloredmeshes()`

**Examples**

```r
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);

---

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 (‘view=’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)

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".

path_vertex_indices
vector of vertex indices, the path. You will need to have it computed already. (This function does **not** compute geodesic paths. You can use it to visualize such a path though.)

See Also
Other surface mesh functions: face.edges(), label.border(), mesh.vertex.included.faces(), mesh.vertex.neighbors(), subject.surface()
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",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  value_for_unlisted_regions = NA,
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = grDevices::heat.colors),
  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.

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.

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".

colormap a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.
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.

value_for_unlisted_regions numerical scalar or ‘NaN’, the value to assign to regions which do not occur in the region_value_lists. Defaults to ‘NaN’.

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 ‘rgloptions’ parameter for the colorbar to show up. 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, see collayer.bg for valid strings.

Value list of coloredmeshes. The coloredmeshes used for the visualization.

See Also Other visualization functions: vis.color.on.subject()
vis.data.on.fsaaverage()
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.symmetric.data.on.subject()
vislayout.from.coloredmeshes()

Other region-based visualization functions: vis.subject.annot()

Examples

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.
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,
**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**

```r
vis.subject.annot(
  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.
  hemi = "both",  # string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
  surface = "white",  # string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
  views = c("t4"),  # 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 = list(),  # option list passed to par3d. Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
  rglactions = list(),  # 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.
  outline = FALSE  # logical, whether to draw an outline only instead of filling the regions. Defaults to 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.
- **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.
- **outline**: logical, whether to draw an outline only instead of filling the regions. Defaults to FALSE.
Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: 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.symmetric.data.on.subject(),
vislayout.from.coloredmeshes()

Other region-based visualization functions: vis.region.values.on.subject()

Examples

```r
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
vis.subject.annot(subjects_dir, "subject1", "aparc", "both");
```

---

**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**

```r
vis.subject.label(
    subjects_dir,
    subject_id,
    label,
    hemi,
    surface = "white",
    colormap = NULL,
    views = c("t4"),
    rgloptions = list(),
    rglactions = list(),
    draw_colorbar = FALSE,
    makecmap_options = list(colFn = squash::rainbow2, col.na = "#FFFFFF00"),
    map_to_NA = 0L,
    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.
- **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".
- **colormap** a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.
- **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.
- **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 'rlgloptions' parameter for the colorbar to show up. 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 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, pass 'NULL'.
- **bg** a background definition. Can be a surface color layer or a character string, see collayer.bg for valid strings.

**Value**

list of coloredmeshes. The coloredmeshes used for the visualization.

**See Also**

Other visualization functions: 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.symmetric.data.on.subject(), vislayout.from.coloredmeshes()
Other label functions: `apply.label.to.morphdata()`, `apply.labdata.to.morphdata()`, `subject.mask()`, `vis.labdata.on.subject()`

Examples

```r
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\");
```

---

**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**

```r
vis.subject.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi = "both",
  surface = "white",
  colormap = NULL,
  views = c("t4"),
  rgloptions = list(),
  rglactions = list(),
  draw_colorbar = FALSE,
  cortex_only = FALSE,
  style = "default",
  makecmap_options = list(colFn = squash::jet),
  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".

- **colormap**: a colormap function. **DEPRECATED**: use parameter `makecmap_options` instead.

- **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.

- **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 `rgloptions` parameter for the colorbar to show up. 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, see `collayer.bg` for valid strings.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `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.morph.native`
vis.subject.morph.standard

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.standard(
    subjects_dir, 
    subject_id, 
    measure, 
    hemi = "both", 
    fwhm = "10", 
    surface = "white", 
    template_subject = "fsaverage", 
    template_subjects_dir = NULL, 
    colormap = NULL, 
    views = c("t4"), 
    rgloptions = list(), 
    rglactions = list(), 
    draw_colorbar = FALSE, 
    cortex_only = FALSE, 
    makecmap_options = list(colFn = squash::jet), 
    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.

