

Package ‘fsbrain’

September 26, 2026

Type Package

Title Managing and Visualizing Brain Surface Data

Version 1.0.0

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

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

License MIT + file LICENSE

Copyright Tim Schäfer (package code); the optional data files that can be downloaded at runtime are owned by their respective copyright holders, see 'inst/COPYRIGHTS'.

Encoding UTF-8

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

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

Imports freesurferformats (>= 1.1.0), pkgfilecache (>= 0.4.1), rgl, squash, fields, viridis, magick, methods

Suggests knitr, rmarkdown, scimesh (>= 0.4.0), testthat (>= 2.1.0), misc3d, oro.nifti, RColorBrewer, Rvcg (>= 0.20.2), igraph, pracma, withr

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

NeedsCompilation no

Author Tim Schäfer [aut, cre] (ORCID: <<https://orcid.org/0000-0002-3683-8070>>), The General Hospital Corporation (Boston, MA) [cph] (Copyright holder of FreeSurfer and thus the fsaverage template subject that can OPTIONALLY be downloaded via fsbrain functions.), Van Essen Lab [cph] (Copyright holder of the fs_LR 32k template that

can OPTIONALLY be downloaded via fsbrain functions.),
 Alexander Schaefer [cph] (Copyright holder of the Schaefer 100-1000
 atlases that can OPTIONALLY be downloaded via fsbrain functions.),
 Ru Kong [cph] (Copyright holder of the Brainnetome atlas that can
 OPTIONALLY be downloaded via fsbrain functions.),
 Lingzhong Fan [cph] (Copyright holder of the AAL3 atlas that can
 OPTIONALLY be downloaded via fsbrain functions.),
 Edmund T. Rolls [cph] (Copyright holder of the HCP-MMP1 atlas that can
 OPTIONALLY be downloaded via fsbrain functions.),
 Matthew F. Glasser [cph] (Copyright holder of the HCP-MMP1 atlas that
 can OPTIONALLY be downloaded via fsbrain functions.),
 Kathryn Mills [cph] (Provided the conversion of the HCP-MMP1 atlas to
 fsaverage space that can OPTIONALLY be downloaded via fsbrain
 functions.)

Repository CRAN

Date/Publication 2026-09-26 20:40:02 UTC

Contents

alphablend	7
annot.outline	8
annot.outline.border.vertices	9
apply.label.to.morphdata	10
apply.labeldata.to.morphdata	11
apply.transform	12
arrange.brainview.images	13
arrange.brainview.images.grid	14
brainviews	15
clip.data	16
clip_fun	17
cm.cbry	18
cm.div	18
cm.heat	19
cm.qual	19
cm.seq	20
collayer.bg	20
collayer.bg.atlas	21
collayer.bg.meancurv	22
collayer.bg.sulc	23
collayer.from.annot	24
collayer.from.annotdata	24
collayer.from.mask.data	25
collayer.from.morphlike.data	26
collayers.merge	27
coloredmesh.from.annot	27
coloredmesh.from.color	28
coloredmesh.from.label	30

coloredmesh.from.mask	31
coloredmesh.from.morph.native	32
coloredmesh.from.morph.standard	33
coloredmesh.from.morphdata	35
coloredmesh.from.preloaded.data	36
coloredmesh.plot.colorbar.separate	36
coloredmeshes.from.color	38
colorlist.brain.clusters	40
colors.are.grayscale	40
colors.have.transparency	41
combine.colorbar.with.brainview.animation	41
combine.colorbar.with.brainview.image	42
constant.pervertexdata	44
cube3D.tris	44
cubes3D.tris	46
delete_all_optional_data	46
demographics.to.fsgd.file	47
demographics.to.qdec.table.dat	48
desaturate	49
download_fsaverage	50
download_fsaverage3	51
download_fsaverage6	52
download_fsaverage_atlases	52
download_fsaverage_minimal	53
download_fs_LR_32_atlases	54
download_fs_LR_32_labels	55
download_fs_LR_32_meshes	56
download_optional_data	56
download_optional_paper_data	57
download_xtract_tracts	58
export	59
export.coloredmesh.ply	61
face.edges	62
find.freesurferhome	63
find.subjectsdir.of	63
fs.coloredmesh	64
fs.coloredpaths	65
fs.home	66
fs.surface.as.adjacencylist	67
fs.surface.to.igraph	67
fs.surface.to.tmesh3d	68
fs.surface.vertex.neighbors	68
fsaverage.path	69
fsbrain.set.default.figsize	70
fup	71
gen.test.volume	71
geod.patches.color.overlay	72
geod.vert.neighborhood	73

geodesic.circles	74
geodesic.dists.to.vertex	75
geodesic.path	75
get.atlas.region.names	76
get.fsbrain.renderer.backend	77
get.rglstyle	78
get.view.angle.names	79
getIn	80
get_optional_data_filepath	80
group.agg.atlas.native	81
group.agg.atlas.standard	82
group.annot	84
group.concat.measures.native	85
group.concat.measures.standard	86
group.label	88
group.label.from.annot	89
group.morph.agg.native	90
group.morph.agg.standard	92
group.morph.agg.standard.vertex	93
group.morph.native	95
group.morph.standard	96
group.morph.standard.sf	97
group.multimorph.agg.native	98
group.multimorph.agg.standard	100
group.surface	102
groupmorph.split.hemilist	103
hasIn	103
hemilist	104
hemilist.derive.hemi	105
hemilist.from.prefixedList	105
hemilist.get.combined.data	106
hemilist.unwrap	107
hemilist.wrap	107
highlight.points.spheres	108
highlight.vertices.on.subject	109
highlight.vertices.on.subject.spheres	110
highlight.vertices.spheres	112
images.dimmax	113
index2ras_tkr	114
is.fs.coloredmesh	115
is.fs.coloredpaths	115
is.fs.coloredvoxels	116
is.fsbrain	116
is.hemilist	117
label.border	117
label.colFn	118
label.colFn.inv	119
label.from.annotdata	119

label.to.annot	120
labeldata.from.mask	121
limit_fun	122
limit_fun_na	123
limit_fun_na_inside	124
list_optional_data	125
mask.from.labeldata.for.hemi	125
mesh.vertex.neighbors	126
mkco.cluster	128
mkco.div	128
mkco.heat	129
mkco.seq	129
numverts.lh	130
numverts.rh	130
principal.curvatures	131
print.fs.coloredmesh	131
print.fs.coloredpaths	132
print.fs.coloredvoxels	132
print.fsbrain	133
qc.for.group	133
qc.from.regionwise.df	134
qc.from.segstats.tables	135
qc.vis.failcount.by.region	135
qdec.table.skeleton	136
ras2vox_tkr	137
read.colorcsv	138
read.md.demographics	138
read.md.subjects	139
read.md.subjects.from.fsgd	140
read.tract.bundles	141
regions.to.ignore	143
report.on.demographics	143
rglactions	144
rglo	145
rglot	145
rglvoxels	146
scale01	147
shape.descriptor.names	147
shape.descriptors	148
shift.hemis.apart	148
sjd.demo	149
sjld	150
spread.values.over.annot	150
spread.values.over.hemi	152
spread.values.over.subject	153
subject.annot	154
subject.annot.border	156
subject.atlas.agg	157

subject.filepath.morph.native	158
subject.filepath.morph.standard	159
subject.label	160
subject.label.from.annot	161
subject.lobes	162
subject.mask	163
subject.morph.native	165
subject.morph.standard	166
subject.num.verts	168
subject.region.centroids	168
subject.surface	170
subject.vol2surf	171
subject.volume	174
surface.curvatures	175
template.vol2surf	176
tmesh3d.to.fs.surface	178
Triangles3D.to.coloredmesh	178
vdata.split.by.hemi	179
vertex.coords	180
vertex.hemis	180
vis.color.on.subject	181
vis.coloredmeshes	183
vis.coloredmeshes.rotating	184
vis.colortable.legend	185
vis.connectome	186
vis.data.on.fsaverage	190
vis.data.on.group.native	192
vis.data.on.group.standard	193
vis.data.on.subject	194
vis.dti.trk	197
vis.export.from.coloredmeshes	198
vis.fs.surface	200
vis.group.annot	201
vis.group.coloredmeshes	202
vis.group.morph.native	203
vis.group.morph.standard	204
vis.labeldata.on.subject	206
vis.mask.on.subject	208
vis.path.along.verts	210
vis.paths	212
vis.paths.along.verts	212
vis.region.values.on.subject	213
vis.rglwidget	215
vis.seg.legend	217
vis.subcortical.region.values	217
vis.subject.annot	221
vis.subject.label	222
vis.subject.morph.native	224

vis.subject.morph.standard	226
vis.subject.pre	229
vis.symmetric.data.on.subject	230
vis.tracts	232
vis.volume.clusters	236
vis.volume.on.surface	240
vislayout.from.coloredmeshes	243
vol.boundary.box	245
vol.boundary.box.apply	245
vol.hull	246
vol.imagestack	247
vol.intensity.to.color	247
vol.mask.from.segmentation	248
vol.merge	249
vol.overlay.colors.from.activation	250
vol.overlay.colors.from.colortable	251
vol.planes	251
vol.slice	252
vol.vox.from.crs	253
volvis.contour	254
volvis.lb	255
volvis.lb.with.surface	256
volvis.lightbox	259
volvis.shells	261
volvis.slices.with.surface	264
volvis.voxels	266
vox2ras_tkr	267
write.group.morph.standard	268
write.group.morph.standard.mf	269
write.group.morph.standard.sf	270
write.region.aggregated	271
write.region.values	272
write.region.values.fsaverage	273

Index**275**

alphablend

*Perform alpha blending for pairs of RGBA colors.***Description**

Implements the *over* alpha blending operation.

Usage

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

Arguments

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

Value

rgba color strings, the alpha-blended colors

References

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

See Also

Other color functions: [desaturate\(\)](#)

annot.outline	<i>Compute outline vertex colors from annotation.</i>
---------------	---

Description

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

Usage

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

Arguments

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

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

Value

vector of colors, one color for each mesh vertex

Note

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

annot.outline.border.vertices

Compute the border vertices for each region in an annot.

Description

Compute the border vertices for each region in an annot.

Usage

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

Arguments

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

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

Value

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

`apply.label.to.morphdata`

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

Description

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

Usage

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

Arguments

<code>morphdata</code>	numerical vector, the morphometry data for one hemisphere
<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier
<code>hemi</code>	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.

label string, fs.label instance, or label vertex data. If a string, interpreted as the file name of the label file, without the hemi part (if any), optionally including the '.label' suffix. E.g., 'cortex.label' or 'cortex' for '?h.cortex.label'.

masked_data_value numerical, the value to set for all morphometry data values of vertices which are *not* part of the label. Defaults to NA.

Value

numerical vector, the masked data.

See Also

Other label functions: [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

Other morphometry data functions: [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

`apply.labeldata.to.morphdata`

Apply a label to morphometry data.

Description

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

Usage

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

Arguments

morphdata numerical vector, the morphometry data for one hemisphere

labeldata integer vector or fs.label instance. A label as returned by [subject.label](#).

masked_data_value numerical, the value to set for all morphometry data values of vertices which are *not* part of the label. Defaults to NA.

Value

numerical vector, the masked data.

See Also

Other label functions: [apply.label.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject](#), [vis.subject.label\(\)](#)

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

apply.transform	<i>Apply affine transformation to input.</i>
-----------------	--

Description

Apply an affine transformation, like a *vox2ras_tkr* transformation, to input. This is just matrix multiplication for different input objects. Supported input types are coordinate vectors, coordinate matrices, `fs.surface` meshes (the vertex coordinates are transformed, the face indices stay the same), renderable objects like `fs.coloredmesh`, `fs.coloredvoxels`, `fs.coloredpaths` or *misc3d* `Triangles3D`, `rgl mesh3d/tmesh3d` instances (including their normals, if any), and (hemi-)lists of such objects (which are transformed element-wise).

Usage

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

Arguments

object	numerical vector/matrix, <code>fs.surface</code> , <code>fs.coloredmesh</code> , <code>fs.coloredvoxels</code> , <code>fs.coloredpaths</code> , <code>Triangles3D</code> , <code>mesh3d/tmesh3d</code> instance, or a list (e.g., a hemilist) of such objects, the coordinates or objects to transform.
matrix_fun	a 4x4 affine matrix or a function returning such a matrix. If <code>NULL</code> , the input is returned as-is. In many cases you may want to use a matrix computed from the header of a volume file, e.g., the <i>vox2ras</i> matrix of the respective volume. See the <code>mgheader.*</code> functions in the <i>freesurferformats</i> package to obtain these matrices. Registration files can be read with <code>freesurferformats::read.fs.transform</code> and friends, but note that such files often describe a <i>voxel</i> to <i>surface RAS</i> mapping, so you may have to compose them with a <i>vox2ras</i> matrix to get a transformation between RAS coordinates.

Value

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

Note

The affine matrix is applied in the standard way: the coordinates are interpreted as homogeneous *column* vectors, i.e., a vertex v is transformed as $v' = M \%* \% v$. Note that `rgl`, and `fsbrain` functions that are implemented on top of `rgl` (like the camera transforms used internally for views), use the transposed convention for their rotation matrices, see [rotationMatrix](#). For pure translations and scalings, both conventions are identical.

Meshes keep their orientation: if the linear part of the matrix has a negative determinant, the transformation mirrors the object (this is the case for the FreeSurfer vox2ras_tkr matrix, which flips and permutes axes), which would invert all surface normals and make the mesh render inside-out. In that case, the vertex order within each face is reversed to preserve the original orientation, and the stored normals (if any) are transformed with the linear part of the matrix so that they stay consistent with the faces.

Examples

```
## Not run:
# Transform the vertex coordinates of a surface mesh:
cube_file = system.file("extdata", "cube.ply", package = "fsbrain");
cube = freesurferformats::read.fs.surface(cube_file);
translation = matrix(c(1,0,0,10, 0,1,0,20, 0,0,1,30, 0,0,0,1), nrow = 4L, byrow = TRUE);
cube_moved = apply.transform(cube, translation);

## End(Not run)
```

arrange.brainview.images

Combine several brainview images into a new figure.

Description

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

Usage

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

Arguments

brainview_images	vector of character strings, paths to the brainview images, usually in PNG format
output_img	path to output image that including the file extension
colorbar_img	path to the main image containing the separate colorbar, usually an image in PNG format

<code>silent</code>	logical, whether to suppress messages
<code>grid_like</code>	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally.
<code>border_geometry</code>	string, a geometry string passed to <code>magick::image_border</code> to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
<code>background_color</code>	hex color string, such as <code>"#DDDDDD"</code> or <code>"#FFFFFF"</code> . The color to use when extending images (e.g., when creating the border). WARNING: Do not use color names (like <code>'gray'</code>), as their interpretation differs between <code>rgl</code> and image <code>magick</code> !
<code>map_bg_to_transparency</code>	logical, whether to map the <code>background_color</code> to transparency for the final PNG export.

Value

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

```
arrange.brainview.images.grid
```

Combine several brainview images as a grid into a new figure.

Description

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

Usage

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

Arguments

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

Value

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

Note

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

brainviews

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

Description

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

Usage

```
brainviews(
  views,
  coloredmeshes,
  rgloptions = rglo(),
  rglactions = list(),
  style = "default",
```

```

    draw_colorbar = FALSE,
    background = "white"
)

```

Arguments

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

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

[get.view.angle.names](#)

clip.data

Clip data at quantiles to remove outliers.

Description

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

Usage

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

Arguments

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

Value

numeric vector. The output data.

See Also

The [clip_fun](#) function is more convenient when used in [rglactions](#), as it allows specification of custom quantiles.

Examples

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

clip_fun

Get data clipping function.

Description

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

Usage

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

Arguments

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

Value

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

See Also

[rglactions](#)

Examples

```

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

```

cm.cbry

Get cyan blue red yellow colormap function.

Description

Get cyan blue red yellow colormap function.

Usage

```
cm.cbry()
```

Note

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

cm.div

Return the standard fsbrain diverging colormap.

Description

Return the standard fsbrain diverging colormap.

Usage

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

Arguments

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

Note

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

cm.heat	<i>Return the standard fsbrain heat colormap.</i>
---------	---

Description

Return the standard fsbrain heat colormap.

Usage

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

Arguments

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

Note

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

cm.qual	<i>Return the standard fsbrain qualitative colormap.</i>
---------	--

Description

Return the standard fsbrain qualitative colormap.

Usage

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

Arguments

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

Note

Returns some qualitative palette, suitable for visualizing categorical data.

cm.seq	<i>Return the standard fsbrain sequential colormap.</i>
--------	---

Description

Return the standard fsbrain sequential colormap.

Usage

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

Arguments

report	logical, whether to print a message with a name of the chosen colormap, in format <code>package::function#palette</code> .
--------	--

Note

This returns a sequential, multi-hue palette.

collayer.bg	<i>Compute binarized mean curvature surface color layer.</i>
-------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

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

<code>collayer.bg.atlas</code>	<i>Compute atlas or annotation surface color layer.</i>
--------------------------------	---

Description

Compute atlas or annotation surface color layer.

Usage

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

Arguments

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

Value

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

Note

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

See Also

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

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

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

Description

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

Usage

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

Arguments

<code>subjects_dir</code>	character string, the FreeSurfer SUBJECTS_DIR.
<code>subject_id</code>	character string, the subject identifier.
<code>hemi</code>	character string, one of 'lh', 'rh', or 'both'. The latter will merge the data for both hemis into a single vector.
<code>cortex_only</code>	logical, whether to restrict pattern computation to the cortex.
<code>bin_colors</code>	vector of two character strings, the two colors to use.
<code>bin_thresholds</code>	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\(\)](#), [collayer.bg.atlas\(\)](#), [collayer.bg.sulc\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.data\(\)](#), [collayers.merge\(\)](#)

collayer.bg.sulc	<i>Compute binarized sulcal depth surface color layer.</i>
------------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

You can plot the return value using [vis.color.on.subject](#).

Other surface color layer: [collayer.bg\(\)](#), [collayer.bg.atlas\(\)](#), [collayer.bg.meancurv\(\)](#), [collayer.from.annot\(\)](#), [collayer.from.annotdata\(\)](#), [collayer.from.mask.data\(\)](#), [collayer.from.morphlike.d](#), [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

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

Value

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

See Also

You can plot the return value using [vis.color.on.subject](#).

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

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

Description

Compute surface color layer from annotation or atlas data.

Usage

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

Arguments

lh_annotdata loaded annotation data for left hemi, as returned by [subject.annot](#)
rh_annotdata loaded annotation data for right hemi

Value

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

See Also

You can plot the return value using [vis.color.on.subject](#).

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

`collayer.from.mask.data`

Compute surface color layer from morph-like data.

Description

Compute surface color layer from morph-like data.

Usage

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

Arguments

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

Value

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

See Also

You can plot the return value using [vis.color.on.subject](#).

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

`collayer.from.morphlike.data`

Compute surface color layer from morph-like data.

Description

Compute surface color layer from morph-like data.

Usage

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

Arguments

`lh_morph_data` numerical vector, can be NULL

`rh_morph_data` numerical vector, can be NULL

`makecmap_options`
 named list of parameters to pass to [makecmap](#). Must not include the unnamed first parameter, which is derived from 'measure'.

`return_metadata`
 logical, whether to return additional metadata as entry 'metadata' in the returned list

Value

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

See Also

You can plot the return value using [vis.color.on.subject](#).

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

collayers.merge	<i>Merge two or more color layers based on their transparency values.</i>
-----------------	---

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

coloredmesh.from.annot	<i>Create a coloredmesh from an annotation of an atlas.</i>
------------------------	---

Description

Create a coloredmesh from an annotation of an atlas.

Usage

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

Arguments

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

Value

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

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

coloredmesh.from.color

Create a coloredmesh from a mesh and pre-defined colors.

Description

Create a coloredmesh from a mesh and pre-defined colors.