**fwhm**
string, smoothing setting. 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')`.

**colormap**
a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.

**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.

**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 'rgloptions' parameter for the colorbar to show up. 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 '?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, see collayer.bg for valid strings.
vis.symmetric.data.on.subject

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: 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.symmetric.data.on.subject(), vislayout.from.coloredmeshes()

Other morphometry visualization functions: vis.data.on.fsaverage(), vis.data.on.subject(), vis.subject.morph.native(), vis.symmetric.data.on.subject()

Examples

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*.

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,
  morph_data_rh,
  surface = "white",
  colormap = NULL,
  views = c("t4"),
)
rgloptions = list(),
rglactions = list(),
draw_colorbar = FALSE,
makecmap_options = list(colFn = squash::jet, symm = TRUE, col.na = "#FFFFFF00"),
map_to_NA = c(0),
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.

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".

colormap a colormap function. **DEPRECATED**: use parameter 'makecmap_options' instead.

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.

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 'rgloptions' parameter for the colorbar to show up. 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. 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, pass 'NULL'.

bg a background definition. Can be a surface color layer or a character string, see `collayer.bg` for valid strings.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `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()`, `vislayout.from.coloredmeshes()`

Other morphometry visualization functions: `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`

Examples

```r
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);
```
vislayout.from.coloredmeshes

Usage

vislayout.from.coloredmeshes(
  coloredmeshes,
  view_angles = get.view.angle.names(angle_set = "t4"),
  rgloptions = list(),
  rglactions = list(),
  style = "default",
  output_img = "fsbrain_arranged.png",
  silent = FALSE,
  grid_like = TRUE
)

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.

Value

  list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: 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()
Examples

```r
## 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

```r
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.
**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**

```r
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**

```r
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. (Obviously, 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

```r
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 `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 `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

**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**

```r
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 range `[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**

```r
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**

```r
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**

```r
vol.merge(
    volume,
    overlay_colors,
    bbox_threshold = 0L,
    forced_overlay_color = NULL
)
```
**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**

```r
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’.

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.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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>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.</td>
</tr>
<tr>
<td>slice_index</td>
<td>positive integer or vector of positive integers, the index into the slices (for the axis). A <em>slice</em> 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.</td>
</tr>
<tr>
<td>frame</td>
<td>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.</td>
</tr>
<tr>
<td>axis</td>
<td>positive integer, the axis to use when indexing the slices. Defaults to 1.</td>
</tr>
<tr>
<td>rotation</td>
<td>integer, rotation in degrees. Defaults to 0 (no rotation). Must be a multiple of 90L if given.</td>
</tr>
<tr>
<td>flip</td>
<td>NULL or one of the character strings ‘vertically’ or ‘horizontally’. Note that flipping <em>horizontally</em> means that the image will be mirrored along the central <em>vertical</em> axis. If ‘NULL’ is passed, nothing is flipped. Flipping occurs after rotation.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs_crs</td>
<td>integer vector of length 3, Freesurfer indices for column, row, and slice (CRS).</td>
</tr>
<tr>
<td>add_affine</td>
<td>logical, whether to add 1 to the output vector as the 4th value</td>
</tr>
</tbody>
</table>
Value

the R indices into the volume data for the given FreeSurfer CRS indices

Examples

```r
# 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**

```r
volvis.contour(volume, level = 80, show = TRUE)
```

**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’.

**Value**

the rendered triangles (a ‘Triangles3D’ instance) with coordinates in surface RAS space if any, ‘NULL’ otherwise.

**Examples**

```r
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);
```
**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 appear in 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**

```r
draw_lightbox <- function(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 integer `-n`, interpreted as every `n`th 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 `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.
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

```r
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. Obviously, you should not pass a color material parameter (see ‘...’) when using this.
- **...**: material properties, passed to `triangles3d`. Example: `color = "#0000ff", lit=FALSE.`
Examples

```r
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);
```

---

**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

```r
vox2ras_tkr()
```

Value

numeric 4x4 matrix, the FreeSurfer vox2ras_tkr matrix.