Usage

```
coloredmesh.from.color(
    subjects_dir,
    subject_id,
    color_data,
    hemi,
    surface = "white",
    metadata = list(),
    style = 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	vector of hex color strings, a single one or one per vertex.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or fs.surface instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
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.
style	NULL or a rendering style for this mesh, see get.rglstyle . Styles can be a style name (like 'default' or 'glass') or a named list of material properties (like list('alpha'=0.2)). The style is stored in the 'style' field of the returned coloredmesh and is used when the mesh is rendered with style='from_mesh', see vis.coloredmeshes . This is how you can give individual meshes in a scene their own look, e.g., to draw a cortex mesh semi-transparently behind colored data meshes.

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'. If not NULL, also "style": the rendering style for this mesh.

Note

You will usually not call this directly, but use [coloredmeshes.from.color](#) or [vis.color.on.subject](#) instead. It is exported for cases in which you want to build a single hemisphere of a scene manually.

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

coloredmesh.from.label

Create a coloredmesh from a label.

Description

Create a coloredmesh from a label.

Usage

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

Arguments

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

Value

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

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

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

Description

Create a coloredmesh from a mask.

Usage

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

Arguments

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

Value

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

See Also

Other mask functions: [mask.from.labeldata.for.hemi\(\)](#), [vis.mask.on.subject\(\)](#)

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

coloredmesh.from.morph.native

Create a coloredmesh from native space morphometry data.

Description

Create a coloredmesh from native space morphometry data.

Usage

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

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
measure	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render the surface in white, without any data. One can also pass the pre-loaded morphometry data as a numerical vector, the length of which must match the number of surface vertices.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	character string or fs.surface instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".

clip	numeric vector of length 2 or NULL. If given, the 2 values are interpreted as lower and upper percentiles, and the morph data is clipped at the given lower and upper percentile (see clip.data). Defaults to NULL (no data clipping).
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'.

Value

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

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.st](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

coloredmesh.from.morph.standard

Create a coloredmesh from standard space morphometry data.

Description

Create a coloredmesh from standard space morphometry data.

Usage

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

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier.
<code>measure</code>	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.
<code>hemi</code>	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
<code>fwhm</code>	string, smoothing setting. The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.
<code>surface</code>	character string or <code>fs.surface</code> instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
<code>template_subject</code>	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to 'fsaverage'.
<code>template_subjects_dir</code>	The template subjects dir. If NULL, the value of the parameter 'subjects_dir' is used. Defaults to NULL. If you have FreeSurfer installed and configured, and are using the standard fsaverage subject, try passing the result of calling <code>file.path(Sys.getenv('FREESURFER_HOME'), 'subjects')</code> .
<code>clip</code>	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).
<code>cortex_only</code>	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) 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.
<code>makecmap_options</code>	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'.

Value

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

See Also

Other `coloredmesh` functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.nat](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

coloredmesh.from.morphdata

Create a coloredmesh from arbitrary data.

Description

Create a coloredmesh from arbitrary data.

Usage

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

Arguments

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

Value

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

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.nat](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmeshes.from.color\(\)](#)

```
coloredmesh.from.preloaded.data
```

Generate coloredmesh from loaded data.

Description

Generate coloredmesh from loaded data.

Usage

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

Arguments

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

Value

as `fs.coloredmesh` instance

```
coloredmesh.plot.colorbar.separate
```

Draw colorbar for coloredmeshes in separate 2D plot.

Description

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

Usage

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

Arguments

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

ticks, note that they must be on a $\log(\text{data})$ scale. If the 'makecmap_options' stored in the passed 'coloredmeshes' contain a 'base' value of 10, log 10 is assumed (with the default 5 ticks), even if this parameter is left at its default value, logical FALSE.

Value

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

Note

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

See Also

Other colorbar functions: [combine.colorbar.with.brainview.animation\(\)](#), [combine.colorbar.with.brainview.image.plot.extra_options\(\)](#)

Examples

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

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

## End(Not run)
```

coloredmeshes.from.color

Create coloredmeshes for both hemis using pre-defined colors.

Description

Create coloredmeshes for both hemis using pre-defined colors.

Usage

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

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier.
<code>color_data</code>	a hemilist containing vectors of hex color strings
<code>hemi</code>	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
<code>surface</code>	character string or <code>fs.surface</code> instance. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
<code>metadata</code>	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.
<code>style</code>	NULL or a rendering style for the created meshes, see get.rglstyle . Styles can be a style name (like 'default' or 'glass') or a named list of material properties (like <code>list('alpha'=0.2)</code>). The style is stored in the 'style' field of the returned coloredmeshes and is used when the meshes are rendered with <code>style='from_mesh'</code> , see vis.coloredmeshes .

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'. If not NULL, also "style": the rendering style for the mesh.

See Also

Other coloredmesh functions: [Triangles3D.to.coloredmesh\(\)](#), [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.nat](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#)

```
colorlist.brain.clusters
```

Return diverging color list

Description

Return diverging color list

Usage

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

Arguments

num_colors integer, the number of colors you want

Value

vector of colors

```
colors.are.grayscale
```

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

Description

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

Usage

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

Arguments

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

Value

logical vector

Examples

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

colors.have.transparency

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

Description

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

Usage

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

Arguments

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

Value

logical vector

Examples

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

combine.colorbar.with.brainview.animation

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

Description

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

Usage

```

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

```

Arguments

brain_animation	path to the brain animation in GIF format
colorbar_img	path to the main image containing the separate colorbar, usually an image in PNG format
output_animation	path to output image in gif format, must include the '.gif' file extension
offset	offset string passed to <code>magick::image_composite</code> . Allows you to shift the location of the colorbar in the final image.
extend_brainview_img_height_by	integer value in pixels, the size of the lower border to add to the brainview_img. Use this if the lower part of the colorbar is off the image canvas.
silent	logical, whether to silence all messages
allow_colorbar_shrink	logical, whether to shrink the colorbar to the width of the animation in case it is considerably wider (more than 20 percent). Defaults to TRUE.
background_color	color string, the background color to use. Use 'transparency_color' if you want a transparent background.

See Also

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

```
combine.colorbar.with.brainview.image
```

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

Description

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

Usage

```

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

```

Arguments

<code>brainview_img</code>	path to the main image containing the view of the brain, usually an image in PNG format.
<code>colorbar_img</code>	path to the main image containing the separate colorbar, usually an image in PNG format.
<code>output_img</code>	path to output image, including the file extension.
<code>offset</code>	offset string passed to <code>magick::image_composite</code> . Allows you to shift the location of the colorbar in the final image.
<code>extend_brainview_img_height_by</code>	integer value in pixels, the size of the lower border to add to the <code>brainview_img</code> . Increase this if the lower part of the colorbar is off the image canvas.
<code>silent</code>	logical, whether to silence all messages
<code>allow_colorbar_shrink</code>	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.
<code>horizontal</code>	logical, whether the colorbar is horizontal. If so, it will be added below the <code>'brainview_img'</code> . If it is vertical, it will be added to the right of the <code>'brainview_img'</code> .
<code>background_color</code>	color string, the background color to use. Use <code>'transparency_color'</code> if you want a transparent background.
<code>transparency_color</code>	the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try <code>'white'</code> or <code>'black'</code> if in doubt.
<code>delete_colorbar_img</code>	logical, whether to delete the <code>colorbar_img</code> after the combine operation.

Value

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

See Also

Other colorbar functions: [coloredmesh.plot.colorbar.separate\(\)](#), [combine.colorbar.with.brainview.animation\(\)](#)

constant.pervertexdata

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

Description

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

Usage

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

Arguments

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

Value

a vector or hemilist of vectors of values

cube3D.tris

Return triangles for a 3D cube or cuboid.

Description

Each row of the returned matrix encodes a point (the x, y, and z coordinates), and 3 consecutive rows encode a triangle. 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

```
# 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	<i>Vectorized version of cube3D.tris</i>
--------------	--

Description

Vectorized version of cube3D.tris

Usage

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

Arguments

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

Value

matrix of triangle coordinates, see [cube3D.tris](#).

Examples

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

delete_all_optional_data

Delete all data in the package cache.

Description

Delete all data in the package cache.

Usage

```
delete_all_optional_data()
```

Value

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

demographics.to.fsgd.file

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

Description

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

Usage

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

Arguments

filepath	character string, the path to the output file in FSGD format
demographics_df	data.frame, as returned by read.md.demographics or created manually. Note that the data.frame must not contain any character columns, they should be converted to factors.
group_column_name	character string, the column name of the group column in the 'demographics_df'
subject_id_column_name	character string, the column name of the subject identifier column in the 'demographics_df'
var_columns	vector of character strings, the column names to include as variables in the FSGD file. If NULL (the default), all columns will be included (with the exception of the group column and the subject id column).
ftitle	character string, freeform title for the FSGD file
fsgd_flag_lines	vector of character strings, extra flag lines to write to the file. The default setting will activate de-meaning and rescaling.

Value

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

See Also

Other metadata functions: [read.md.demographics\(\)](#), [read.md.subjects\(\)](#), [report.on.demographics\(\)](#)

demographics.to.qdec.table.dat

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

Description

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

Usage

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

Arguments

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

`long_timecolumn` character string, the name of the column holding the inter-scan time. Ignored unless parameter `long` is `TRUE`. See the description for parameter `long` for details.

`qdec_file_name` character string, the filename of the QDEC file to write. Must be only the file name (with extension if you want). See `output_path` to set the output directory where this will be created.

Note

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

See Also

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

Examples

```
## Not run:
dem = readxl::read_xls("~/data/study1/demographics.xls");
# or: dem = read.table("~/demographics.csv", check.names=FALSE);
# You may want to rearrange/rename/delete some columns here.
demographics.to.qdec.table.dat(dem, "~/data/study1/qdec/");
#
# a second one with real data:
dem = data.frame("ID"=paste("subject", seq(5), sep=""),
  "age"=sample.int(20, 5)+10L, "isi"=rnorm(5, 2.0, 0.1)); #sample data.
long_table = qdec.table.skeleton(dem$ID, dem$isi);
demographics.to.qdec.table.dat(long_table, long=TRUE);

## End(Not run)
```

desaturate

Perform simple desaturation or grayscale conversion of RGBA colors.

Description

Perform simple desaturation or grayscale conversion of RGBA colors.

Usage

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

Arguments

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

Value

rgba color strings, the gray-scale colors. The information from one of the three RGB channels would be enough. The alpha value is not touched.

Note

Assumes sRGB color space.

References

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

See Also

Other color functions: [alphablend\(\)](#)

download_fsaverage	<i>Download the FreeSurfer v6 fsaverage subject.</i>
--------------------	--

Description

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

Usage

```
download_fsaverage(accept_freesurfer_license = FALSE, scheme = "https")
```

Arguments

accept_freesurfer_license	logical, whether you accept the FreeSurfer license for fsaverage, available at https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense . Defaults to FALSE.
scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.

Value

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

download_fsaverage3	<i>Download the FreeSurfer v6 low-resolution fsaverage3 subject.</i>
---------------------	--

Description

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

Usage

```
download_fsaverage3(accept_freesurfer_license = FALSE, scheme = "https")
```

Arguments

accept_freesurfer_license	logical, whether you accept the FreeSurfer license for fsaverage, available at https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense . Defaults to FALSE.
scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.

Value

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

Note

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

download_fsaverage6 *Download the FreeSurfer v6 fsaverage6 subject.*

Description

Download some relevant files from the FreeSurfer v6 fsaverage6 subject. The files are subject to the FreeSurfer software license, see parameter 'accept_freesurfer_license' for details. This data is not required for the package to work. If you are working on a machine that has FreeSurfer installed, you already have this data anyways and do not need to download it.

Usage

```
download_fsaverage6(accept_freesurfer_license = FALSE, scheme = "https")
```

Arguments

accept_freesurfer_license	logical, whether you accept the FreeSurfer license for fsaverage, available at https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense . Defaults to FALSE.
scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.

Value

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

Note

The subject fsaverage6 is a slightly downsampled (low mesh resolution) version of the standard fsaverage. If you never heard about fsaverage6, you do not need it. You will have to manually re-sample your data in FreeSurfer if you want to use it with fsaverage6.

download_fsaverage_atlases *Download atlas files for the fsaverage template subject.*

Description

Download a set of atlas files defined in the space of the fsaverage template subject, based on a declarative manifest file shipped with this package. The atlases (e.g., Schaefer 100-1000, Brainnetome, HCP-MMP1, AAL3) are defined in fsaverage space, but they are not part of FreeSurfer and are not subject to the FreeSurfer license. This data is not required for the package to work. Note that some atlases are not defined on the cortical surface and ship their own surface mesh, e.g., the subcortical atlas of the ENIGMA aseg structures: for those, the 'surf/lh.subcortical' and 'label/lh.subcortical.annot' files (and the same for the right hemisphere) are downloaded, see [vis.subcortical.region.value](#)

Usage

```
download_fsaverage_atlases(scheme = "https")
```

Arguments

scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.
--------	--

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_filepath()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

download_fsaverage_minimal

Download only essential FreeSurfer v6 fsaverage files for quick visualization.

Description

Download a minimal subset of files from the FreeSurfer v6 fsaverage subject, containing only the white surfaces, cortex labels, and curvature files needed for basic 3D visualization (e.g., in a Shiny app). This is much faster than [download_fsaverage](#) which downloads 19 files. The files are subject to the FreeSurfer software license, see parameter 'accept_freesurfer_license' for details.

Usage

```
download_fsaverage_minimal(accept_freesurfer_license = FALSE, scheme = "https")
```

Arguments

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

scheme character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.

Value

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

Note

This downloads approximately 8 files instead of the 19 downloaded by `download_fsaverage`, making it suitable for environments with startup time constraints (e.g., Posit Connect Cloud, shinyapps.io).

download_fs_LR_32_atlases

Download atlas files for the fs_LR 32k template.

Description

Download a set of cortical atlas files (annotations) defined in the space of the fs_LR 32k template (the HCP-style surface space), based on a declarative manifest file shipped with this package. The atlases (e.g., Schaefer 100-1000, Brainnetome, AAL3) are not part of FreeSurfer and are not subject to the FreeSurfer license. This data is not required for the package to work.

Usage

```
download_fs_LR_32_atlases(scheme = "https")
```

Arguments

scheme character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.

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_filepath()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

See Also

Other fs_LR 32k template functions: [download_fs_LR_32_labels\(\)](#), [download_fs_LR_32_meshes\(\)](#)

`download_fs_LR_32_labels`*Download label files for the fs_LR 32k template.*

Description

Download a set of label files for the fs_LR 32k template (the HCP-style surface space), based on a declarative manifest file shipped with this package. Currently this is the cortex label (`?h.cortex.label`), which defines the medial wall: all vertices which are *not* part of the label are medial wall vertices. It can be used to mask the medial wall when projecting volume data to the fs_LR 32k surface, see [template.vol2surf](#) (parameter `cortex_only`). This data is not required for the package to work.

Usage

```
download_fs_LR_32_labels(scheme = "https")
```

Arguments

<code>scheme</code>	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.
---------------------	--

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_filepath()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

See Also

Other fs_LR 32k template functions: [download_fs_LR_32_atlases\(\)](#), [download_fs_LR_32_meshes\(\)](#)

Examples

```
## Not run:
fsbrain::download_fs_LR_32_labels();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cortex_lh = subject.label(subjects_dir, "fs_LR_32", "cortex.label", "lh");

## End(Not run)
```

download_fs_LR_32_meshes

Download surface meshes for the fs_LR 32k template.

Description

Download a set of surface mesh files for the fs_LR 32k template (the HCP-style surface space), based on a declarative manifest file shipped with this package. The meshes include the white, pial, inflated, very inflated, midthickness, sphere and flat surfaces for both hemispheres. This data is not required for the package to work.

Usage

```
download_fs_LR_32_meshes(scheme = "https")
```

Arguments

scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.
--------	--

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_filepath()`. "missing": vector of strings. The names of the files that this function was unable to retrieve.

See Also

Other fs_LR 32k template functions: [download_fs_LR_32_atlases\(\)](#), [download_fs_LR_32_labels\(\)](#)

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. This function also calls [download_fsaverage_atlases](#), so the atlases defined in fsaverage space (including the subcortical atlas, see [vis.subcortical.region.values](#)) become available as well.

Usage

```
download_optional_data(scheme = "https")
```

Arguments

scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.
--------	--

Value

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

`download_optional_paper_data`

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

Description

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

Usage

```
download_optional_paper_data(scheme = "https")
```

Arguments

scheme	character string, the URL scheme to use. Either "https" (the default) or "http". Switching to "http" can be useful as a fallback if the HTTPS server is unreachable.
--------	--

Value

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

Note

Called for side effect of data download.

download_xtract_tracts

Download a white matter tract atlas (streamlines).

Description

Download one of the tract atlases that are distributed as streamlines in TrackVis TRK format, one file per white matter bundle. The bundle files (and the provenance and attribution files that document them) are downloaded into the fsbrain file cache, where [get_optional_data_filepath](#) can be used to access them. The bundles can then be read with [read.tract.bundles](#) and plotted with [vis.tracts](#).

Usage

```
download_xtract_tracts(
    atlas = "xtract_medium",
    download = TRUE,
    scheme = "https",
    silent = FALSE
)
```

Arguments

atlas	character string, the atlas to download. One of 'xtract_tiny' (the smallest one, useful for quick tests), 'xtract_small', 'xtract_medium' (the default) or 'xtract_large' (the most detailed one).
download	logical, whether to download the files if they are not in the cache. If FALSE, the function only reports the status of the files.
scheme	character string, the URL scheme to use, see the 'scheme' parameter of download_fs_LR_32_meshes .
silent	logical, whether to suppress the progress messages.

Details

The atlases are the XTRACT atlas of the 42 major white matter tracts (Warrington et al., 2020, [doi:10.1126/sciadv.aba8245](https://doi.org/10.1126/sciadv.aba8245)), whose streamlines were converted from the probabilistic tract atlases and are available in four levels of spatial detail. Note that the bundles are defined in MNI152 space, while the fs_LR_32 and fsaverage templates of fsbrain use a different (fsaverage-like) space: the two spaces are very similar, so the tracts align with the template surfaces well enough for visualization, but they are not identical. If you need an exact alignment, register the template to MNI152 and pass the resulting matrix to the parameter `transform_matrix` of [vis.tracts](#).

The files are hosted on the rcmd.org server of this project, in the same way as the other optional data of the package, see [download_optional_data](#). They are redistributed from the archives of the MIT licensed 'yabplot' Python package, which uses the same streamlines; the attribution and provenance files that come with them document the origin (see `tracts/xtract.attribution.json` and the per-level `tracts/xtract_<level>/xtract_<level>.provenance.json` in the file cache). This data is not required for the package to work.

Value

named list. The list has the entries "available" (vector of character strings, the paths of the bundle files that are available in the local file cache) and "missing" (vector of character strings, the files that could not be retrieved).

Note

The levels of detail do not only differ in the number of streamlines, but also in the set of bundles they contain: 'xtract_tiny' has 37 bundles, 'xtract_small' and 'xtract_medium' have 40, and 'xtract_large' has 42 (the bundles 'SLF1_L' and 'Cing_PeriGen_L' are only present in the large one). Since the bundles are selected by name (see the parameter 'bundle_values' of [vis.tracts](#)), data that is mapped to bundles has to match the atlas that is actually used.

See Also

Other tracts functions: [read.tract.bundles\(\)](#), [vis.tracts\(\)](#)

Examples

```
## Not run:
# Download the small version of the XTRACT atlas:
download_xtract_tracts("xtract_small");

# The bundle files are now in the cache:
atlas_dir = file.path(get_optional_data_filepath("tracts"), "xtract_small");
list.files(atlas_dir);

## End(Not run)
```

export

Export high-quality brainview image with a colorbar.

Description

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

Usage

```
export(
  coloredmeshes,
  colorbar_legend = NULL,
  img_only = TRUE,
  draw_colorbar = "horizontal",
  horizontal = NULL,
  silent = TRUE,
```

```

quality = 1L,
output_img = "fsbrain_arranged.png",
image.plot_extra_options = NULL,
large_legend = TRUE,
view_angles = get.view.angle.names(angle_set = "t4"),
style = "default",
grid_like = TRUE,
background_color = "white",
transparency_color = NULL,
...
)

```

Arguments

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

```

transparency_color
    hex color string (like '#FFFFFF'), the temporary background color that will get
    mapped to transparency, or NULL if you do not want a transparent background.
    If used, it can be any color that does not occur in the foreground. Try '#FFFFFF'
    (white) or '#000000' (black) if in doubt. WARNING: Do not use color names
    (like 'gray'), as their interpretation differs between rgl and image magick!
...
    extra arguments passed to vislayout.from.coloredmeshes.

```

Value

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

Note

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

Examples

```

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

## End(Not run)

```

```
export.coloredmesh.ply
```

Export a coloredmeshes with vertexcolors in PLY format.

Description

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

Usage

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

Arguments

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

Examples

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

## End(Not run)
```

face.edges

Enumerate all edges of the given faces or mesh.

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

find.freesurferhome	<i>Find the FREESURFER_HOME directory on disk.</i>
---------------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

[fs.home](#)

find.subjectsdir.of	<i>Find the subject directory containing the fsaverage subject (or others) on disk.</i>
---------------------	---

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. Note that a subject directory in the package cache is only considered complete if it contains the essential files of a subject, see [subject.dir.has.core.files](#): the cache can contain a subject which only has some atlas files (see [download_fsaverage_atlases](#)), and such a directory would make all functions that need the surfaces of the subject fail.

Usage

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

Arguments

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

Value

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

See Also

[fsaverage.path](#)

fs.coloredmesh	<i>fs.coloredmesh constructor</i>
----------------	-----------------------------------

Description

fs.coloredmesh constructor

Usage

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

Arguments

mesh	a mesh3d instance as returned by tmesh3d or an fs.surface brain surface mesh as returned by functions like subject.surface .
col	vector of vertex colors for the mesh, one color per vertex. Expanded if exactly one color.
hemi	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.

render	logical, whether to render this mesh during visualization
metadata	optional, named list containing metadata
add_normals	logical, whether to compute normals and save them in the mesh.

Value

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

fs.coloredpaths	<i>Create fs.coloredpaths instance from 2 point matrices.</i>
-----------------	---

Description

Create a renderable that draws the line segments from from to to, with one color and (optionally) one width per segment. This is the line equivalent of a coloredmesh, see the details.

Usage

```
fs.coloredpaths(
  from,
  to,
  col = "#FF0000",
  width = 1,
  depth_test = TRUE,
  lit = FALSE,
  hemi = NULL,
  metadata = list(),
  render = TRUE,
  style = NULL
)
```

Arguments

from	numeric matrix of size (n, 3), the start points of the n segments.
to	numeric matrix of size (n, 3), the end points of the n segments.
col	vector of hex color strings, either a single one or one per segment.
width	vector of positive numbers or a single one, the line width(s) in pixels.
depth_test	logical, whether the lines should be hidden by geometry which is closer to the camera. Defaults to TRUE. Set to FALSE to draw the lines on top of everything, which is useful for annotations.
lit	logical, whether to apply lighting to the lines. Defaults to FALSE, which means the color is used as-is (like hardware-rendered lines, and like segments3d without a lit material).

hemi	character string or NULL, the hemisphere this renderable belongs to. Defaults to NULL, which means that it is not hemisphere-specific and is rendered in the views of both hemispheres.
metadata	named list, metadata for this renderable. See the details.
render	logical, whether to render this instance. Meshes with render=FALSE are skipped by all rendering functions, see the parameter 'skip_all_na' of vis.coloredmeshes .
style	NULL or a rendering style for this renderable, see get.rglstyle .

Details

Both renderer backends support this class. The rgl backend renders the segments as hardware lines (`rgl::segments3d`), the headless scimesh backend uses a scimesh line layer (see `scimesh::line_layer`), which rasterizes the lines directly instead of creating tube geometry. Note that the width of lines is measured in pixels and is a screen-space property, so lines do not get thinner when the camera moves away (this is what hardware line rendering does, and it also means that a line is always visible, even when it is very thin).

The metadata field works exactly like the one of a `coloredmesh`: if it contains the entries 'src_data' and 'makecmap_options', a colorbar can be plotted for this renderable, see [coloredmesh.plot.colorbar.separate](#).

Value

`fs.coloredpaths` instance. A named list with entries: "from" (the start points), "to" (the end points), "col" (the colors), "width" (the line widths), "depth_test", "lit", "hemi", "render", and "metadata".

Examples

```
# A single white line segment from the origin to (10, 0, 0):
p = fs.coloredpaths(matrix(c(0, 0, 0), ncol = 3), matrix(c(10, 0, 0), ncol = 3), col = "#FFFFFF");
p;

# Two segments in different colors, both of them with a width of 2 pixels:
from = matrix(c(0, 0, 0, 0, 0, 10), ncol = 3, byrow = TRUE);
to = matrix(c(10, 0, 0, 0, 10, 10), ncol = 3, byrow = TRUE);
p2 = fs.coloredpaths(from, to, col = c("#FF0000", "#00FF00"), width = 2);
```

fs.home

Return FreeSurfer path.

Description

Return FreeSurfer path.

Usage

```
fs.home()
```

Value

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

Note

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

```
fs.surface.as.adjacencylist
```

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

Description

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

Usage

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

Arguments

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

Value

list of integer vectors, the adjacency list.

```
fs.surface.to.igraph
```

Create igraph undirected graph from a brain surface mesh.

Description

Create igraph undirected graph from a brain surface mesh.

Usage

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

Arguments

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

Value

igraph::graph instance

Examples

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

## End(Not run)
```

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

Description

Get an rgl tmesh3d instance from a brain surface mesh.

Usage

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

Arguments

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

Value

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

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

Description

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

Usage

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

Arguments

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

Value

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

Note

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

See Also

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

fsaverage.path	<i>Return path to fsaverage dir.</i>
----------------	--------------------------------------

Description

Return path to fsaverage dir.

Usage

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

Arguments

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

Value

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

Note

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

```
fsbrain.set.default.figsiz
```

Set default figure size for fsbrain visualization functions.

Description

Set default figure size for fsbrain visualization functions.

Usage

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

Arguments

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

Note

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

fup	<i>Transform first character of a string to uppercase.</i>
-----	--

Description

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

Usage

```
fup(word)
```

Arguments

word	string. Any string.
------	---------------------

Value

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

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

a 3d array of integers

Note

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

```
geod.patches.color.overlay
```

Generate color overlay from geodesic patches around several vertices.

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

## End(Not run)
```

geod.vert.neighborhood

Compute all vertices within given geodesic distance on the mesh.

Description

Compute all vertices within given geodesic distance on the mesh.

Usage

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

Arguments

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

Value

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

Note

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

Examples

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

## End(Not run)
```

geodesic.circles	<i>Compute geodesic circles and ball stats for given vertices.</i>
------------------	--

Description

Compute geodesic circles and ball stats for given vertices.

Usage

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

Arguments

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

Note

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

Examples

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

## End(Not run)
```

geodesic.dists.to.vertex

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

Description

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

Usage

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

Arguments

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

Value

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

geodesic.path

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

Description

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

Usage

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

Arguments

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

Value

list of integer vectors, the paths

Note

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

Examples

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

## End(Not run)
```

```
get.atlas.region.names
```

Determine atlas region names from a subject.

Description

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

Usage

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

Arguments

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

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

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

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

Value

vector of strings, the region names.

See Also

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

Examples

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

## End(Not run)
```

get.fsbrain.renderer.backend

Get the current fsbrain renderer backend

Description

Returns the current fsbrain renderer backend setting. When set to "scimesh", fsbrain will use the scimesh software renderer instead of rgl/OpenGL for image export. The default is "rgl". Set it with `options(fsbrain.renderer_backend = "scimesh")` at the start of your R session.

Usage

```
get.fsbrain.renderer.backend()
```

Value

character string, either "rgl" or "scimesh".

Note

Only functions that produce static PNG output are affected. Interactive viewers, animations, and the rglwidget (WebGL) always use rgl.

Examples

```
## Not run:
# Switch to the scimesh software renderer for headless environments
options(fsbrain.renderer_backend = "scimesh")

# Check current backend
get.fsbrain.renderer.backend()

# Switch back to rgl
options(fsbrain.renderer_backend = "rgl")

## End(Not run)
```

get.rglstyle

Get the default visualization style parameters as a named list.

Description

Run [material3d](#) without arguments to see valid style keywords to create new styles.

Usage

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

Arguments

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

Value

a style, resolved to a parameter list compatible with [material3d](#).

Note

In addition to the style names listed above, the magic word 'from_mesh' can be passed as parameter 'style' to the rendering functions: it makes them use the style stored in the 'style' field of each individual mesh (see [coloredmesh.from.color](#)). This allows you to give individual meshes of a scene their own look, e.g., to render a cortex mesh semi-transparently behind colored data meshes, see [vis.subcortical.region.values](#). Note that the scimesh renderer backend currently only supports the alpha channel of a style, not other material properties.

See Also

[shade3d](#) can use the returned style

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

Description

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

Usage

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

Arguments

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

Value

vector of character strings, all valid view angle strings.

getIn	<i>Retrieve values from nested named lists</i>
-------	--

Description

Retrieve values from nested named lists

Usage

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

Arguments

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

Value

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

Examples

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

get_optional_data_filepath	<i>Access a single file from the package cache by its file name.</i>
----------------------------	--

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 must-Work=FALSE and the file does not exist, the empty string is returned.

Value

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

group.agg.atlas.native

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

Description

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

Usage

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

Arguments

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

cache_file string or NULL. If given, it is interpreted as path of a file, and the data will be cached in the file `cache_file` in RData format. If the file does not exist yet, the function will run and cache the data in the file. If the file exists, the function will load the data from the file instead of running. The filename should end in '.RData', but that is not enforced or checked in any way. **WARNING:** If cached data is returned, all parameters passed to this function (with the exception of 'cache_file') are ignored! Whether the cached data is for another subjects_list or hemi is NOT checked! You have to ensure this yourself, by using different filenames. Defaults to NULL.

Value

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

See Also

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

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

Examples

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

## End(Not run)
```

group.agg.atlas.standard

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

Examples

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

## End(Not run)
```

group.annot	<i>Load annotations for a group of subjects.</i>
-------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

Examples

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

## End(Not run)
```

```
group.concat.measures.native
```

Concatenate native space data for a group of subjects.

Description

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

Usage

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

Arguments

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

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

## End(Not run)
```

```
group.concat.measures.standard
```

Concatenate standard space data for a group of subjects.

Description

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

Usage

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

Arguments

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

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

## End(Not run)
```

group.label	<i>Retrieve label data for a group of subjects.</i>
-------------	---

Description

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

Usage

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

Arguments

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

Value

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

See Also

Other label data functions: [labeldata.from.mask\(\)](#), [mask.from.labeldata.for.hemi\(\)](#), [subject.label\(\)](#)

Examples

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

## End(Not run)
```

group.label.from.annot

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

Description

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

Usage

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

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	vector of string. The subject identifiers.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
region	string. A valid region name for the annotation, i.e., one of the regions of the atlas.
return_one_based_indices	logical. Whether the indices should be 1-based. Indices are stored zero-based in label files, but R uses 1-based indices. Defaults to TRUE.

invert logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.

error_on_invalid_region logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

Value

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

See Also

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

group.morph.agg.native

Aggregate native space morphometry data over one hemisphere for a group of subjects.

Description

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

Usage

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

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , 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 <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) 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 <code>mean</code> function, you could set <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> 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 <code>agg_fun</code> .

Value

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

See Also

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

Examples

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

```
head(fullldata);

## End(Not run)
```

```
group.morph.agg.standard
```

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

Description

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

Usage

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

Arguments

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

template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cast	Whether a separate 'hemi' column should exist.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> 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 <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> 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 <code>agg_fun</code> .

Value

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

See Also

Other global aggregation functions: [group.morph.agg.native\(\)](#), [group.multimorph.agg.native\(\)](#), [group.multimorph.agg.standard\(\)](#)

group.morph.agg.standard.vertex

Aggregate standard space morphometry data over subjects.

Description

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

Usage

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

```

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

```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
agg_fun	function. An R function that aggregates data, typically max , mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> 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.
split_by_hemi	logical, whether to return a hemilist

See Also

Other aggregation functions: [group.agg.atlas.native\(\)](#), [group.agg.atlas.standard\(\)](#), [subject.atlas.agg\(\)](#)

group.morph.native	<i>Retrieve native space morphometry data for a group of subjects.</i>
--------------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

Examples

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

## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

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

Value

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

See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [subject.morph.native\(\)](#), [subject.morph.standard\(\)](#)

Examples

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

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

## End(Not run)
```

```
group.morph.standard.sf
```

Read combined data for a group from a single file.

Description

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

Usage

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

Arguments

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

Value

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

Note

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

See Also

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

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

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

Description

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

Usage

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

```

    cortex_only = FALSE,
    agg_fun_extra_params = NULL
  )

```

Arguments

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

Value

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

See Also

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

Examples

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

## End(Not run)
```

```
group.multimorph.agg.standard
```

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

Description

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

Usage

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

Arguments

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

measures	vector of strings. Names of the vertex-wise morphometry measures. E.g., <code>c("area", "thickness")</code> . 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 <code>max</code> , 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 <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) 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 <code>mean</code> function, you could set <code>agg_fun_extra_params=list("na.rm"=TRUE)</code> 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 <code>agg_fun</code> .

Value

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

See Also

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

group.surface	<i>Retrieve surface mesh data for a group of subjects.</i>
---------------	--

Description

Retrieve surface mesh data for a group of subjects.

Usage

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

Arguments

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

Value

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

Examples

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

## End(Not run)
```

groupmorph.split.hemilist

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

Description

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

Usage

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

Arguments

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

Value

hemilist of the data, split at the index.

Examples

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

## End(Not run)
```

hasIn

Check for values in nested named lists

Description

Check for values in nested named lists

Usage

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

Arguments

named_list a named list

listkeys vector of character strings, the nested names of the lists

Value

whether a non-NULL value exists at the path

Examples

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

hemilist	<i>Create a hemilist from lh and rh data.</i>
----------	---

Description

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

Usage

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

Arguments

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

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

Value

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

See Also

Other hemilist functions: [hemilist.derive.hemi\(\)](#), [hemilist.from.prefixed.list\(\)](#), [hemilist.get.combined.data](#), [hemilist.unwrap\(\)](#), [hemilist.wrap\(\)](#), [is.hemilist\(\)](#)

Examples

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

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

Description

Derive 'hemi' string from the data in a hemilist

Usage

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

Arguments

hemilist hemilist, an existing hemilist

Value

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

Note

See [hemilist](#) for details.

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.from.prefixed.list\(\)](#), [hemilist.get.combined.data\(\)](#), [hemilist.unwrap\(\)](#), [hemilist.wrap\(\)](#), [is.hemilist\(\)](#)

hemilist.from.prefixed.list

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

Description

A hemilist is a named list with entries 'lh' and/or 'rh', see [hemilist](#).

Usage

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

Arguments

`named_list` a named list, the keys must start with 'lh_' or 'rh_' to be assigned to the 'lh' and 'rh' entries of the returned hemilist. Other entries will be ignored.

`report_ignored` logical, whether to print a message with the ignored entries, if any.

`return_ignored` logical, whether to add a key 'ignored' to the returned hemilist, containing the ignored entries.

Value

a hemilist

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.derive.hemi\(\)](#), [hemilist.get.combined.data\(\)](#), [hemilist.unwrap\(\)](#), [hemilist.wrap\(\)](#), [is.hemilist\(\)](#)

```
hemilist.get.combined.data
```

Get combined data of hemi list

Description

Get combined data of hemi list

Usage

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

Arguments

`hemi_list` named list, can have entries 'lh' and/or 'rh', see [hemilist](#)

Value

the data combined with [c](#), or NULL if both entries are NULL.

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.derive.hemi\(\)](#), [hemilist.from.prefixed.list\(\)](#), [hemilist.unwrap\(\)](#), [hemilist.wrap\(\)](#), [is.hemilist\(\)](#)

hemilist.unwrap	<i>Unwrap hemi data from a named hemi list.</i>
-----------------	---

Description

Unwrap hemi data from a named hemi list.

Usage

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

Arguments

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

Value

the data

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.derive.hemi\(\)](#), [hemilist.from.prefixes.list\(\)](#), [hemilist.get.combined.data\(\)](#), [hemilist.wrap\(\)](#), [is.hemilist\(\)](#)

hemilist.wrap	<i>Wrap data into a named hemi list.</i>
---------------	--

Description

Wrap data into a named hemi list.

Usage

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

Arguments

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

Value

a [hemilist](#): a named list, with the 'data' in the name given by parameter 'hemi'

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.derive.hemi\(\)](#), [hemilist.from.prefixes.list\(\)](#), [hemilist.get.combined.data\(\)](#), [hemilist.unwrap\(\)](#), [is.hemilist\(\)](#)

highlight.points.spheres

Draw small 3D spheres at given points.

Description

Draw small 3D spheres at given points.

Usage

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

Arguments

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

See Also

Other 3d utility functions: [highlight.vertices.spheres\(\)](#), [vertex.coords\(\)](#)

highlight.vertices.on.subject

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

Description

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

Usage

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

Arguments

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

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

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

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

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

Examples

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

## End(Not run)
```

`highlight.vertices.on.subject.spheres`

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

Description

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

Usage

```
highlight.vertices.on.subject.spheres(
    subjects_dir,
    vis_subject_id,
    vertices,
    surface = "white",
    patch_size = 25,
    show_patch = TRUE,
    style = "glass2",
    export_img = NULL,
    sphere_colors = c("#FF0000"),
    sphere_radius = 3,
    ...
)
```

Arguments

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

Value

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

Note

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

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other surface visualization functions: [highlight.vertices.on.subject\(\)](#), [vis.color.on.subject\(\)](#)

Examples

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

## End(Not run)
```

highlight.vertices.spheres

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

Description

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

Usage

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

Arguments

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

Note

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

See Also

Other 3d utility functions: [highlight.points.spheres\(\)](#), [vertex.coords\(\)](#)

Examples

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

## End(Not run)
```

images.dimmax

Compute max width and height of magick images.

Description

Compute max width and height of magick images.

Usage

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

Arguments

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

Value

named list with entries 'width' and 'height'

index2ras_tkr

The affine matrix that maps 1-based R array indices to surface RAS.

Description

Meshes and coordinates that were computed from a volume *array* in R are expressed in **1-based R array indices**: the first voxel of the volume is at index 1, the second at index 2, and so on. This is the convention of the iso-surfaces returned by [shell.extract.mesh](#) and [volvis.contour](#), and of the indices returned by `which(volume != 0, arr.ind = TRUE)`.

The FreeSurfer [vox2ras_tkr](#) matrix, in contrast, expects **0-based CRS** (column, row, slice) indices: the first voxel of the volume is at CRS (0, 0, 0), and (as the name says) the *voxel* index is mapped to RAS. Feeding 1-based R indices to `vox2ras_tkr()` shifts the result by one voxel (1 mm for a conformed volume), which is why volume data seemed to be slightly offset from the brain surface it was rendered with.

This function returns the matrix which maps 1-based R array indices to surface RAS, i.e., `vox2ras_tkr() %% translation.matrix(-1, -1, -1)`. It is used internally by the volume visualization functions, and can be used to transform the meshes returned by [volvis.contour](#) (or `misc3d::contour3d`) so that they are aligned with surface renderings of the same subject.

Usage

```
index2ras_tkr()
```

Value

numeric 4x4 matrix, the mapping from 1-based R array indices to surface RAS.

See Also

Other surface and volume coordinates: [ras2vox_tkr\(\)](#), [vox2ras_tkr\(\)](#)

Examples

```
# The first voxel of a conformed volume (R index 1) is at CRS (0, 0, 0):
(index2ras_tkr() %% c(1, 1, 1, 1))[1:3];
# which is the position reported by the vox2ras_tkr matrix for CRS (0, 0, 0):
(vox2ras_tkr() %% c(0, 0, 0, 1))[1:3];
```

is.fs.coloredmesh	<i>Check whether object is an fs.coloredmesh (S3)</i>
-------------------	---

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.coloredpaths	<i>Check whether object is an fs.coloredpaths instance (S3)</i>
--------------------	---

Description

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

Usage

```
is.fs.coloredpaths(x)
```

Arguments

x	any R object
---	--------------

Value

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

is.fs.coloredvoxels	<i>Check whether object is an fs.coloredvoxels instance (S3)</i>
---------------------	--

Description

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

Usage

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

Arguments

x	any R object
---	--------------

Value

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

is.fsbrain	<i>Check whether object is an fsbrain (S3)</i>
------------	--

Description

Check whether object is an fsbrain (S3)

Usage

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

Arguments

x	any R object
---	--------------

Value

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

is.hemilist	<i>Check whether x is a hemilist</i>
-------------	--------------------------------------

Description

A hemilist is a named list with entries 'lh' and/or 'rh', see [hemilist](#).

Usage

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

Arguments

x	any R object
---	--------------

Value

whether 'x' is a hemilist

See Also

Other hemilist functions: [hemilist\(\)](#), [hemilist.derive.hemi\(\)](#), [hemilist.from.prefixes.list\(\)](#), [hemilist.get.combined.data\(\)](#), [hemilist.unwrap\(\)](#), [hemilist.wrap\(\)](#)

label.border	<i>Compute border of a label.</i>
--------------	-----------------------------------

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

Value

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

See Also

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

label.colFn	<i>A simple colormap function for binary colors.</i>
-------------	--

Description

Useful for plotting labels.

Usage

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

Arguments

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

Value

vector of 'n' RGB colorstrings

label.colFn.inv	<i>A simple colormap function for binary colors.</i>
-----------------	--

Description

Useful for plotting labels.

Usage

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

Arguments

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

Value

vector of 'n' RGB colorstrings

label.from.annotdata	<i>Extract a region from an annotation as a label.</i>
----------------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

label.to.annot	<i>Merge several labels into an annotation</i>
----------------	--

Description

Merge several labels and a colortable into an annotation.

Usage

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

Arguments

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

Value

an annotation, see [read.fs.annot](#) for details.

See Also

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

Examples

```
# Create two labels. Real-word labels would have more vertices, of course.
label1 = c(46666, 46777);
label2 = c(99888, 99889);
label_vertices = list("region1"=label1, "region2"=label2);
colortable_df = data.frame("struct_index"=seq(0, 2),
  "struct_name"=c("unknown", "region1", "region2"),
  "r"=c(255L, 255L, 0L), "g"=c(255L, 0L, 255L), "b"=c(255L, 0L, 0L), "a"=c(0L, 0L, 0L));
annot = label.to.annot(label_vertices, 100000, colortable_df);
```

labeldata.from.mask	<i>Create labeldata from a mask.</i>
---------------------	--------------------------------------

Description

Create labeldata from a mask. This function is trivial and only calls [which](#) after performing basic sanity checks.

Usage

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

Arguments

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

Value

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

See Also

Other label data functions: [group.label\(\)](#), [mask.from.labeldata.for.hemi\(\)](#), [subject.label\(\)](#)

limit_fun

Get data limiting function.

Description

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

Usage

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

Arguments

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

Value

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

See Also

[rglactions](#)

Examples

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

limit_fun_na

Get data limiting function to NA.

Description

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

Usage

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

Arguments

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

Value

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

Note

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

See Also

[limit_fun_na_inside](#) which will set the values inside the range to NA.

Examples

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

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

Description

Get data limiting function to use in [rglactions](#) as 'trans_fun' to transform data.

Usage

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

Arguments

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

Value

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

Note

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

See Also

[limit_fun_na](#) which will set the values outside the range to NA.

Examples

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

list_optional_data	<i>Get file names available in package cache.</i>
--------------------	---

Description

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

Usage

```
list_optional_data()
```

Value

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

mask.from.labeldata.for.hemi	<i>Create a binary mask from labels.</i>
------------------------------	--

Description

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

Usage

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

Arguments

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

Value

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

See Also

Other label data functions: [group.label\(\)](#), [labeldata.from.mask\(\)](#), [subject.label\(\)](#)

Other mask functions: [coloredmesh.from.mask\(\)](#), [vis.mask.on.subject\(\)](#)

Examples

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

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

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

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

## End(Not run)
```

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

mkco.cluster	<i>Return recommended 'makecmap_options' for diverging cluster data.</i>
--------------	--

Description

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

Usage

```
mkco.cluster()
```

Value

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

Note

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

mkco.div	<i>Return recommended 'makecmap_options' for diverging data.</i>
----------	--

Description

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

Usage

```
mkco.div()
```

Value

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

mkco.heat	<i>Return recommended 'makecmap_options' for sequential data with heatmap style.</i>
-----------	--

Description

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

Usage

```
mkco.heat()
```

Value

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

mkco.seq	<i>Return recommended 'makecmap_options' for sequential data.</i>
----------	---

Description

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

Usage

```
mkco.seq()
```

Value

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

numverts.lh	<i>Determine vertex count of left hemi from hemilist of surfaces or the count itself.</i>
-------------	---

Description

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

Usage

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

Arguments

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

Value

integer, the number of vertices.

numverts.rh	<i>Determine vertex count of right hemi from hemilist of surfaces or the count itself.</i>
-------------	--

Description

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

Usage

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

Arguments

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

Value

integer, the number of vertices.

principal.curvatures	<i>Computes principal curvatures according to 2 definitions from raw k1 and k2 values.</i>
----------------------	--

Description

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

Usage

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

Arguments

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

Value

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

Note

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

print.fs.coloredmesh	<i>Print description of a brain coloredmesh (S3).</i>
----------------------	---

Description

Print description of a brain coloredmesh (S3).