See Also

Other surface and volume coordinates: `ras2vox_tkr()`

Examples

```r
# 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.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**

```r
code
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**

```r
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'.

---


write.region.values.fsaverage

- **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**

```r
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);
```

**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**

```r
write.region.values.fsaverage(
  hemi, atlas, region_value_list, output_file, template_subject = "fsaverage",
```
write.region.values.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

alphablend, 4, 28
annot.outline, 5
apply.label.to.morphdata, 6, 7, 49, 50, 85–87, 97, 106
apply.labeldata.to.morphdata, 6, 7, 49, 50, 85–87, 97, 106
apply.transform, 7
arrange.brainview.images, 8, 113
brainviews, 9
c, 55
clip.data, 10, 20
collayer.bg, 10, 12–18, 93, 95, 102, 105, 107, 109, 112
collayer.bg.atlas, 11, 11, 13–18
collayer.bg.meancurv, 11, 12, 12, 14–18
collayer.bg.sulc, 11–13, 13, 15–18
collayer.from.annot, 11–14, 14, 15–18
collayer.from.annotdata, 11–15, 15, 16–18
collayer.from.mask.data, 11–15, 16, 17, 18
collayer.from.morphlike.data, 11–16, 16, 18
collayers.merge, 10–17, 17
coloredmesh.from.mask, 18, 63, 99
coloredmesh.from.morph.native, 19, 113
coloredmesh.plot.colorbar.separate, 9, 20, 24, 25, 92, 93, 95, 97, 99, 102, 105, 107, 109, 111
coloredmeshes.from.color, 22
colorlist.brain.clusters, 23
combine.colorbar.with.brainview.animation, 21, 23, 25
combine.colorbar.with.brainview.image, 21, 24, 24
cube3D.tris, 25, 27
cubes3D.tris, 26
delete_all_optional_data, 27

desaturate, 5, 27
download_fsaverage, 28, 30
download_optional_data, 29
face.edges, 29, 59, 65, 88, 100
find.freesurferhome, 30
find.subjectsdir.of, 30, 130
flip3D, 31, 72
fs.coloredmesh, 32
fup, 32
get.atlas.region.names, 33, 37, 39, 40, 45, 60, 61, 69, 74–77, 79, 83
group.agg.atlas.native, 33, 36, 39, 40, 45, 60, 61, 69, 74–77, 79, 83
group.agg.atlas.standard, 33, 37, 37, 40, 45, 60, 61, 69, 74–77, 79, 83
group.annot, 33, 37, 39, 39, 45, 60, 61, 69, 74–77, 79, 83
group.concat.measures.native, 40, 42
group.concat.measures.standard, 41, 41
group.label, 43, 62, 63, 82
group.label.from.annot, 33, 37, 39, 40, 44, 60, 61, 69, 74–77, 79, 83
group.morph agg.native, 45, 48, 52, 54
group.morph agg.standard, 46, 47, 52, 54
group.morph.native, 6, 7, 48, 50, 86, 87
group.morph.standard, 6, 7, 49, 49, 86, 87
group.multimorph.agg.native, 46, 48, 51, 54
group.multimorph.agg.standard, 46, 48, 52, 53
hasIn, 54
hemilist.derive.hemi, 55
hemilist.get.combined.data, 55
hemilist.unwrap, 56
hemilist.wrap, 56
image.plot, 21
image.border, 8, 124
image.composite, 24, 25
image_read, 116
is.fs.coloredmesh, 57
is.fs.coloredvoxels, 57
is.hemilist, 58
jet, 19, 20
label.border, 30, 58, 65, 88, 100
label.from.annotdata, 33, 37, 39, 40, 45, 59, 61, 69, 74–77, 79, 83
label.to.annot, 33, 37, 39, 40, 45, 60, 69, 74–77, 79, 83
labeldata.from.mask, 43, 61, 63, 82
list_optional_data, 62
makecmap, 16, 17, 19, 20, 93, 95, 97, 99, 102, 105, 107, 109, 111