Usage

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

Arguments

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

print.fs.coloredpaths *Print description of an fs.coloredpaths instance (S3).*

Description

Print description of an fs.coloredpaths instance (S3).

Usage

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

Arguments

x	an fs.coloredpaths instance.
...	further arguments passed to or from other methods.

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

Description

Print description of fs.coloredvoxels (S3).

Usage

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

Arguments

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

print.fsbrain	<i>Print description of an fsbrain (S3).</i>
---------------	--

Description

Print description of an fsbrain (S3).

Usage

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

Arguments

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

qc.for.group	<i>Perform data quality check based on computed region stats.</i>
--------------	---

Description

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

Usage

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

Arguments

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

Value

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

See Also

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

Other quality check functions: [qc.for.group\(\)](#), [qc.from.segstats.table\(\)](#)

```
qc.from.segstats.tables
```

Perform data quality check based on a segstats table.

Description

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

Usage

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

Arguments

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

Value

qc result as a hemilist, each entry contains a named list as returned by [qc.from.regionwise.df](#).

```
qc.vis.failcount.by.region
```

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

Description

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

Usage

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

Arguments

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

Note

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

qdec.table.skeleton	<i>Generate skeleton dataframe for FreeSurfer QDEC long file from subjects list.</i>
---------------------	--

Description

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

Usage

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

Arguments

`subjects_list` vector of character strings, the FreeSurfer subject IDs (cross-sectional names, without any suffixes like `_MR1`, `long`, etc.)

`isi` numerical vector, the inter-scan interval for the subjects, in a unit of your choice. Typically in years.

`isi_name` character string, the name for the `isi` columns. Defaults to "years".

`timepoint_names` vector of character strings, the timepoint names. These are mandatory for QDEC, so there should be very little reason to change them. Leave along unless you know what you are doing.

Value

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

See Also

The function [demographics.to.qdec.table.dat](#) to write the result to a QDEC file.

Examples

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

ras2vox_tkr

The FreeSurfer default ras2vox_tkr matrix.

Description

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

Usage

```
ras2vox_tkr()
```

Value

numeric 4x4 matrix, the FreeSurfer `ras2vox_tkr` matrix.

See Also

Other surface and volume coordinates: [index2ras_tkr\(\)](#), [vox2ras_tkr\(\)](#)

Examples

```
# Compute the FreeSurfer CRS voxel index of surface RAS coordinate (0.0, 0.0, 0.0):
ras2vox_tkr() %*% c(0, 0, 0, 1);
# Show that the voxel at surface RAS corrd (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	<i>Read colors from CSV file.</i>
---------------	-----------------------------------

Description

Read colors from CSV file.

Usage

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

Arguments

filepath character string, path to a CSV file containing colors

Value

vector of hex color strings

read.md.demographics	<i>Read demographics file</i>
----------------------	-------------------------------

Description

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

Usage

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

Arguments

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

Value

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

See Also

Other metadata functions: [demographics.to.fsgd.file\(\)](#), [read.md.subjects\(\)](#), [report.on.demographics\(\)](#)

Examples

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

read.md.subjects	<i>Read subjects file</i>
------------------	---------------------------

Description

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

Usage

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

Arguments

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

Value

vector of strings, the subject identifiers.

See Also

Other metadata functions: [demographics.to.fsgd.file\(\)](#), [read.md.demographics\(\)](#), [report.on.demographics\(\)](#)

Examples

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

```
read.md.subjects.from.fsgd
```

Read subjects list from an FSGD file.

Description

Read subjects list from an FSGD file.

Usage

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

Arguments

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

Value

vector of character strings, the subject identifiers

Note

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

See Also

[demographics.to.fsgd.file](#)

`read.tract.bundles` *Read tract (streamline) files into named bundles.*

Description

Reads one or more tract files in TRK or TCK format (see the details) and returns them as a named list of bundles, i.e., one entry per file. This is the input format of [vis.tracts](#). A single directory can be passed to read a whole tract atlas (one file per bundle, e.g. the XTRACT atlas, see [download_xtract_tracts](#)).

Usage

```
read.tract.bundles(
  paths,
  coords = "ras",
  transform_matrix = NULL,
  max_tracks = Inf,
  skip_tracks = 0L,
  bbox = NULL,
  pattern = "\\.(trk|tck)(\\.gz)?$",
  silent = FALSE
)
```

Arguments

<code>paths</code>	character string, either the path to a directory containing tract files, or a vector of paths to tract files.
<code>coords</code>	character string or NULL, the coordinate system of the returned coordinates for TRK files, see read.dti.trk . The default 'ras' returns RAS+ mm coordinates, which is required to combine the tracts with the fsbrain template surfaces: 'native' returns the coordinates as stored, which for TRK files is usually a voxel grid, so the result will be mirrored when plotted. Note that this parameter has no effect for TCK files, which do not store a transformation and are always returned as stored.
<code>transform_matrix</code>	a 4x4 numeric matrix or NULL. If given, it is applied to the coordinates of all files after reading them, e.g. to move a tract atlas from MNI space into the space of the surfaces, see the note in vis.tracts .
<code>max_tracks</code>	numeric, the maximum number of streamlines to read per file. Use Inf (the default) to read all of them. Reading a subset is the way to handle whole-brain tractograms, which can contain millions of streamlines and cannot be plotted (or held in memory) as a whole.
<code>skip_tracks</code>	integer, the number of streamlines to skip per file, see read.dti.tck .
<code>bbox</code>	numeric vector of length 6 or NULL, a bounding box to select the streamlines that pass through a region, see read.dti.tck . The box is interpreted in the coordinate system given by <code>coords</code> .

pattern	character string, a regular expression to select the files in a directory. Ignored if paths is not a directory.
silent	logical, whether to suppress the progress messages.

Details

The files are read with the readers of the 'freesurferformats' package. Both the TrackVis TRK format (used by FSL, DSI Studio, AFQ, TractSeg and the XTRACT atlas) and the MRtrix3 TCK format (used by MRtrix3, QSIREcon, MRtrix3_connectome) are supported; TCK files may be gzip-compressed, TRK files may not. The bundle names are the file names without the extension, e.g. 'CST_L.trk' becomes the bundle 'CST_L'.

Note that TRK files store a transformation matrix which is not necessarily the identity: a TRK file can contain coordinates in a voxel grid (e.g. all files of the XTRACT atlas do). With the default `coords = "ras"`, the coordinates are returned in RAS+ mm space, which is what the template surfaces of fsbrain are in, see the parameter documentation. Pass `coords = "native"` to get the coordinates exactly as stored in the file.

Value

named list of `fs.tracts` instances, one per file (see [fs.tracts](#)). The names are the file names without the extension. The attribute 'files' contains the file paths, and the attribute 'coords_space' the coordinate system of the coordinates ('ras' or 'native', see the parameter `coords`). An entry which was read from a TCK file is a TCK entry, i.e., its `[[i]]` entries are coordinate matrices, while the entries of a TRK file also contain the per-point scalars and per-track properties of the file.

See Also

Other tracts functions: [download_xtract_tracts\(\)](#), [vis.tracts\(\)](#)

Examples

```
## Not run:
# Read a whole tract atlas from a directory, one bundle per file:
atlas_dir = file.path(get_optional_data_filepath("tracts"), "xtract_tiny");
bundles = read.tract.bundles(atlas_dir);
names(bundles);
fs.tracts.lengths(bundles$CST_L);

# Read a single bundle:
cst = read.tract.bundles(file.path(atlas_dir, "CST_L.trk"));

## End(Not run)
```

regions.to.ignore	<i>Give suggestions for regions to ignore for an atlas.</i>
-------------------	---

Description

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

Usage

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

Arguments

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

Value

vector of strings, the region names.

See Also

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

Examples

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

```
report.on.demographics
```

Print a demographics report

Description

Print a demographics report

Usage

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

Arguments

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

Value

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

See Also

Other metadata functions: [demographics.to.fsgd.file\(\)](#), [read.md.demographics\(\)](#), [read.md.subjects\(\)](#)

rglactions	<i>Create rglactions list, suitable to be passed as parameter to vis functions.</i>
------------	---

Description

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

Usage

```
rglactions()
```

Value

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

Note

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

Examples

```

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

```

rglo*Get rgloptions and consider global options.*

Description

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

Usage

```
rglo()
```

Value

named list, usable as 'rgloptions' parameter for vis functions like [vis.subject.morph.native](#).

Note

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

rglot*Get rgloptions for testing.*

Description

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

Usage

```
rglot()
```

Value

named list, usable as 'rgloptions' parameter for vis functions like [vis.subject.morph.native](#).

rglvoxels

Draw 3D boxes at locations using rgl.

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

## End(Not run)
```

scale01	<i>Scale given values to range 0..1.</i>
---------	--

Description

Scale given values to range 0..1.

Usage

scale01(x, ...)

Arguments

x	the numeric data
...	the numeric data

Value

the scaled data

shape.descriptor.names	<i>Get all shape descriptor names.</i>
------------------------	--

Description

Get all shape descriptor names.

Usage

shape.descriptor.names()

Value

vector of character strings, the names

shape.descriptors	<i>Computes geometric curvature-based descriptors.</i>
-------------------	--

Description

Computes geometric curvature-based descriptors.

Usage

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

Arguments

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

Value

dataframe of descriptor values, each columns contains one descriptor.

References

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

shift.hemis.apart	<i>Shift hemispheres apart.</i>
-------------------	---------------------------------

Description

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

Usage

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

Arguments

<code>coloredmeshes_h1</code>	hemilist of coloredmeshes
<code>shift_by</code>	numerical vector of length 2, the amount by which to shift the hemis. The first value is for the left hemi, the second for the right hemi (values can be negative). Pass NULL to determine the shift automatically from the mesh coordinates, and adapt 'hemi_order_on_axis' to define how that happens.
<code>axis</code>	positive integer, one of 1L, 2L or 3L. The axis on which to shift (x,y,z).
<code>hemi_order_on_axis</code>	character string, one of 'auto', 'auto_flipped', 'lr' or 'rl'. Defines how to determine the order of the hemis on the axes. This is ignored unless 'shift_by' is NULL. The 'auto' setting assumes that the hemisphere with the smaller minimal vertex coordinate (on the given axis) comes first. Note that if the overlap (or shift) is extreme, this may not hold anymore. Therefore, the default value is 'lr', which works for FreeSurfer data. The 'auto_flipped' setting will always return the inverse of 'auto', so if 'auto' did not work, 'auto_flipped' will.
<code>min_dist</code>	numerical scalar, the minimal distance of the hemispheres. Ignored unless 'shift_by' is NULL.

Value

hemilist of coloredmeshes, the shifted meshes

sjd.demo

Download optional demo data if needed and return its path.

Description

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

Usage

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

Arguments

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

Value

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

Note

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

 sjld

Get subjects list from subjects.txt file in dir.

Description

Get subjects list from subjects.txt file in dir.

Usage

```
sjld(subjects_dir)
```

Arguments

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

Value

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

Note

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

 spread.values.over.annot

Spread a single value for a region to all region vertices.

Description

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

Usage

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

Arguments

annot annotation. The result of calling `fs.read.annot`.

region_value_list named list of strings. Each name must be a region name from the annotation, and the value must be the value to spread to all region vertices.

value_for_unlisted_regions numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in `region_value_list`. Defaults to `NaN`.

warn_on_unmatched_list_regions logical. Whether to print a warning when a region occurs in the `region_value_list` that is not part of the given atlas (and the value assigned to this region is thus ignored in the output file and data). Defaults to `FALSE`.

warn_on_unmatched_atlas_regions logical. Whether to print a warning when a region occurs in the atlas that is not part of the given `region_value_list` (and thus the vertices of the region will be assigned the value 'value_for_unlisted_regions' in the output file and data). Defaults to `FALSE`.

Value

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

See Also

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

Examples

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

## End(Not run)
```

```
spread.values.over.hemi
```

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

Description

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

Usage

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

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier
<code>hemi</code>	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
<code>atlas</code>	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
<code>region_value_list</code>	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list or vector, but then the length must exactly match the number of regions in the atlas, and the order must match the annotation file of the subject and hemisphere. Use with care, and keep in mind that some subjects do not have all regions.
<code>value_for_unlisted_regions</code>	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.
<code>silent</code>	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).

Value

numeric vector containing the data.

See Also

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

Examples

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

## End(Not run)
```

`spread.values.over.subject`

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

Description

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

Usage

```
spread.values.over.subject(
  subjects_dir,
  subject_id,
  atlas,
  lh_region_value_list,
  rh_region_value_list,
  value_for_unlisted_regions = NaN,
  silent = FALSE
)
```

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier

atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
lh_region_value_list	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the left hemisphere.
rh_region_value_list	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region. Applied to the right hemisphere.
value_for_unlisted_regions	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.
silent	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).

Value

named list with entries 'lh' and 'rh'. Each value is a numeric vector containing the data for the respective hemisphere.

See Also

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

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
rh_region_value_list = list("bankssts"=0.5);
morph_like_data =
  spread.values.over.subject(subjects_dir, 'subject1', 'aparc',
    lh_region_value_list, rh_region_value_list);

## End(Not run)
```

subject.annot

Load an annotation for a subject.

Description

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

Usage

```
subject.annot(subjects_dir, subject_id, hemi, atlas)
```

Arguments

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

Value

the annotation, as returned by `read.fs.annot`. It is a named list, 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.lobes()`

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot_lh = subject.annot(subjects_dir, "subject1", "lh", "aparc");

## End(Not run)
```

subject.annot.border *Compute annot border vertices.*

Description

Compute annot border vertices.

Usage

```
subject.annot.border(
    subjects_dir,
    subject_id,
    hemi,
    atlas,
    surface = "white",
    expand_inwards = 0L,
    limit_to_regions = NULL
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
expand_inwards	integer, additional thickness of the borders. Increases computation time, defaults to 0L.
limit_to_regions	vector of character strings or NULL, a list of regions for which to draw the outline (see get.atlas.region.names). If NULL, all regions will be used. If (and only if) this parameter is used, the 'outline_color' parameter can be a vector of color strings, one color per region.

Value

hemilist of integer vectors, the vertices in the border

subject.atlas.agg	<i>Aggregate morphometry data over brain atlas regions for a subject.</i>
-------------------	---

Description

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

Usage

```
subject.atlas.agg(  
  vertex_morph_data,  
  vertex_label_names,  
  agg_fun = base::mean,  
  requested_label_names = c()  
)
```

Arguments

vertex_morph_data	numeric vector. The morphometry data, one value per vertex. The morphometry data are typically loaded from an MGZ or curv format file with the <code>read.fs.curv</code> or <code>read.fs.mgh</code> 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 <code>read.fs.annot</code> function.
agg_fun	function. An R function that aggregates data, typically <code>max</code> , <code>mean</code> , <code>min</code> or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to <code>base::mean</code> .
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 <code>NaN</code> . If given, and one of the names from the data does NOT occur in the requested list, it will NOT occur in the output. So if you specify this, the output dataframe will contain a row for a region if and only if it is in the requested list.

Value

dataframe with aggregated values for all regions, with 2 columns and `n` rows, where `n` is the number of effective regions. The columns are: "region": string, contains the region name. "aggregated": numeric, contains the result of applying `agg_fun` to the morphometry data in that region.

See Also

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

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

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
agg = subject.atlas.agg(morph_data, annot$label_names);

## End(Not run)
```

`subject.filepath.morph.native`

Construct filepath of native space morphometry data file.

Description

Construct filepath of native space morphometry data file.

Usage

```
subject.filepath.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  format = "curv",
  warn_if_nonexistent = FALSE,
  error_if_nonexistent = FALSE
)
```

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier
<code>measure</code>	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.

hemi	string, one of 'lh' or 'rh'. The hemisphere name.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'curv'.
warn_if_nonexistent	logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.
error_if_nonexistent	logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

Value

string, the file path.

subject.filepath.morph.standard
Construct filepath of standard space morphometry data file.

Description

Construct filepath of standard space morphometry data file.

Usage

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

Arguments

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

format string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.

warn_if_nonexistent logical. Whether to print a warning if the file does not exist or cannot be accessed. Defaults to FALSE.

error_if_nonexistent logical. Whether to raise an error if the file does not exist or cannot be accessed. Defaults to FALSE.

Value

string, the file path. (Or a vector if 'subject_id' is a vector.)

subject.label	<i>Retrieve label data for a single subject.</i>
---------------	--

Description

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

Usage

```
subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  return_one_based_indices = TRUE,
  full = FALSE
)
```

Arguments

subjects_dir string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.

subject_id string. The subject identifier

label string. Name of the label file, without the hemi part. You can include the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'. You can also pass just the label (e.g., 'cortex'): if the string does not end with the suffix '.label', that suffix gets added 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.

`full` logical, whether to return the full label structure instead of only the vertex indices.

Value

integer vector with label data: the list of vertex indices in the label. See 'return_one_based_indices' for important information. If parameter `hemi` is set to both, a named list with entries `lh` and `rh` is returned, and the values of are the respective labels.

See Also

Other label data functions: [group.label\(\)](#), [labeldata.from.mask\(\)](#), [mask.from.labeldata.for.hemi\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cortex_lh = subject.label(subjects_dir, "subject1", "cortex.label", "lh");

## End(Not run)
```

`subject.label.from.annot`

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

Description

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

Usage

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

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier.
<code>hemi</code>	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the label data files to be loaded.
<code>atlas</code>	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
<code>region</code>	string. A valid region name for the annotation, i.e., one of the regions of the atlas.
<code>return_one_based_indices</code>	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.
<code>invert</code>	logical. If TRUE, return the indices of all vertices which are NOT part of the region. Defaults to FALSE.
<code>error_on_invalid_region</code>	logical. Whether to throw an error if the given region does not appear in the region list of the annotation. If set to FALSE, this will be ignored and an empty vertex list will be returned. Defaults to TRUE.

Value

integer vector with label data: the list of vertex indices in the label.

See Also

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

<code>subject.lobes</code>	<i>Load labels representing brain lobes.</i>
----------------------------	--

Description

This gives you labels that represent brain lobes for a subject. The lobe definition is based on the Desikan-Killiany atlas (Desikan *et al.*, 2010) as suggested on the FreeSurfer website at <https://surfer.nmr.mgh.harvard.edu/fsw>

Usage

```
subject.lobes(  
  subjects_dir,  
  subject_id,  
  hemi = "both",  
  include_cingulate = TRUE,  
  as_annot = FALSE  
)
```

Arguments

- subjects_dir string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
- subject_id string. The subject identifier
- hemi string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
- include_cingulate logical, whether to include the vertices of the cingulate in the lobes
- as_annot return a hemilist of annotations instead of the return value described in the *value* section

Value

hemilist of integer vectors, the vectors represent vertex indices of the hemispheres, and each vertex is assigned one of the following values: 0=no_lobe, 1=frontal, 2=parietal, 3=temporal, 4=occipital.

See Also

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

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

subject.mask	<i>Compute a mask for a subject.</i>
--------------	--------------------------------------

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

<code>subjects_dir</code>	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.
<code>subject_id</code>	string. The subject identifier
<code>hemi</code>	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.
<code>from_label</code>	string, the label file to use. Defaults to 'cortex', which will result in a mask of the medial wall versus cortex vertices.
<code>surf_num_verts</code>	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).
<code>invert_mask</code>	logical, whether to invert the mask. E.g., when the mask is loaded from the cortex labels, if this is set to FALSE, the cortex would be masked (set to 0 in the final mask). If you want everything but the cortex to be masked (set to 0), you should set this to TRUE. Defaults to TRUE.

Value

the mask, a logical vector with the length of the vertices in the surface. If parameter hemi is set to both, a named list with entries lh and rh is returned, and the values of are the respective masks.

See Also

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.subject.label\(\)](#)

Examples

```

## Not run:
# Generate a binary mask of the medial wall. Wall vertices will
# be set to 0, cortex vertices will be set to 1.
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
mask = subject.mask(subjects_dir, "subject1");

```

```

# Print some information on the mask:
#cat(sprintf("lh: %d verts, %d in cortex, %d medial wall.\n", length(mask$lh),
# sum(mask$lh), (length(mask$lh)- sum(mask$lh))))
# Output: lh: 149244 verts, 140891 in cortex, 8353 medial wall.
# Now visualize the mask to illustrate that it is correct:
vis.mask.on.subject(subjects_dir, "subject1", mask$lh, mask$rh);

## End(Not run)

```

subject.morph.native *Retrieve native space morphometry data for a single subject.*

Description

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

Usage

```

subject.morph.native(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  format = "curv",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)

```

Arguments

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

Value

vector with native space morph data, as returned by [read.fs.morph](#).

See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.standard\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

# Load the full data:
thickness_lh = subject.morph.native(subjects_dir, "subject1", "thickness", "lh");
mean(thickness_lh); # prints 2.437466

# Load the data again, but this time exclude the medial wall:
thickness_lh_cortex = subject.morph.native(subjects_dir, "subject1", "thickness",
"lh", cortex_only=TRUE);
mean(thickness_lh_cortex, na.rm=TRUE); # prints 2.544132
vis.data.on.subject(subjects_dir, "subject1", thickness_lh_cortex, NULL);

## End(Not run)
```

subject.morph.standard

Retrieve standard space morphometry data for a single subject.

Description

Load standard space morphometry data (like 'surf/lh.area.fwhm10.fsaverage.mgh') for a subject from disk. Uses knowledge about the FreeSurfer directory structure to load the correct file.

Usage

```
subject.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  hemi,
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  cortex_only = FALSE,
  split_by_hemi = FALSE
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file label/?h.cortex.label) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the template subject. Defaults to FALSE.
split_by_hemi	logical, whether the returned data should be encapsulated in a named list, where the names are from 'lh' and 'rh', and the values are the respective data.

Value

vector with standard space morph data

See Also

Other morphometry data functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [group.morph.native\(\)](#), [group.morph.standard\(\)](#), [subject.morph.native\(\)](#)

Examples

```
## Not run:
  fsbrain::download_optional_data();
  subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
  thickness_lh = subject.morph.standard(subjects_dir, "subject1", "thickness", "lh", fwhm='10');

## End(Not run)
```

subject.num.verts	<i>Get subjects vertex count.</i>
-------------------	-----------------------------------

Description

Determine vertex counts for the brain meshes of a subject.

Usage

```
subject.num.verts(
    subjects_dir,
    subject_id,
    surface = "white",
    hemi = "both",
    do_sum = FALSE
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
surface	string. The surface name. E.g., "white", or "pial". Used to construct the name of the surface file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
do_sum	logical, whether to return the sum of the vertex counts for lh and rh. Ignored unless 'hemi' is 'both'. If set, a single scalar will be returned.

Value

hemilist of integers, the vertex count. If hemi is 'both' and 'do_sum' is FALSE, a hemilist of integers is returned. Otherwise, a single integer.

subject.region.centroids	<i>Compute the centroid of every region of an atlas on a surface.</i>
--------------------------	---

Description

Computes, for every region of an atlas (annotation), the mean position of the surface vertices that belong to the region. These centroids are the node positions of a connectome visualization, see [vis.connectome](#). This is the equivalent of the `_extract_centroids()` function of the Python package yabplot.

Usage

```
subject.region.centroids(
  subjects_dir,
  subject_id,
  atlas,
  surface = "midthickness",
  hemi = "both",
  exclude_regions = c("unknown", "medialwall", "")
)
```

Arguments

<code>subjects_dir</code>	character string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all subjects, each in a subdir named after the subject identifier. Can be NULL for a template subject (like <code>fs_LR_32</code> or <code>fsaverage</code>), in which case the standard locations (package cache, FreeSurfer installation) are searched.
<code>subject_id</code>	character string, the subject identifier. For a template space, this is the name of the template subject, e.g., <code>'fs_LR_32'</code> or <code>'fsaverage'</code> .
<code>atlas</code>	character string, the atlas name, e.g., <code>'schaefer400'</code> , <code>'brainnetome'</code> or <code>'aal3'</code> . Used to construct the name of the annotation file.
<code>surface</code>	character string, the name of the surface on which the centroids are computed. Defaults to <code>'midthickness'</code> (the surface in the middle between white and pial matter), which is the surface usually used for connectome figures.
<code>hemi</code>	character string, one of <code>'lh'</code> , <code>'rh'</code> , or <code>'both'</code> . Defaults to <code>'both'</code> .
<code>exclude_regions</code>	vector of character strings, the names of atlas regions to exclude. Defaults to the medial wall and the unknown region, which are not real brain regions.

Value

data.frame with one row per region, the columns `'x'`, `'y'`, `'z'` (the centroid coordinates), `'region'` (the region name) and `'hemi'` (the hemisphere). The row names are the region names, which are unique across hemispheres for the atlases that are distributed by `fsbrain` (see the example).

See Also

Other connectome functions: [vis.connectome\(\)](#)

Examples

```
## Not run:
# For the fs_LR_32 template (HCP-style), download the data first:
fsbrain::download_fs_LR_32_meshes();
fsbrain::download_fs_LR_32_atlases();
centroids = subject.region.centroids(subjects_dir = NULL, subject_id = "fs_LR_32",
  atlas = "schaefer400", surface = "midthickness");
head(centroids);
```

```
## End(Not run)
```

subject.surface	<i>Load a surface for a subject.</i>
-----------------	--------------------------------------

Description

Load a brain surface mesh for a subject.

Usage

```
subject.surface(
  subjects_dir,
  subject_id,
  surface = "white",
  hemi = "both",
  force_hemilist = FALSE,
  as_tm = FALSE
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
surface	string. The surface name. E.g., "white", or "pial". Used to construct the name of the surface file to be loaded.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the surface file to be loaded. For 'both', see the information on the return value.
force_hemilist	logical, whether to return a hemilist even if the 'hemi' parameter is not set to 'both'
as_tm	logical, whether to return an <code>rgl::tmesh3d</code> instead of an <code>fs.surface</code> instance by applying the <code>fs.surface.to.tmesh3d</code> function.

Value

the `fs.surface` instance, as returned by [read.fs.surface](#). If parameter `hemi` is set to `both`, a named list with entries `lh` and `rh` is returned, and the values of are the respective surfaces. The mesh data structure used in `fs.surface` is a *face index set*.

See Also

Other surface mesh functions: [face.edges\(\)](#), [label.border\(\)](#), [mesh.vertex.included.faces\(\)](#), [mesh.vertex.neighbors\(\)](#), [vis.path.along.verts\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_white = subject.surface(subjects_dir, "subject1", "white", "lh");

## End(Not run)
```

subject.vol2surf	<i>Project a volume onto the cortical surface of a subject using its own (native) space.</i>
------------------	--

Description

Samples a brain volume at the positions of the cortical surface vertices of a subject and returns the resulting per-vertex values. This is the equivalent of the FreeSurfer command line tool `mri_vol2surf` (or of the `project_vol2surf` function of the Python package `yabplot`), implemented in R. The volume and the surface must be defined in the same coordinate space, which is the case for the volumes in the `mri` directory of a subject (e.g., `brain.mgz`) and the surfaces in its `surf` directory (e.g., `lh.white`) – both are in the subject’s native space.

The volume is sampled at the vertex positions using trilinear interpolation (default) or nearest neighbor interpolation. Vertices which fall outside the volume (e.g., medial wall vertices at the bottom of a brainmask volume, or vertices outside a truncated field of view) get the value NA by default, and a warning is emitted. Use `clamp = TRUE` to instead use the value of the closest voxel at the volume border, and `check_fov = FALSE` to suppress the check.

To project a group-level statistical map (e.g., in MNI space) onto a template like `fsaverage`, see [template.vol2surf](#).

Usage

```
subject.vol2surf(
  subjects_dir,
  subject_id,
  volume,
  surface = "white",
  hemi = "both",
  surface_frac = NULL,
  frac_surface = "pial",
  interpolation = "trilinear",
  frame = 1L,
  cortex_only = FALSE,
  clamp = FALSE,
  check_fov = TRUE,
  affine = NULL,
  vox2ras = "auto"
)
```

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 or named list. A volume filepath, the name of a volume file in the <code>mri</code> directory of the subject without file extension (e.g., <code>brain</code> or <code>aseg</code>), or an in-memory volume given as a named list with entries <code>'data'</code> (numeric 3D or 4D array) and <code>'affine'</code> (4x4 numeric matrix mapping 0-based voxel indices to world coordinates). Examples for the latter are read.fs.mgh (use <code>with_header = TRUE</code> and pass <code>list(data = vol\$data, affine = vol\$header\$vox2ras_matrix)</code>) and <code>nibabel</code> in Python.
surface	character string, the name of the surface to sample. Examples: <code>'white'</code> , <code>'pial'</code> , <code>'midthickness'</code> . The corresponding <code>surf/?h.<surface></code> files must exist for the subject.
hemi	character string, one of <code>'lh'</code> , <code>'rh'</code> or <code>'both'</code> . The hemisphere to project to.
surface_frac	numeric scalar in range 0..1, or <code>NULL</code> (the default). If given, the volume is not sampled at the vertices of surface, but at the vertices of the surface which lies at the given fraction on the line between the corresponding vertices of surface (fraction 0) and <code>frac_surface</code> (fraction 1). E.g., with <code>surface = "white"</code> , <code>frac_surface = "pial"</code> and <code>surface_frac = 0.5</code> , the mid-cortical surface is used. This is what the <code>--surf-frac</code> option of <code>mri_vol2surf</code> does. Note that no surface normals are required, the interpolation is purely geometric.
frac_surface	character string, the second surface for <code>surface_frac</code> . Ignored if <code>surface_frac</code> is <code>NULL</code> . Defaults to <code>'pial'</code> .
interpolation	character string, one of <code>'trilinear'</code> (the default, an alias <code>'linear'</code> is accepted) or <code>'nearest'</code> . Trilinear interpolation is suitable for continuous data like thickness or t-statistics, while nearest neighbor is what you want for discrete data like atlas labels, segmentation indices or p-values (it never averages the values of neighboring voxels).
frame	positive integer scalar, the index of the volume (<code>'frame'</code>) to use if volume contains more than one. Defaults to 1. Ignored for 3D volumes.
cortex_only	logical, whether to set the values of all vertices which are <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) to NA. This masks the medial wall. Defaults to <code>FALSE</code> .
clamp	logical. How to handle surface vertices which are outside the volume: if <code>FALSE</code> (the default), their value is set to NA; if <code>TRUE</code> , the value of the closest voxel at the volume border is used instead (edge replication, like the <code>mode = 'nearest'</code> parameter of <code>scipy.ndimage.map_coordinates</code>). See also <code>check_fov</code> .
check_fov	logical, whether to check for surface vertices which are outside the volume and emit a warning listing them. Defaults to <code>TRUE</code> . The check is very cheap (a few comparisons per vertex), the parameter exists mainly so that you can silence the warning when you know that a large part of the surface is outside the volume on purpose.

affine	numeric 4x4 matrix or NULL (the default). The matrix that maps 0-based voxel indices to world coordinates, i.e., the coordinate space in which the surface vertices are defined. By default this is derived from the volume (see vox2ras). Use this parameter only if you want to override the transformation completely.
vox2ras	character string, how the transformation from voxel indices to the coordinate space of the surface is obtained. One of 'auto' (the default), 'tkr' or 'header'. For MGH/MGZ volumes, 'auto' uses the FreeSurfer <i>tkregister</i> convention, in which the brain surfaces are defined: the direction cosines and voxel sizes from the volume header, but with the origin moved to the center of the volume (see vox2ras_tkr and vol.tkreg.affine). This matches what the FreeSurfer tools do (it is the difference between the --vox2ras-tkr and the --vox2ras option of mri_vol2surf), and it matters for volumes whose header carries a non-zero center-of-RAS offset (cras) – such volumes are common, e.g., when the data was imported from DICOM. For NIfTI volumes, 'auto' uses the transformation matrix from the file header (sform/qform), as there is no tkregister convention for NIfTI files. Use 'header' or 'tkr' to force one of the two interpretations. Ignored if affine is given.

Value

a numerical vector of per-vertex values, one value per surface vertex, or a hemilist (named list with entries lh and rh) of such vectors if hemi is 'both'. Note that the values are in the order of the vertices of the *sampled* surface, which is the order used by the surface file, so they can be used directly with functions like [vis.subject.morph.native](#).

Note

Trilinear interpolation is used by default, while the FreeSurfer tool mri_vol2surf defaults to nearest neighbor interpolation: pass interpolation = "nearest" to reproduce its output exactly. Note also that the affine matrix of a NIfTI file is converted into an MGH-style header by the reader, which stores the direction cosines with limited precision: for volumes with an oblique (rotated) orientation the transformation can differ from the value in the file by about 1e-07, and sheared transformations are not supported (the reader warns about this).

See Also

Other volume to surface projection functions: [template.vol2surf\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
# Project the brain volume of subject1 onto the white surface:
brain_morph = subject.vol2surf(subjects_dir, "subject1", volume = "brain", surface = "white");
# Project the same volume onto the mid-cortical surface (between white and pial):
mid_morph = subject.vol2surf(subjects_dir, "subject1", volume = "brain",
  surface = "white", frac_surface = "pial", surface_frac = 0.5);
# Visualize the result on the surface:
vis.subject.morph.native(subjects_dir, "subject1", morph_data = brain_morph);
```

```
## End(Not run)
```

subject.volume	<i>Read a brain volume.</i>
----------------	-----------------------------

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

<code>subjects_dir</code>	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.
<code>subject_id</code>	character string, the subject identifier.
<code>volume</code>	character string, name of the volume file without file extension. Examples: <code>brain</code> or <code>aseg</code> .
<code>format</code>	string. One of <code>'mgh'</code> , <code>'mgz'</code> , <code>'AUTO'</code> . If left at the default value <code>'AUTO'</code> , the function will look for files with extensions <code>'mgh'</code> and <code>'mgz'</code> (in that order) and use the first one that exists.
<code>drop_empty_dims</code>	logical, whether to drop empty dimensions of the returned data. Passed to read.fs.mgh .
<code>with_header</code>	logical. Whether to return the header as well. If <code>TRUE</code> , return a named list with entries <code>"data"</code> and <code>"header"</code> . The latter is another named list which contains the header data. These header entries exist: <code>"dtype"</code> : int, one of: 0= <code>MRI_UCHAR</code> ; 1= <code>MRI_INT</code> ; 3= <code>MRI_FLOAT</code> ; 4= <code>MRI_SHORT</code> . <code>"voldim"</code> : integer vector. The volume (=data) dimensions. E.g., <code>c(256, 256, 256, 1)</code> . These header entries may exist: <code>"vox2ras_matrix"</code> (exists if <code>"ras_good_flag"</code> is 1), <code>"mr_params"</code> (exists if <code>"has_mr_params"</code> is 1). Passed to read.fs.mgh .
<code>mri_subdir</code>	character string or <code>NULL</code> , the subdir to use within the <code>mri</code> directory. Defaults to <code>NULL</code> , which means to read directly from the <code>mri</code> dir. You could use this to read volumes from the <code>mri/orig/</code> directory by setting it to <code>orig</code> .

Value

numerical array, the voxel data. If with_header, the full volume datastructure (see above).

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain', with_header = TRUE);
# Use the vox2ras matrix from the header to compute RAS coordinates at CRS voxel (0, 0, 0):
brain$header$vox2ras_matrix %*% c(0,0,0,1);

## End(Not run)
```

surface.curvatures	<i>Compute the k1 and k2 principal curvatures of a mesh.</i>
--------------------	--

Description

Compute the k1 and k2 principal curvatures of a mesh.

Usage

```
surface.curvatures(surface)
```

Arguments

surface an fs.surface instance, as returned by subject.surface.

Value

named list, the entries 'K1' and 'K2' contain the principal curvatures.

Note

Require the optional dependency package 'Rvcg'.

template.vol2surf	<i>Project a volume onto a template surface like fsaverage or fs_LR_32.</i>
-------------------	---

Description

Samples a volume at the positions of the vertices of a template surface, e.g., for group-level statistical maps. The volume and the template surface must be defined in the same coordinate space: fsaverage is the FreeSurfer template and its surfaces are in MNI305 (surface RAS) space, while the fs_LR_32 template (the HCP-style 32k surface space) is in MNI152 space. Projecting an MNI152 volume onto fsaverage (or vice versa) will produce wrong results, but this function will usually detect it and warn (see check_fov).

This is the equivalent of the project_vol2surf function of the Python package yabplot for the fs_LR_32 template, and of mri_vol2surf for fsaverage. See [subject.vol2surf](#) for the per-subject variant, which uses the native space of a subject.

Usage

```
template.vol2surf(
    volume,
    template = "fsaverage",
    surface = NULL,
    hemi = "both",
    subjects_dir = NULL,
    surface_frac = NULL,
    frac_surface = "pial",
    interpolation = "trilinear",
    frame = 1L,
    cortex_only = FALSE,
    clamp = FALSE,
    check_fov = TRUE,
    affine = NULL,
    vox2ras = "auto"
)
```

Arguments

volume	character string or named list, see subject.vol2surf . For template projections, this is typically the filepath of a NIfTI file containing a group-level map, e.g., in MNI152 space.
template	character string, the template identifier. One of 'fsaverage' (the FreeSurfer template, in MNI305 space) or 'fs_LR_32' (the HCP-style 32k template, in MNI152 space). Additional templates can be used by passing the respective subjects_dir and a template identifier that matches a directory in it.
surface	character string, the name of the surface to sample. Examples: 'white', 'pial', 'midthickness', 'inflated'. Defaults to 'white' for 'fsaverage' and to 'midthickness' for 'fs_LR_32' if surface is NULL.

hemi	character string, see subject.vol2surf .
subjects_dir	character string or NULL, the subjects dir which contains the template subject. If NULL (the default), the fsbrain data cache (see get_optional_data_filepath) is searched first, then the FreeSurfer installation. If the required files are missing, the respective download function is suggested in the error message.
surface_frac	numeric scalar or NULL, see subject.vol2surf .
frac_surface	character string, see subject.vol2surf .
interpolation	character string, see subject.vol2surf .
frame	positive integer scalar, see subject.vol2surf .
cortex_only	logical, see subject.vol2surf . For 'fsaverage', the required label/?h.cortex.label file is part of the FreeSurfer installation, for 'fs_LR_32' it can be downloaded with download_fs_LR_32_labels .
clamp	logical, see subject.vol2surf .
check_fov	logical, see subject.vol2surf .
affine	numeric 4x4 matrix or NULL, see subject.vol2surf .
vox2ras	character string, see subject.vol2surf .

Value

a numerical vector of per-vertex values, or a hemilist of such vectors if hemi is 'both'. See [subject.vol2surf](#).

Note

The fs_LR_32 template meshes can be downloaded with [download_fs_LR_32_meshes](#). Note that the surface files of the templates are not part of the fsbrain package, and that the fsaverage data is subject to the FreeSurfer license.