mask.from.labeldata.for.hemi, 19, 43, 62, 63, 82, 99
material3d, 113
max, 46, 47, 52, 53
mean, 46, 48, 52, 54
mesh.vertex.included.faces, 30, 59, 65, 88, 100
mesh.vertex.neighbors, 30, 59, 64, 88, 100
par3d, 9, 90, 92, 93, 95, 97, 99, 102, 103, 105, 107, 109, 111, 113
png, 21
print.fs.coloredmesh, 65
print.fs.coloredvoxels, 66
ras2vox_tkr, 66, 126
read.colorcsv, 67
read.fs.annot, 61, 77
read.fs.colortable, 120
read.fs.mgh, 89
read.fs.morph, 86
read.fs.surface, 5, 29, 58, 64, 88, 100
read.md.demographics, 67, 69
read.md.subjects, 68, 68
regions.to.ignore, 33, 37, 39, 40, 45, 60, 61, 69, 74–77, 79, 83
report.on.demographics, 70
rgb, 18
rgl.spheres, 71
rglvoxels, 71
rotate3d, 31, 72
spread.values.over.annot, 33, 37, 39, 40, 45, 60, 61, 69, 73, 75–77, 79, 83
spread.values.over.hemi, 33, 37, 39, 40, 45, 60, 61, 69, 74, 76, 77, 79, 83
spread.values.over.subject, 33, 37, 39, 40, 45, 60, 61, 69, 74, 75, 77, 79, 83
subject.annot, 5, 15, 33, 37, 39, 40, 45, 60, 61, 69, 74–76, 77, 79, 83
subject.atlas.agg, 33, 37, 39, 40, 45, 60, 61, 69, 74–77, 78, 83
subject.filepath.morph.native, 79
subject.filepath.morph.standard, 80
subject.label, 7, 43, 62, 63, 81
subject.label.from.annot, 33, 37, 39, 40, 45, 60, 61, 69, 74–77, 79, 82
subject.mask, 6, 7, 84, 97, 106
subject.morph.native, 6, 7, 49, 50, 85, 87
subject.morph.standard, 6, 7, 49, 50, 86, 86
subject.surface, 5, 19, 29, 30, 32, 58, 59, 64, 65, 88, 100
subject.volume, 89
tmesh3d, 19, 20, 22, 32
toupper, 32
triangles3d, 25, 71, 125
vis.color.on.subject, 11–17, 90, 94, 95, 97, 99, 102, 104, 105, 107, 110, 112, 113
vis.coloredmeshes, 91
vis.data.on.fsaverage, 91, 92, 95, 97, 99, 102, 104, 105, 107, 110, 112, 113
vis.data.on.subject, 91, 94, 94, 97, 99, 102, 104, 105, 107, 110, 112, 113
vis.data.on.subject, 91, 94, 94, 97, 99, 102, 104, 105, 107, 110, 112, 113
vis.labeldata.on.subject, 6, 7, 85, 91, 94, 95, 96, 99, 102, 104–107, 110, 112, 113
vis.mask.on.subject, 19, 63, 91, 94, 95, 97, 98, 102, 104, 105, 107, 110, 112, 113
vis.path.along.verts, 30, 59, 65, 88, 100
vis.region.values.on.subject, 91, 94, 95, 97, 99, 101, 104, 105, 107, 110, 112, 113
vis.subject.annot, 91, 94, 95, 97, 99, 102, 103, 105, 108, 110, 112, 113
vis.subject.label, 6, 7, 85, 91, 94, 95, 97, 99, 102, 104, 104, 108, 110, 112, 113
vis.subject.morph.native, 21, 34, 91, 94, 95, 97, 99, 100, 102, 104, 105, 106, 110, 112, 113, 125
vis.symmetric.data.on.subject, 91, 94, 95, 97, 99, 102, 104, 105, 108, 110, 110, 113
vol.boundary.box, 114, 116, 119–122
vol.boundary.box.apply, 115
vol.boundary.mask, 114
vol.hull, 115
vol.imagestack, 115, 116, 119–122
vol.intensity.to.color, 117
vol.mask.from.segmentation, 118
vol.merge, 115, 116, 118, 120–122
vol.overlay.colors.from.activation, 115, 116, 119, 119, 121, 122
vol.overlay.colors.from.colortable, 120
vol.planes, 115, 116, 119, 120, 121, 122
vol.slice, 115, 116, 119–121, 121
vol.vox.from.crs, 122
volvis.contour, 123
volvis.lightbox, 121, 124
volvis.voxels, 125
vox2ras_tkr, 66, 125, 126
which, 61
write.region.aggregated, 127, 129, 130
write.region.values, 128, 128, 130
write.region.values.fsaverage, 128, 129, 129