See Also

Other volume to surface projection functions: [subject.vol2surf\(\)](#)

Examples

```
## Not run:
# Project a group-level stat map (e.g., in MNI152 space) onto the HCP-style fs_LR_32 template:
fsbrain::download_fs_LR_32_meshes();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
stat_map = template.vol2surf("/path/to/your_statmap.nii.gz", template = "fs_LR_32",
  surface = "midthickness", subjects_dir = subjects_dir);
# Project an MNI305 map onto fsaverage:
stat_map = template.vol2surf("/path/to/your_mni305_map.nii.gz", template = "fsaverage",
  surface = "white");

## End(Not run)
```

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

Description

Get an fs.surface brain mesh from an rgl tmesh3d instance.

Usage

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

Arguments

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

Value

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

`Triangles3D.to.coloredmesh`

Convert a misc3d Triangles3D iso-surface to a coloredmesh.

Description

Convert a `misc3d::contour3d(draw = FALSE)` result (an iso-surface mesh of class 'Triangles3D', e.g., as returned by [volvis.contour](#)) into an `fs.coloredmesh`. This allows rendering volume iso-surfaces with ANY renderer backend: while the rgl backend can render 'Triangles3D' instances directly, the scimesh backend requires `fs.coloredmesh` instances and converts them automatically, so you normally do not need to call this function yourself.

Usage

```
Triangles3D.to.coloredmesh(tris, hemi = NULL, add_normals = TRUE)
```

Arguments

`tris` a 'Triangles3D' instance as returned by `misc3d::contour3d(draw = FALSE)` or [volvis.contour](#), or a list of such instances (e.g., when multiple frames of a 4D volume were extracted with `frame = "all"`).

`hemi` character string or NULL, the hemisphere for the resulting coloredmesh. 'Triangles3D' iso-surfaces are not hemisphere-specific, so this defaults to NULL. See [fs.coloredmesh](#).

`add_normals` logical, whether to compute per-vertex normals for the mesh. Required for correct lighting in the scimesh backend, defaults to TRUE.

Value

an fs.coloredmesh instance, or a list of such instances if 'tris' is a list.

See Also

Other coloredmesh functions: [coloredmesh.from.annot\(\)](#), [coloredmesh.from.color\(\)](#), [coloredmesh.from.label\(\)](#), [coloredmesh.from.mask\(\)](#), [coloredmesh.from.morph.native\(\)](#), [coloredmesh.from.morph.standard\(\)](#), [coloredmesh.from.morphdata\(\)](#), [coloredmeshes.from.color\(\)](#)

vdata.split.by.hemi *Split morph data vector at hemisphere boundary.*

Description

Given a single vector of per-vertex data for a mesh, split it at the hemi boundary. This is achieved by loading the respective surface and checking the number of vertices for the 2 hemispheres.

Usage

```
vdata.split.by.hemi(
    subjects_dir,
    subject_id,
    vdata,
    surface = "white",
    expand = TRUE
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
vdata	numerical vector of data for both hemispheres, one value per vertex
surface	the surface to load to determine the vertex counts
expand	logical, whether to allow input of length 1, and expand (repeat) it to the length of the hemispheres.

Value

a hemilist, each entry contains the data part of the respective hemisphere.

Note

Instead of calling this function to split the data, you could use the 'split_by_hemi' parameter of [subject.morph.native](#).

vertex.coords	<i>Return coordinates for vertices, supporting entire brain via hemilist.</i>
---------------	---

Description

Return coordinates for vertices, supporting entire brain via hemilist.

Usage

```
vertex.coords(surface, vertices)
```

Arguments

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

Value

double nx3 matrix of vertex coordinates.

See Also

Other 3d utility functions: [highlight.points.spheres\(\)](#), [highlight.vertices.spheres\(\)](#)

vertex.hemis	<i>Return the proper hemi string ('lh' or 'rh') for each vertex.</i>
--------------	--

Description

Return the proper hemi string ('lh' or 'rh') for each vertex.

Usage

```
vertex.hemis(surface, vertices)
```

Arguments

surface	hemilist of surfaces or a single integer which will be interpreted as the vertex count of the left hemisphere.
vertices	vector of positive integers, the query vertex indices. Can be in range 1..(nv(lh)+nv(rh)), i.e., across the whole brain.

Value

vector of character strings, each string is 'lh' or 'rh'.

Note

It is not checked in any way whether the vertex indices are out of bounds on the upper side (higher than the highest rh vertex index).

Examples

```
vertex.hemis(100L, vertices=c(99L, 100L, 101L));
```

vis.color.on.subject	<i>Visualize pre-defined vertex colors on a subject.</i>
----------------------	--

Description

Visualize pre-defined vertex colors on a subject.

Usage

```
vis.color.on.subject(
  subjects_dir,
  vis_subject_id,
  color_lh = NULL,
  color_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  color_both = NULL,
  style = "default"
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
color_lh	vector of colors to visualize on the left hemisphere surface. Length must match number of vertices in hemi surface, or be a single color.
color_rh	vector of colors to visualize on the right hemisphere surface. Length must match number of vertices in hemi surface, or be a single color.

surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
color_both	vector of colors to visualize on the left and right hemispheres. Alternative to 'color_lh' and 'color_rh'. Length must match sum of vertices in both hemis. Can also be a hemilist.
style	character string or rgl rendering style, see get.rglstyle .

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other surface visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
color_lh = '#ff0000';
num_verts_subject1_rh = 153333;
color_rh = rep('#333333', num_verts_subject1_rh);
color_rh[1:30000] = '#00ff00';
color_rh[30001:60000] = '#ff0000';
color_rh[60001:90000] = '#0000ff';
color_rh[90001:120000] = '#ffff00';
color_rh[120001:150000] = '#00ffff';
vis.color.on.subject(subjects_dir, 'subject1', color_lh, color_rh);

## End(Not run)
```

vis.coloredmeshes	<i>Visualize a list of colored meshes in a single scene.</i>
-------------------	--

Description

Visualize a list of colored meshes in a single scene.

Usage

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

Arguments

coloredmeshes	list of coloredmesh. A coloredmesh is a named list as returned by the coloredmesh.from.* functions. It has the entries 'mesh' of type tmesh3d, a 'col', which is a color specification for such a mesh.
background	string, background color passed to rgl::bg3d()
skip_all_na	logical, whether to skip (i.e., not render) meshes in the list that have the property 'render' set to FALSE. Defaults to TRUE. Practically, this means that a hemisphere for which the data was not given is not rendered, instead of being rendered in a single color.
style	a named list of style parameters or a string specifying an available style by name (e.g., 'shiny'). Defaults to 'default', the default style. Use the magic word 'from_mesh' to use the 'style' field of each coloredmesh instead of a single, global style: this allows you to give individual meshes in the scene their own look, e.g., to render a cortex mesh semi-transparently behind colored data meshes. Meshes which have no 'style' field fall back to the 'default' style. See vis.subcortical.region.values for an example, and note that the scimesh renderer backend currently only supports the alpha channel of a style, not other material properties.
rgloptions	option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000));
rglactions	named list. A list in which the names are from a set of pre-defined actions. Defaults to the empty list.
draw_colorbar	logical. Whether to draw a colorbar. WARNING: Will only show up if there is enough space in the plot area and does not resize properly. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.

Value

the list of visualized coloredmeshes

Note

To change or adapt the colorbar, you should use the `makecmap_options` parameter when constructing them in a `vis` function. See the example.

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
cm = vis.subject.morph.native(subjects_dir, 'subject1', 'thickness',
  makecmap_options=list('n'=100, 'colFn'=viridis::viridis));
# You could mess with the meshes here.
vis.coloredmeshes(cm);

## End(Not run)
```

vis.coloredmeshes.rotating

Visualize a list of colored meshes in a single scene and rotate them, movie-style.

Description

Visualize a list of colored meshes in a single scene and rotate them, movie-style.

Usage

```
vis.coloredmeshes.rotating(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  x = 0,
  y = 0,
  z = 1,
  rpm = 6,
  duration = 10,
  rgloptions = rglo(),
  rglactions = list()
)
```

Arguments

coloredmeshes	list of coloredmesh. A coloredmesh is a named list as returned by the coloredmesh.from.* functions. It has the entries 'mesh' of type tmesh3d, a 'col', which is a color specification for such a mesh.
background	string, background color passed to rgl::bg3d()
skip_all_na	logical, whether to skip (i.e., not render) meshes in the list that have the property 'rendner' set to FALSE. Defaults to TRUE. Practically, this means that a hemisphere for which the data was not given is not rendered, instead of being rendered in a single color.
style	a named list of style parameters or a string specifying an available style by name (e.g., 'shiny'). Defaults to 'default', the default style.
x	rotation x axis value, passed to spin3d. Defaults to 0.
y	rotation y axis value, passed to spin3d. Defaults to 1.
z	rotation z axis value, passed to spin3d. Defaults to 0.
rpm	rotation rpm value, passed to spin3d. Defaults to 15.
duration	rotation duration value, passed to spin3d. Defaults to 20.
rgloptions	option list passed to par3d. Example: rgloptions = list("windowRect"=c(50,50,1000,1000));
rglactions	named list. A list in which the names are from a set of pre-defined actions. Defaults to the empty list.

Value

the list of visualized coloredmeshes

vis.colortable.legend *Create a separate legend plot for a colortable or an annotation.*

Description

This plots a legend for a colortable or an atlas (annotation), showing the region names and their assigned colors. This function creates a new plot.

Usage

```
vis.colortable.legend(colortable, ncols = 1L, plot_struct_index = TRUE)
```

Arguments

colortable	dataframe, a colortable as returned by read.fs.colortable or the inner 'colortable_df' returned by subject.annot. One can also pass an annotation (fs.annot instance).
ncols	positive integer, the number of columns to use when plotting
plot_struct_index	logical, whether to plot the region index from tge 'struct_index' field. If there is no such field, this is silently ignored.

Note

This function plots one or more legends (see [legend](#)). You may have to adapt the device size before calling this function if you intend to plot a large colortable.

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
annot = subject.annot(subjects_dir, 'subject1', 'lh', 'aparc');
vis.colortable.legend(annot$colortable_df, ncols=3);

## End(Not run)
```

vis.connectome

Visualize a connectivity matrix as a connectome on a brain surface.

Description

Draws the edges of a connectivity matrix as lines between region centroids, the nodes as spheres, and (optionally) a semi-transparent brain surface as context. This is the fsbrain equivalent of the `plot_connectome` function of the Python package `yabplot`. It can be used for any brain connectome, e.g., structural connectivity (tractography), functional connectivity (correlation), or any other square matrix that describes a relation between brain regions.

Note that this is a *visualization* function: it does not compute a connectivity matrix, and it does not check whether the data is a valid connectivity estimate.

Usage

```
vis.connectome(
  connectivity_matrix,
  node_coords = NULL,
  node_names = NULL,
  subjects_dir = NULL,
  template_id = "fs_LR_32",
  atlas = NULL,
  context = list(surface = "midthickness", alpha = 0.08, color = "#B0B0B0"),
  edge_threshold = NULL,
  edge_threshold_quantile = 0.95,
  edge_absolute = TRUE,
  edge_negative = FALSE,
  edge_width = 1,
  edge_width_range = c(0.5, 3),
  edge_scale = c("weight", "none"),
  edge_color = NULL,
  edge_makecmap_options = modifyList(mkco.div(), list(symm = FALSE)),
```

```

node_radius = 2,
node_radius_range = NULL,
node_scale = c("none", "strength"),
node_color = NULL,
node_makecmap_options = mkco.seq(),
views = c("sd_lateral_lh", "sd_medial_lh", "sd_lateral_rh", "sd_medial_rh"),
rgloptions = rglo(),
rglactions = list(),
style = "default",
draw_colorbar = FALSE,
silent = FALSE
)

```

Arguments

connectivity_matrix	numeric square matrix, the connectivity matrix. The rows and columns are the nodes (brain regions). The names of the rows/columns (or the parameter node_names) define the nodes, see the parameter node_coords. Alternatively, an edge list as a data.frame with the columns 'source', 'target' and 'weight'.
node_coords	(n, 3) numeric matrix, the coordinates of the nodes (brain regions), in the same order as the matrix rows. If NULL, the nodes are looked up by name in the atlas given in parameter atlas, see the parameter subjects_dir and the function subject.region.centroids . The row names of the matrix (or the parameter node_names) must then match the atlas region names.
node_names	vector of character strings, the node names. Only used if node_coords is given and the matrix has no row names. Defaults to NULL.
subjects_dir	character string or NULL. The FreeSurfer SUBJECTS_DIR, see subject.region.centroids . Defaults to NULL, which searches the standard locations for the template subject template_id.
template_id	character string, the identifier of the template (or subject) whose atlas and surface define the nodes and the context surface. Defaults to 'fs_LR_32'. Ignored if node_coords is given <i>and</i> context is NULL.
atlas	character string or NULL, the atlas whose regions are the nodes, e.g., 'schaefer400'. Not required if node_coords is given.
context	named list or NULL, the semi-transparent brain surface drawn behind the connectome. The list can have the entries: 'surface' (character string, the surface to use, defaults to 'midthickness'), 'alpha' (numeric, the transparency of the surface, defaults to 0.08), and 'color' (character string, the color of the surface, defaults to '#B0B0B0'). Set to NULL to draw the connectome without any brain surface, which is much faster and gives a 'spring layout' style figure (the camera then frames the nodes).
edge_threshold	numeric or NULL, an absolute threshold on the edge weight. Edges with a lower magnitude are not drawn. Defaults to NULL (no absolute threshold).
edge_threshold_quantile	numeric or NULL, a quantile (in the range 0 to 1) of the edge magnitudes. Only edges with a magnitude above that quantile are drawn, i.e., the quantile selects

	the strongest edges. Defaults to 0.95, i.e., the strongest 5 percent of the edges are drawn. Set to NULL to use all edges.
edge_absolute	logical, whether the magnitude of the edge weights is used for thresholding and for the line widths. Defaults to TRUE, i.e., a strong negative edge is drawn like a strong positive one (but see the parameter <code>edge_negative</code>).
edge_negative	logical, whether to draw the negative edges instead of the positive ones. Defaults to FALSE, i.e., only positive edges are considered. To show both the positive and the negative edges of a signed matrix in one figure, call this function twice and combine the results (the second call can be rendered on top of the first).
edge_width	positive number, the line width in pixels. Defaults to 1.0 (a thin line, which is what hardware line rendering supports best).
edge_width_range	numeric vector of length 2 or NULL, the range of the line widths used when <code>edge_scale</code> is 'weight'. Defaults to <code>c(0.5, 3.0)</code> .
edge_scale	character string, one of 'weight' (the line width encodes the edge weight) or 'none' (all edges have the width <code>edge_width</code>). Defaults to 'weight'.
edge_color	vector of hex color strings or NULL. A single color for all edges, or one color per edge. If NULL (the default), the edges are colored by their weight, using the colormap defined by <code>edge_makecmap_options</code> .
edge_makecmap_options	named list of parameters to pass to <code>makecmap</code> , see <code>mkco.div</code> . Used to color the edges by weight and to draw the colorbar. Defaults to a diverging colormap which spans the weights of the drawn edges (note the 'symm' setting: only one sign of edges is drawn per call, so a symmetric range would waste half of the colormap).
node_radius	positive number, the radius of the node spheres, in the units of the surface coordinates (millimeters). Defaults to 2.0.
node_radius_range	numeric vector of length 2 or NULL, the range of the node radii used when <code>node_scale</code> is 'strength'. Defaults to NULL, which uses 0.5 and 1.5 times <code>node_radius</code> .
node_scale	character string, one of 'none' (all nodes have the radius <code>node_radius</code>) or 'strength' (the node radius encodes the node strength, i.e., the sum of the absolute weights of all its edges). Defaults to 'none'.
node_color	vector of hex color strings or NULL. A single color for all nodes, or one color per node. If NULL (the default), the nodes are colored by their strength (see <code>node_scale</code>), using the colormap defined by <code>node_makecmap_options</code> .
node_makecmap_options	named list of parameters to pass to <code>makecmap</code> , see <code>mkco.seq</code> . Used to color the nodes by strength and to draw the colorbar. Defaults to <code>mkco.seq()</code> .
views	vector of character strings, the views to render, see <code>get.view.angle.names</code> . Defaults to the 4 standard views.
rgloptions	option list passed to <code>par3d</code> , see <code>rglo</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions, see <code>rglactions</code> .

style	a rendering style for the surface meshes, see get.rglstyle . Note that the style of the context surface and of the nodes is not affected by this, only the style requested for the <i>edges</i> (which is of little relevance, as lines are drawn unlit by default).
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar for the <i>edges</i> . Defaults to FALSE. Note that a colorbar is only drawn for the edges, not for the nodes: if you want to draw a colorbar for the node strengths, use coloredmesh.plot.colorbar.separate on the nodes (the entry 'nodes' of the return value). Also note that the headless scimesh renderer backend does not support colorbars in this function, use export instead (see the return value).
silent	logical, whether to suppress the status messages. Defaults to FALSE.

Value

named list of renderables, invisibly: the entries 'edges' (an `fs.coloredpaths` instance), 'nodes' (an `fs.coloredmesh` instance) and 'context_lh'/'context_rh' (the context surface, if requested). This list can be passed to [export](#) to create a publication quality figure with a colorbar, which is also the recommended way to use the scimesh renderer backend.

See Also

Other connectome functions: [subject.region.centroids\(\)](#)

Examples

```
## Not run:
# Use the fs_LR_32 template (see subject.region.centroids for the download commands).
centroids = subject.region.centroids(subjects_dir = NULL, subject_id = "fs_LR_32",
  atlas = "schaefer400", surface = "midthickness");
set.seed(42);
mat = matrix(runif(nrow(centroids)^2), nrow = nrow(centroids),
  dimnames = list(rownames(centroids), rownames(centroids)));
mat = (mat + t(mat)) / 2;
diag(mat) = 0;

# Render the connectome in 3 views:
cm = vis.connectome(mat, atlas = "schaefer400",
  views = c("sd_lateral_lh", "sd_dorsal", "sd_caudal"));

# Or create a publication quality image with a colorbar for the edges:
export(cm, draw_colorbar = "horizontal",
  output_img = "connectome.png", colorbar_legend = "Edge weight");

## End(Not run)
```

vis.data.on.fsaverage *Visualize arbitrary data on the fsaverage template subject, if available.*

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain. This function tries to automatically retrieve the subjects dir of the fsaverage template subject by checking the environment variables SUBJECTS_DIR and FREESURFER_HOME for the subject. The subject is required for its surfaces, which are not shipped with this package for licensing reasons.

Usage

```
vis.data.on.fsaverage(
    subjects_dir = NULL,
    vis_subject_id = "fsaverage",
    morph_data_lh = NULL,
    morph_data_rh = NULL,
    surface = "white",
    views = c("t4"),
    rgloptions = rglo(),
    rglactions = list(),
    draw_colorbar = FALSE,
    makecmap_options = mkco.seq(),
    bg = NULL,
    morph_data_both = NULL,
    style = "default"
)
```

Arguments

- | | |
|----------------|---|
| subjects_dir | string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier. |
| vis_subject_id | string. The subject identifier from which to obtain the surface for data visualization. Defaults to 'fsaverage'. |
| morph_data_lh | numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL. |
| morph_data_rh | numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has |

	vertices. If NULL, this surface will not be rendered. Only one of <code>morph_data_lh</code> or <code>morph_data_rh</code> is allowed to be NULL.
<code>surface</code>	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
<code>views</code>	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.
<code>rgloptions</code>	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
<code>rglactions</code>	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
<code>draw_colorbar</code>	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
<code>makecmap_options</code>	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'.
<code>bg</code>	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
<code>morph_data_both</code>	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, 'morph_data_lh' and 'morph_data_rh' must be NULL.
<code>style</code>	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.subject\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

vis.data.on.group.native

Visualize native space data on a group of subjects.

Description

Plot surface data on the native space surfaces of a group of subjects and combine the tiles into a single large image.

Usage

```
vis.data.on.group.native(
    subjects_dir,
    subject_id,
    morph_data_both,
    view_angles = "sd_dorsal",
    output_img = "fsbrain_group_morph.png",
    num_per_row = 5L,
    captions = subject_id,
    rglactions = list(no_vis = TRUE),
    ...
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	vector of character strings, the subject identifiers
morph_data_both	named list of numerical vectors, the morph data for both hemispheres of all subjects. Can be loaded with group.morph.native .
view_angles	see get.view.angle.names .
output_img	character string, the file path for the output image. Should end with '.png'.
num_per_row	positive integer, the number of tiles per row.
captions	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png" = "~/fsbrain.png", "clip_data" = c(0.05, 0.95))</code> . See rglactions .
...	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: <code>surface='pial'</code> .

Value

named list, see the return value of `arrange.brainview.images.grid` for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'morph_data_both' parameter. The surfaces are loaded in the order of the 'subject_id' parameter, so the order in both must match.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi
'range'=c(0, 4))`.

See Also

Other group visualization functions: `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.coloredmeshes`
`vis.group.morph.native()`, `vis.group.morph.standard()`

`vis.data.on.group.standard`

Visualize standard space data for a group on template.

Description

Plot standard space data for a group of subjects onto a template brain and combine the tiles into a single large image.

Usage

```
vis.data.on.group.standard(  
  subjects_dir,  
  vis_subject_id,  
  morph_data_both,  
  captions = NULL,  
  view_angles = "sd_dorsal",  
  output_img = "fsbrain_group_morph.png",  
  num_per_row = 5L,  
  rglactions = list(no_vis = TRUE),  
  ...  
)
```

Arguments

`subjects_dir` character string, the path to the SUBJECTS_DIR containing the template subject
`vis_subject_id` character string, the template subject name. A typical choice is 'fsaverage'.

morph_data_both	named list of numerical vectors, 4D array or dataframe, the morph data for both hemispheres of all subjects. Can be loaded with group.morph.standard or group.morph.standard.sf .
captions	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
view_angles	see get.view.angle.names .
output_img	character string, the file path for the output image. Should end with '.png'.
num_per_row	positive integer, the number of tiles per row.
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
...	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: <code>surface='pial'</code> .

Value

named list, see the return value of [arrange.brainview.images.grid](#) for details.

Note

The subject data are plotted row-wise, in the order in which they appear in the 'morph_data_both' parameter.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi, 'range'=c(0, 4))`.

See Also

Other group visualization functions: [vis.data.on.group.native\(\)](#), [vis.group.annot\(\)](#), [vis.group.coloredmeshes\(\)](#), [vis.group.morph.native\(\)](#), [vis.group.morph.standard\(\)](#)

vis.data.on.subject	<i>Visualize arbitrary data on the surface of any subject.</i>
---------------------	--

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.data.on.subject(
  subjects_dir,
  vis_subject_id,
  morph_data_lh = NULL,
  morph_data_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = mkco.seq(),
  bg = NULL,
  morph_data_both = NULL,
  style = "default"
)
```

Arguments

- | | |
|----------------|--|
| subjects_dir | string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier. |
| vis_subject_id | string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'. |
| morph_data_lh | numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL. |
| morph_data_rh | numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL. |
| surface | string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white". |
| views | list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles. |
| rgloptions | option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)). |
| rglactions | named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95)). See rglactions . |

<code>draw_colorbar</code>	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
<code>makecmap_options</code>	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'.
<code>bg</code>	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
<code>morph_data_both</code>	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, 'morph_data_lh' and 'morph_data_rh' must be NULL.
<code>style</code>	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
morph_data_rh = NULL;
vis.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);

## End(Not run)
```

vis.dti.trk

*Visualize DTI tracks from Diffusion Toolkit/TrackVis TRK format file.***Description**

Visualize DTI tracks from Diffusion Toolkit/TrackVis TRK format file.

Usage

```
vis.dti.trk(
    trk,
    filter_tracks = list(min_length = 15, min_segment_count = 6),
    color_by_orientation = FALSE
)
```

Arguments

trk	character string, the path to a TRK file that should be loaded. Alternatively, a loaded trk instance as returned by <code>freesurferformats::read.dti.trk</code> .
filter_tracks	optional, named list of filters. Can contain fields <code>min_length</code> and <code>min_segment_count</code> . Set the whole thing to <code>NULL</code> or an entry to 0 for no filtering.
color_by_orientation	logical, whether to color the tracks by orientation. Slower, but may make the resulting visualization easier to interpret.

Value

The (loaded or received) trk instance. Note that this function is typically called for the side effect of visualization.

Note

The current simple implementation is very slow if the number of tracks becomes large (several thousand tracks).

Examples

```
## Not run:
# Create the following file with Diffusion Toolkit from your DTI data:
trk = freesurferformats::read.dti.trk("~/data/tim_only/tim/DICOM/dti.trk");
vis.dti.trk(trk);

## End(Not run)
```

vis.export.from.coloredmeshes

Export high-quality brainview image with a colorbar.

Description

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

Usage

```
vis.export.from.coloredmeshes(
    coloredmeshes,
    colorbar_legend = NULL,
    img_only = TRUE,
    horizontal = TRUE,
    silent = TRUE,
    quality = 1L,
    output_img = "fsbrain_arranged.png",
    image.plot_extra_options = NULL,
    large_legend = TRUE,
    view_angles = get.view.angle.names(angle_set = "t4"),
    style = "default",
    grid_like = TRUE,
    background_color = "white",
    transparency_color = NULL,
    ...
)
```

Arguments

<code>coloredmeshes</code>	list of coloredmesh. A coloredmesh is a named list as returned by the <code>coloredmesh.from*</code> functions (like <code>coloredmesh.from.morph.native</code>). It has the entries 'mesh' of type <code>tmesh3d</code> , a 'col', which is a color specification for such a mesh. The <code>vis*</code> functions (like <code>vis.subject.morph.native</code>) all return a list of coloredmeshes.
<code>colorbar_legend</code>	character string or NULL, the title for the colorbar.
<code>img_only</code>	logical, whether to return only the resulting image
<code>horizontal</code>	logical, whether to plot the colorbar horizontally (TRUE) or vertically (FALSE). Pass 'NULL' to force no colorbar at all.
<code>silent</code>	logical, whether to suppress messages
<code>quality</code>	integer, an arbitrary quality. This is the resolution per tile before trimming, divided by 1000, in pixels. Example: 1L means 1000x1000 pixels per tile before trimming. Currently supported values: 1L . . 2L. Note that the resolution you can get is also limited by your screen resolution.

<code>output_img</code>	string, path to the output file. Defaults to "fsbrain_arranged.png"
<code>image.plot_extra_options</code>	named list, custom options for <code>fields::image.plot</code> . Overwrites those derived from the quality setting. If in doubt, leave this alone.
<code>large_legend</code>	logical, whether to plot extra large legend text, affects the font size of the colorbar legend and the tick labels.
<code>view_angles</code>	list of strings. See get.view.angle.names for all valid strings.
<code>style</code>	the rendering style, see <code>material3d</code> or use a predefined style like 'default' or 'shiny'.
<code>grid_like</code>	logical, passed to <code>vislayout.from.coloredmeshes</code> .
<code>background_color</code>	hex color string (like '#FFFFFF'), the color to use for the background. Ignored if 'transparency_color' is not NULL. To get a transparent background, use 'transparency_color' instead of this parameter. WARNING: Do not use color names (like 'gray'), as their interpretation differs between <code>rgl</code> and <code>image magick</code> !
<code>transparency_color</code>	hex color string (like '#FFFFFF'), the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try '#FFFFFF' (white) or '#000000' (black) if in doubt. WARNING: Do not use color names (like 'gray'), as their interpretation differs between <code>rgl</code> and <code>image magick</code> !
<code>...</code>	extra arguments passed to <code>vislayout.from.coloredmeshes</code> .

Value

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

Note

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

See Also

This function should not be used anymore, it will be deprecated soon. Please use the [export](#) function instead.

Examples

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

```
## End(Not run)
```

vis.fs.surface	Visualize <i>fs.surface</i> mesh
----------------	----------------------------------

Description

Render a mesh. All mesh formats supported by the *freesurferformats* package are supported, including OFF, PLY, OBJ, STL, and many more.

Usage

```
vis.fs.surface(
  fs_surface,
  col = "white",
  per_vertex_data = NULL,
  hemi = "lh",
  makecmap_options = mkco.seq(),
  ...
)
```

Arguments

fs_surface	an fs.surface instance, as returned by function like subject.surface or read.fs.surface . If a character string, it is assumed to be the full path of a surface file, and the respective file is loaded with read.fs.surface . If parameter 'hemi' is 'both', this must be a hemilist. A single <code>rgl::tmesh</code> is also fine.
col	vector of colors, the per-vertex-colors. Defaults to white. Must be a single color or one color per vertex. If parameter 'hemi' is 'both', this must be a hemilist.
per_vertex_data	numerical vector, per-vertex data. If given, takes precedence over 'col'. Used to color the mesh using the colormap options in parameter 'makecmap_options'. If a character string, it is assumed to be the full path of a morphometry data file, and the respective file is loaded with read.fs.morph . If parameter 'hemi' is 'both', this must be a hemilist.
hemi	character string, one of 'lh' or 'rh'. This may be used by visualization functions to decide whether or not to show this mesh in a certain view.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
...	extra parameters to pass to vis.coloredmeshes .

Value

see [vis.coloredmeshes](#)

Note

This function can be used to visualize arbitrary triangular meshes in R. Despite its name, it is not limited to brain surface meshes.

vis.group.annot	<i>Plot atlas annotations for a group of subjects.</i>
-----------------	--

Description

Plot atlas annotations for a group of subjects and combine them into a single large image.

Usage

```
vis.group.annot(
  subjects_dir,
  subject_id,
  atlas,
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_annot.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	vector of character strings, the subject identifiers
atlas	vector of character strings, the atlas names. Example: <code>c('aparc', 'aparc.a2009s')</code>
view_angles	see get.view.angle.names .
output_img	character string, the file path for the output image. Should end with '.png'.
num_per_row	positive integer, the number of tiles per row.
captions	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png" = "~/fsbrain.png", "clip_data" = c(0.05, 0.95))</code> . See rglactions .
...	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: <code>surface='pial'</code> .

Value

named list, see the return value of [arrange.brainview.images.grid](#) for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id' and 'atlas'.

See Also

Other group visualization functions: [vis.data.on.group.native\(\)](#), [vis.data.on.group.standard\(\)](#), [vis.group.coloredmeshes\(\)](#), [vis.group.morph.native\(\)](#), [vis.group.morph.standard\(\)](#)

vis.group.coloredmeshes

Plot coloredmeshes for a group of subjects.

Description

Plot coloredmeshes for a group of subjects into a single image.

Usage

```
vis.group.coloredmeshes(
    coloredmeshes,
    view_angles = "sd_dorsal",
    output_img = "fsbrain_group_annot.png",
    num_per_row = 5L,
    captions = NULL,
    background_color = "white"
)
```

Arguments

coloredmeshes	a list of coloredmeshes lists, each entry in the outer list contains the hemilist of coloredmeshes (left and right hemisphere mesh) for one subject.
view_angles	see get.view.angle.names .
output_img	character string, the file path for the output image. Should end with '.png'.
num_per_row	positive integer, the number of tiles per row.
captions	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
background_color	color for image background (transparency is not supported).

Value

named list, see the return value of [arrange.brainview.images.grid](#) for details.

Note

This is a mid-level function, end users may want to call high-level functions like `vis.group.annot` instead.

See Also

Other group visualization functions: `vis.data.on.group.native()`, `vis.data.on.group.standard()`, `vis.group.annot()`, `vis.group.morph.native()`, `vis.group.morph.standard()`

`vis.group.morph.native`

Plot native space morphometry data for a group of subjects.

Description

Plot native space morphometry data for a group of subjects and combine them into a single large image.

Usage

```
vis.group.morph.native(
    subjects_dir,
    subject_id,
    measure,
    view_angles = "sd_dorsal",
    output_img = "fsbrain_group_morph.png",
    num_per_row = 5L,
    captions = subject_id,
    rglactions = list(no_vis = TRUE),
    ...
)
```

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	vector of character strings, the subject identifiers
<code>measure</code>	vector of character strings, the morphometry measures, e.g., <code>c('thickness', 'area')</code>
<code>view_angles</code>	see <code>get.view.angle.names</code> .
<code>output_img</code>	character string, the file path for the output image. Should end with <code>'png'</code> .
<code>num_per_row</code>	positive integer, the number of tiles per row.
<code>captions</code>	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.

`rglactions` named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: `rglactions = list("snapshot_png" = "~/fsbrain.png", "clip_data" = c(0.05, 0.95))`. See [rglactions](#).

`...` extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: `surface='pial'`.

Value

named list, see the return value of [arrange.brainview.images.grid](#) for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id' and 'measure'.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi, 'range'=c(0, 4))`.

See Also

Other group visualization functions: [vis.data.on.group.native\(\)](#), [vis.data.on.group.standard\(\)](#), [vis.group.annot\(\)](#), [vis.group.coloredmeshes\(\)](#), [vis.group.morph.standard\(\)](#)

`vis.group.morph.standard`

Plot standard space morphometry data for a group of subjects.

Description

Plot standard space morphometry data for a group of subjects and combine them into a single large image.

Usage

```
vis.group.morph.standard(
  subjects_dir,
  subject_id,
  measure,
  fwhm = "10",
  view_angles = "sd_dorsal",
  output_img = "fsbrain_group_morph.png",
  num_per_row = 5L,
  captions = subject_id,
  rglactions = list(no_vis = TRUE),
  ...
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	vector of character strings, the subject identifiers
measure	vector of character strings, the morphometry measures, e.g., c('thickness', 'area')
fwhm	vector of character strings, the smoothing kernel FWHM strings, e.g., c('0', '10', '15')
view_angles	see get.view.angle.names .
output_img	character string, the file path for the output image. Should end with '.png'.
num_per_row	positive integer, the number of tiles per row.
captions	optional vector of character strings, the short text annotations for the individual tiles. Typically used to plot the subject identifier.
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
...	extra parameters passed to the subject level visualization function. Not all may make sense in this context. Example: <code>surface='pial'</code> .

Value

named list, see the return value of [arrange.brainview.images.grid](#) for details.

Note

The subjects are plotted row-wise, in the order in which they appear in the 'subject_id' parameter. This function is vectorized over 'subject_id', 'measure' and 'fwhm'.

You can force an identical plot range for all subjects, so that one color represents identical values across subjects, via 'makecmap_options'. E.g., for the ... parameter, pass `makecmap_options=list('colFn'=viridis::vi
'range'=c(0, 4))`.

See Also

Other group visualization functions: [vis.data.on.group.native\(\)](#), [vis.data.on.group.standard\(\)](#), [vis.group.annot\(\)](#), [vis.group.coloredmeshes\(\)](#), [vis.group.morph.native\(\)](#)

```
vis.labeldata.on.subject
```

Visualize a label on the surface of a subject.

Description

Visualizes a label. Note that a label is just a set of vertices, and that you can use this function to visualize sets of vertices, e.g., to see where on the mesh a certain vertex lies. It may be helpful to visualize the vertex with its neighbors, because otherwise it may be too small to spot. Use the function [mesh.vertex.neighbors](#) to get them. It is advisable to set the view to the interactive 'si' mode and use the 'inflated' surface to identify single vertices.

Usage

```
vis.labeldata.on.subject(
    subjects_dir,
    vis_subject_id,
    lh_labeldata,
    rh_labeldata,
    surface = "white",
    views = c("t4"),
    rgloptions = rglo(),
    rglactions = list(),
    draw_colorbar = FALSE,
    makecmap_options = list(colFn = label.colFn.inv),
    style = "default",
    ...
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
lh_labeldata	integer vector of vertex indices for the left hemisphere
rh_labeldata	integer vector of vertex indices for the right hemisphere
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).

<code>rglactions</code>	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
<code>draw_colorbar</code>	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
<code>makecmap_options</code>	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'.
<code>style</code>	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
<code>...</code>	extra arguments to pass to coloredmesh.from.label .

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.subject.label\(\)](#)

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Examples

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

# Define the data to use:
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
lh_labeldata = c(1000, 1001, 1002); # only the vertices, will be tiny.
subject_id = 'subject1';
surface = 'white'; # Should use 'inflated', but we do not currently
```

```

# ship it for the example subject to reduce download size.

# For the right hemi, extend them to neighborhood for better visibility:
rh_labeldata = c(500, 5000);
rh_surface = subject.surface(subjects_dir, subject_id, surface, 'rh');
rh_labeldata_neighborhood = mesh.vertex.neighbors(rh_surface, rh_labeldata);
vis.labeldata.on.subject(subjects_dir, subject_id, lh_labeldata,
  rh_labeldata_neighborhood$vertices, surface=surface, views=c('si'));

## End(Not run)

```

vis.mask.on.subject *Visualize a vertex mask on the surface of a subject.*

Description

A mask is a logical vector that contains one value per vertex. You can create it manually, or use functions like [mask.from.labeldata.for.hemi](#) to create and modify it. Check the example for this function.

Usage

```

vis.mask.on.subject(
  subjects_dir,
  vis_subject_id,
  mask_lh,
  mask_rh,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  makecmap_options = list(colFn = label.colFn.inv),
  style = "default"
)

```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
mask_lh	logical vector or NULL, the mask to visualize on the left hemisphere surface. Must have the same length as the lh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.

mask_rh	logical vector or NULL, the mask to visualize on the right hemisphere surface. Must have the same length as the rh surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of mask_lh or mask_rh is allowed to be NULL.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95)). See rglactions .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other mask functions: [coloredmesh.from.mask\(\)](#), [mask.from.labeldata.for.hemi\(\)](#)

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Examples

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

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

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

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

## End(Not run)
```

vis.path.along.verts *Draw a 3D line from vertex to vertex*

Description

To get a nice path along the surface, pass the vertex indices along a geodesic path. Note: You can first open an interactive brain view (views='si') with a vis* function like [vis.subject.morph.native](#), then run this function to draw into the active plot.

Usage

```
vis.path.along.verts(
  surface_vertices,
  path_vertex_indices = NULL,
  do_vis = TRUE,
```

```

    color = "#FF0000",
    no_material = FALSE
)

```

Arguments

surface_vertices	float matrix of size (n, 3), the surface vertex coordinates, as returned as part of subject.surface or read.fs.surface , in the member "vertices". Can also be a <code>freesurferformats::fs.surface</code> or <code>rgl::tmesh3d</code> instance, in which case the coordinates are extracted automatically.
path_vertex_indices	vector of vertex indices, the path. You will need to have it computed already. (This function does not compute geodesic paths, see geodesic.path for that. You can use it to visualize such a path though.) If omitted, the vertex coordinates will be traversed in their given order to create the path.
do_vis	logical, whether to actually draw the path.
color	a color string, like <code>'#FF0000'</code> to color the path.
no_material	logical, whether to use set the custom rendering material properties for path visualization using <code>rgl::material3d</code> before plotting. If you set this to <code>FALSE</code> , no material will be set and you should set it yourself before calling this function, otherwise the looks of the path are undefined (dependent on the default material on your system, or the last material call). Setting this to <code>TRUE</code> also means that the 'color' argument is ignored of course, as the color is part of the material.

Value

n x 3 matrix, the coordinates of the path, with appropriate ones duplicated for `rgl` pair-wise segments3d rendering.

See Also

[vis.paths](#) if you need to draw many paths, [geodesic.path](#) to compute a geodesic path.

Other surface mesh functions: [face.edges\(\)](#), [label.border\(\)](#), [mesh.vertex.included.faces\(\)](#), [mesh.vertex.neighbors\(\)](#), [subject.surface\(\)](#)

Examples

```

## Not run:
sjd = fsaverage.path(TRUE);
surface = subject.surface(sjd, 'fsaverage3',
  surface = "white", hemi = "lh");
p = geodesic.path(surface, 5, c(10, 20));
vis.subject.morph.native(sjd, 'fsaverage3', views='si');
vis.path.along.verts(surface$vertices, p[[1]]);

## End(Not run)

```

vis.paths	<i>Visualize many paths.</i>
-----------	------------------------------

Description

Visualize many paths.

Usage

```
vis.paths(coords_list, path_color = "#FF0000")
```

Arguments

coords_list	list of m matrices, each n x 3 matrix must contain the 3D coords for one path.
path_color	a color value, the color in which to plot the paths.

Note

This function is a lot faster than calling vis.path.along.verts many times and having it draw each time.

vis.paths.along.verts	<i>Visualize several paths in different colors.</i>
-----------------------	---

Description

Visualize several paths in different colors.

Usage

```
vis.paths.along.verts(
  surface_vertices,
  paths,
  color = viridis::viridis(length(paths))
)
```

Arguments

surface_vertices	float matrix of size (n, 3), the surface vertex coordinates, as returned as part of subject.surface or read.fs.surface , in the member "vertices". Can also be a freesurferformats::fs.surface or rgl::tmesh3d instance, in which case the coordinates are extracted automatically.
paths	list of positive integer vectors, the vertex indices of the paths
color	a color string, like '#FF0000' to color the path.

vis.region.values.on.subject

Visualize arbitrary data, one value per atlas region, on the surface of any subject (including template subjects).

Description

This function can be used for rendering a single value (color) for all vertices of an atlas region. The typical usecase is the visualization of results of atlas-based analyses, e.g., p-value, means or other aggregated values over all vertices of a region.

Usage

```
vis.region.values.on.subject(
  subjects_dir,
  subject_id,
  atlas,
  lh_region_value_list,
  rh_region_value_list,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  value_for_unlisted_regions = NA,
  draw_colorbar = FALSE,
  makecmap_options = mkco.heat(),
  bg = NULL,
  silent = FALSE,
  style = "default",
  border = NULL
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
atlas	string. The brain atlas to use. E.g., 'aparc' or 'aparc.a2009s'.
lh_region_value_list	named list. A list for the left hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region. You can pass an unnamed list, but then the its length must exactly match the number of atlas regions. The order of values must also match the order of regions in the annotation, of course. The resulting mapping will be printed so you can check it (unless 'silent' is set).

rh_region_value_list	named list. A list for the right hemisphere in which the names are atlas regions, and the values are the value to write to all vertices of that region.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
value_for_unlisted_regions	numerical scalar or NA, the value to assign to regions which do not occur in the region_value_lists. Defaults to NA.
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
silent	logical, whether to suppress mapping info in case of unnamed region value lists (see 'lh_region_value_list' description).
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
border	logical, whether to add a black border around the regions. Alternatively, the parameter can be given as a named list with entries 'color' and 'expand_inwards', where the latter defines the borders thickness. E.g., <code>border = list('color'='#FF0000', 'expand_inwards'=2L)</code> . Border computation is slow, sorry.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#),

[vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#),
[vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.colored](#)

Other region-based visualization functions: [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
atlas = 'aparc'; # Desikan atlas
# For the left hemisphere, we just assign a subset of the
# atlas regions. The others will get the default value.
lh_region_value_list = list("bankssts"=0.9, "precuneus"=0.7, "postcentral"=0.8, "lingual"=0.6);
# For the right hemisphere, we retrieve the full list of regions for
# the atlas, and assign random values to all of them.
atlas_region_names = get.atlas.region.names(atlas, template_subjects_dir = subjects_dir,
  template_subject='subject1');
rh_region_value_list = rnorm(length(atlas_region_names), 3.0, 1.0);
names(rh_region_value_list) = atlas_region_names;
vis.region.values.on.subject(subjects_dir, 'subject1', atlas,
  lh_region_value_list, rh_region_value_list);

## End(Not run)
```

vis.rglwidget

Visualize coloredmeshes as an interactive rgl WebGL widget for use in R Shiny apps and RMarkdown documents.

Description

Create an interactive 3D brain view that can be embedded in R Shiny applications or RMarkdown/HTML documents. Unlike the standard `vis.*` functions which open an X11/OpenGL window and are designed for static screenshot export, this function renders to a headless rgl device and returns an [rglwidget](#) object that provides interactive 3D rendering in a web browser. The user can rotate, zoom, and pan the brain in the widget.

Usage

```
vis.rglwidget(
  coloredmeshes,
  background = "white",
  skip_all_na = TRUE,
  style = "default",
  rgloptions = rglo(),
  ...
)
```

Arguments

coloredmeshes	a hemilist of coloredmeshes (as returned by vis.subject.morph.native and similar functions when called with <code>rglactions=list('no_vis'=TRUE)</code>) or a flat list of coloredmesh instances.
background	string, background color passed to bg3d . Defaults to "white".
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.
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	named list. Parameters passed to par3d . Defaults to the value returned by rglo .
...	extra arguments passed to rglwidget .

Value

an `htmlwidget` object from the `rgl` package, suitable for use with [rglwidgetOutput](#) / [renderRglwidget](#) in Shiny, or for direct embedding in RMarkdown HTML output.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
fsbrain::download_fsaverage(accept_freesurfer_license=TRUE);
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

# Get coloredmeshes without opening a window:
cm = vis.subject.annot(subjects_dir, 'subject1', 'aparc', 'both',
  rglactions=list('no_vis'=TRUE));

# Create an interactive WebGL widget:
widget = vis.rglwidget(cm);

# In RMarkdown, just print it:
# widget

# In Shiny, use rglwidgetOutput / renderRglwidget (see the shiny demo app).

## End(Not run)
```

vis.seg.legend

*Plot legend for a brain volume segmentation based on colorLUT.***Description**

Plot legend for a brain volume segmentation based on colorLUT.

Usage

```
vis.seg.legend(colortable, segvol, ...)
```

Arguments

colortable	a colortable data.frame, or a character string, which will be treated as a filename and loaded with read.fs.colortable . Typically FS_HOME/FreeSurferColorLUT.txt.
segvol	optional 3D or 4D array of integer data, the brain segmentation. Or a character string, which will be treated as a filename and loaded with read.fs.volume . If given, only colortable entries which actually occur in the volume data are plotted. If NULL, all entries are plotted, which may be a lot.
...	passed on to vis.colortable.legend

Examples

```
## Not run:
ct = file.path(fs.home(), "FreeSurferColorLUT.txt");
seg = file.path(fs.home(), "subjects", "fsaverage", "mri", "aseg.mgz");
vis.seg.legend(ct, seg);

## End(Not run)
```

vis.subcortical.region.values

*Visualize one value per region of the subcortical atlas of a subject.***Description**

Render the subcortical structures of a subject and assign one color per structure, based on one value per atlas region. The subcortical atlas is not defined on the cortical surface: it comes with its own surface mesh that contains the 8 subcortical structures per hemisphere (accumbens area, amygdala, caudate, hippocampus, pallidum, putamen, thalamus and lateral ventricle). The mesh and the annotation file are expected in the subject directory ('surf/lh.subcortical', 'surf/rh.subcortical', 'label/lh.subcortical.annot', 'label/rh.subcortical.annot').

The atlas files for the fsaverage template subject are not part of FreeSurfer and are not required for the package to work. They can be downloaded with [download_optional_data](#) or [download_fsaverage_atlases](#) into the package cache, which is searched for a subject named 'fsaverage' by default.

Optionally, the structures can be rendered inside a semi-transparent context mesh, typically the cortex of the same subject (see parameter 'cortex'). Note that the context mesh and the atlas mesh must be defined in the same coordinate space, which is why both are taken from the same subject by default.

Usage

```
vis.subcortical.region.values(
  subjects_dir = NULL,
  subject_id = "fsaverage",
  lh_region_value_list,
  rh_region_value_list,
  atlas = "subcortical",
  surface = "subcortical",
  cortex = NULL,
  views = c("sd_lateral_lh", "sd_medial_lh", "sd_lateral_rh", "sd_medial_rh"),
  rgloptions = rglo(),
  rglactions = list(),
  value_for_unlisted_regions = NA,
  draw_colorbar = FALSE,
  makecmap_options = mkco.seq(),
  style = "default",
  silent = FALSE
)
```

Arguments

<code>subjects_dir</code>	string or NULL. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier. If NULL, the locations searched by find.subjectsdir.of (package cache and FreeSurfer/SUBJECTS_DIR configuration) are checked for one that contains the atlas files, and the first such location is used.
<code>subject_id</code>	string. The subject identifier. Defaults to 'fsaverage', the template subject for which the subcortical atlas is available for download.
<code>lh_region_value_list</code>	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, see vis.region.values.on.subject . Use NaN as the value of a region to hide it completely: the vertices of that region are removed from the mesh and it is not rendered at all, see the 'details' section.
<code>rh_region_value_list</code>	named list, the same for the right hemisphere.
<code>atlas</code>	string. The name of the atlas to use. Defaults to 'subcortical', the subcortical atlas that ships with the package. Used to construct the annotation file name.

surface	string. The name of the surface mesh that belongs to the atlas. Defaults to 'subcortical'. Used to construct the surface file name, in contrast to the other vis functions this is not a cortical surface.
cortex	NULL or the definition of a context mesh to render the structures in, typically a semi-transparent cortex of the same subject. Supported values are a character string (the surface name, e.g., 'white' or 'pial'), a named list of options for the context mesh (entries 'surface', 'color', 'alpha', 'style', 'subjects_dir' and 'subject_id'), an coloredmesh.from.color instance, or a hemilist of such instances. Use NULL to render the structures without any context.
views	list of strings. The views to render. Defaults to lateral and medial views of both hemispheres. See get.view.angle.names for valid entries.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions, see rglactions . Note that the action 'shift_hemis_apart' is not supported here: the structures of a mesh atlas are rendered in their anatomical position.
value_for_unlisted_regions	numerical scalar or NA, the value to assign to regions which do not occur in the region value lists, see vis.region.values.on.subject . Set this to NaN to hide all regions that are not listed explicitly.
draw_colorbar	logical. Whether to draw a colorbar. Defaults to FALSE, see coloredmesh.plot.colorbar.separate for a better looking alternative.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from the data.
style	a rendering style for the data meshes, see get.rglstyle . Defaults to 'default'. The context mesh defined via parameter 'cortex' has its own style, see there.
silent	logical, whether to suppress messages.

Value

list of coloredmeshes. The coloredmeshes used for the visualization, invisibly. The list contains the data meshes (and the context meshes, if any) as a flat list, so it can be passed to [export](#) to save the rendered views into an image file.

Hiding regions

Assigning the value NaN to a region hides it: the vertices of the region are removed from the mesh (together with the faces that use them), so the structure is not rendered at all, in contrast to drawing it in the color that represents missing data. This is a per-vertex operation on the shared mesh of all regions of a hemisphere, so hiding is not limited to entire meshes and can be combined with any rendering style.

The values of hidden regions are excluded from the colorbar range, just like NA values. Setting `value_for_unlisted_regions = NaN` hides all regions that are not listed in the region value lists, which is a convenient way of visualizing only a few structures of an atlas.

Note

The subcortical atlas is defined in MNI305 space (fsaverage surface RAS), so it can be combined with the cortical surfaces of the fsaverage template subject. The 8 structures per hemisphere are colored with the standard FreeSurfer 'aseg' colors when you visualize the atlas itself, this function assigns data-driven colors instead. Region names are the FreeSurfer 'aseg' structure names, e.g., 'Left-Hippocampus' or 'Right-Thalamus-Proper'.

This function is not limited to the subcortical atlas: any atlas that comes with its own mesh and annotation files works, pass the respective names via parameters 'atlas' and 'surface'.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.colored](#)

Other region-based visualization functions: [vis.region.values.on.subject\(\)](#), [vis.subject.annot\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data(); # includes the subcortical atlas for fsaverage
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

# One value per region, for all 8 subcortical structures of the left and right hemisphere.
lh_region_values = list("Left-Accumbens-area"=0.1, "Left-Amygdala"=0.2,
  "Left-Caudate"=0.3, "Left-Hippocampus"=0.4, "Left-Pallidum"=0.5,
  "Left-Putamen"=0.6, "Left-Thalamus-Proper"=0.7, "Left-Lateral-Ventricle"=0.8);
rh_region_values = list("Right-Accumbens-area"=0.1, "Right-Amygdala"=0.2,
  "Right-Caudate"=0.3, "Right-Hippocampus"=0.4, "Right-Pallidum"=0.5,
  "Right-Putamen"=0.6, "Right-Thalamus-Proper"=0.7, "Right-Lateral-Ventricle"=0.8);

# Render the structures on their own, and save the result to a file:
cm = vis.subcortical.region.values(subjects_dir, "fsaverage", lh_region_values,
  rh_region_values, rglactions = list("no_vis" = TRUE));
export(cm, colorbar_legend = "my values", output_img = "subcortical.png");

# Render the structures inside a semi-transparent cortex:
cm_ctx = vis.subcortical.region.values(subjects_dir, "fsaverage", lh_region_values,
  rh_region_values, cortex = "white", rglactions = list("no_vis" = TRUE));
export(cm_ctx, colorbar_legend = "my values", output_img = "subcortical_in_cortex.png");

# Visualize only the two hippocampi, by hiding all other regions (they are set to NaN):
cm_hippo = vis.subcortical.region.values(subjects_dir, "fsaverage",
  list("Left-Hippocampus" = 0.2), list("Right-Hippocampus" = 0.8),
  value_for_unlisted_regions = NaN, rglactions = list("no_vis" = TRUE));
export(cm_hippo, colorbar_legend = "my values", output_img = "subcortical_hippocampi.png");

## End(Not run)
```

vis.subject.annot	<i>Visualize an annotation for a subject.</i>
-------------------	---

Description

Creates a surface mesh, loads the colors from the annotation, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.subject.annot(
    subjects_dir,
    subject_id,
    atlas,
    hemi = "both",
    surface = "white",
    views = c("t4"),
    rgloptions = rglo(),
    rglactions = list(),
    outline = FALSE,
    style = "default"
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded. Can also be a hemilist of already loaded annotations.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95)). See rglactions .

outline	logical, whether to draw an outline only instead of filling the regions. Defaults to FALSE. Instead of passing TRUE, one can also pass a list of extra parameters to pass to <code>annot.outline</code> , e.g., <code>outline=list('outline_color'='#000000')</code> . Using this increases computation time dramatically, sorry for the performance.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: `highlight.vertices.on.subject()`, `highlight.vertices.on.subject.spheres()`, `vis.color.on.subject()`, `vis.data.on.fsaverage()`, `vis.data.on.subject()`, `vis.labeldata.on.subject()`, `vis.mask.on.subject()`, `vis.region.values.on.subject()`, `vis.rglwidget()`, `vis.subcortical.region.values()`, `vis.subject.label()`, `vis.subject.morph.native()`, `vis.subject.morph.standard()`, `vis.subject.pre()`, `vis.symmetric.data.on.subject()`, `vis.volume.clusters()`, `vis.volume.on.surface()`, `vislayout.from.colored`

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

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
vis.subject.annot(subjects_dir, 'subject1', 'aparc', 'both');

## End(Not run)
```

vis.subject.label	<i>Visualize a binary label for a subject.</i>
-------------------	--

Description

Visualize a label for a subject. A label is just a logical vector with one entry for each vertex in the mesh. Each vertex may additionally be associated with a scalar value, but this function ignored that.

Usage

```
vis.subject.label(
  subjects_dir,
  subject_id,
  label,
  hemi,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
```

```

draw_colorbar = FALSE,
makecmap_options = list(colFn = label.colFn.inv, col.na = "#FFFFFF00"),
map_to_NA = 0L,
bg = NULL,
style = "default"
)

```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
label	string. Name of the label file, without the hemi part (if any), but including the '.label' suffix. E.g., 'cortex.label' for '?h.cortex.label'.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
map_to_NA	the value or value range that should not be considered part of the label, and should thus be plotted as background color. Only used if 'bg' is not NULL. If a single value, only 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 like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

Note

Drawing a colorbar for label data makes limited sense, use a legend instead. The colorbar can give a rough overview of the relative number of label and non-label vertices though, so it is possible to request one.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other label functions: [apply.label.to.morphdata\(\)](#), [apply.labeldata.to.morphdata\(\)](#), [subject.lobes\(\)](#), [subject.mask\(\)](#), [vis.labeldata.on.subject\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
subject_id = 'subject1';
surface = 'white';
hemi = 'both';
label = 'cortex.label';
vis.subject.label(subjects_dir, subject_id, label, hemi, views="si");

## End(Not run)
```

vis.subject.morph.native

Visualize native space morphometry data for a subject.

Description

Creates a surface mesh, applies a colormap transform the morphometry data values into colors, and renders the resulting colored mesh in an interactive window. If hemi is 'both', the data is rendered for the whole brain.

Usage

```
vis.subject.morph.native(
    subjects_dir,
    subject_id,
    measure,
    hemi = "both",
    surface = "white",
    views = c("t4"),
    rgloptions = rglo(),
    rglactions = list(),
    draw_colorbar = FALSE,
    cortex_only = FALSE,
    style = "default",
    makecmap_options = mkco.seq(),
    bg = NULL
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier.
measure	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file

	label/?h.cortex.label) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
vis.subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh', views=c("t9"));

## End(Not run)
```

vis.subject.morph.standard

Visualize native space morphometry data for a subject or a group.

Description

Renders standard space morphometry data for a single subject, or the group mean for a group of subjects. The default template subject is fsaverage.

Usage

```
vis.subject.morph.standard(
    subjects_dir,
    subject_id,
    measure,
    hemi = "both",
    fwhm = "10",
    surface = "white",
    template_subject = "fsaverage",
    template_subjects_dir = NULL,
    views = c("t4"),
    rgloptions = rglo(),
    rglactions = list(),
    draw_colorbar = FALSE,
    cortex_only = FALSE,
    makecmap_options = mkco.seq(),
    bg = NULL,
    style = "default"
)
```

Arguments

<code>subjects_dir</code>	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.
<code>subject_id</code>	character string or vector of character strings, the subject or subjects. For a single subjects, its data will be plotted. If a group of subjects is given instead, at each vertex the mean value over all the subjects will be plotted.
<code>measure</code>	string. The morphometry data to use. E.g., 'area' or 'thickness'. Pass NULL to render just the surface in white, without any data.
<code>hemi</code>	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
<code>fwhm</code>	string, smoothing setting (full width at half maximum of the kernel). The smoothing part of the filename, typically something like '0', '5', '10', ..., or '25'.
<code>surface</code>	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
<code>template_subject</code>	The template subject used. This will be used as part of the filename, and its surfaces are loaded for data visualization. Defaults to 'fsaverage'.
<code>template_subjects_dir</code>	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')'.
<code>views</code>	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: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
cortex_only	logical, whether to mask the medial wall, i.e., whether the morphometry data for all vertices which are <i>not</i> part of the cortex (as defined by the label file <code>label/?h.cortex.label</code>) should be replaced with NA values. In other words, setting this to TRUE will ignore the values of the medial wall between the two hemispheres. If set to true, the mentioned label file needs to exist for the subject. Defaults to FALSE.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
bg	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [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

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
fsaverage_dir = file.path(Sys.getenv('FREESURFER_HOME'), 'subjects');
if(dir.exists(fsaverage_dir)) {
  vis.subject.morph.standard(subjects_dir, 'subject1', 'thickness', 'lh',
```

```

    '10', template_subjects_dir=fsaverage_dir);
}
# The last command will load the file
# *<subjects_dir>/subject1/surf/lh.thickness.fwhm10.fsaverage.mgh* and
# visualize the data on *$FREESURFER_HOME/subjects/fsaverage/surf/lh.white*.

## End(Not run)

```

vis.subject.pre	<i>Visualize pre-loaded data.</i>
-----------------	-----------------------------------

Description

Visualize pre-loaded data.

Usage

```

vis.subject.pre(
  surfaces,
  pvertex_data,
  hemi = "both",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = FALSE,
  style = "default",
  makecmap_options = mkco.seq()
)

```

Arguments

surfaces	a hemilist of surfaces loaded with a function like <code>freesurferformats::read.fs.surface</code> .
pvertex_data	a hemilist of per-vertex data for the surfaces, i.e., a list of numeric vectors. E.g., loaded from a morphometry data file with a function like <code>freesurferformats::read.fs.morph</code> .
hemi	string, one of 'lh', 'rh', or 'both'. The hemisphere name. Used to construct the names of the label data files to be loaded.
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> .
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: <code>rglactions = list("snapshot_png"=~"/fsbrain.png", "clip_data"=c(0.05, 0.95))</code> . See rglactions .

draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.colored](#)

vis.symmetric.data.on.subject

Visualize clusters or activation data on the surface of any subject.

Description

This function is intended to plot symmetric data around zero (like positive and negative activation data, signed p-values, etc.) on a subject's surface. It is a thin wrapper around [vis.data.on.subject](#).

Usage

```
vis.symmetric.data.on.subject(
  subjects_dir,
  vis_subject_id,
  morph_data_lh = NULL,
  morph_data_rh = NULL,
  surface = "white",
  views = c("t4"),
  rgloptions = rglo(),
  rglactions = list(),
  draw_colorbar = TRUE,
  makecmap_options = list(colFn = cm.cbry(), symm = TRUE, col.na = "#FFFFFF00", n = 200),
  map_to_NA = c(0),
  bg = NULL,
  morph_data_both = NULL,
  style = "default"
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
vis_subject_id	string. The subject identifier from which to obtain the surface for data visualization. Example: 'fsaverage'.
morph_data_lh	numeric vector or character string or NULL, the data to visualize on the left hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
morph_data_rh	numeric vector or character string or NULL, the data to visualize on the right hemisphere surface. If a string, it is treated as a filename and data is loaded from it first. When it is a numerical vector, this is assumed to be the data already. The data must have the same length as the surface of the vis_subject_id has vertices. If NULL, this surface will not be rendered. Only one of morph_data_lh or morph_data_rh is allowed to be NULL.
surface	string. The display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
views	list of strings. Valid entries include: 'si': single interactive view. 't4': tiled view showing the brain from 4 angles. 't9': tiled view showing the brain from 9 angles.
rgloptions	option list passed to par3d . Example: rgloptions = list("windowRect"=c(50,50,1000,1000)).
rglactions	named list. A list in which the names are from a set of pre-defined actions. The values can be used to specify parameters for the action. The following example clips outliers in the data before plotting and writes a screenshot in PNG format: rglactions = list("snapshot_png"="~/fsbrain.png", "clip_data"=c(0.05, 0.95)). See rglactions .
draw_colorbar	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar. Notice: the colorbar is drawn to a separate subplot, and this only works if there is enough space for it, i.e., the plot resolution must be high enough. You may have to increase the plot size for the colorbar to show up, see the vignette for instructions. Defaults to FALSE. See coloredmesh.plot.colorbar.separate for an alternative.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from 'measure'. Should include at least a colormap function as name 'colFn'.
map_to_NA	the value or value range that should not be considered a cluster, and should thus be plotted as background color. These values will be set to NA, leading to transparent rendering, so the background will be visible instead. If a single value, only exactly this value is used (typically 0). If two values, they are interpreted as a range, and a values between them are mapped to NA. If you prefer to map the data to NA yourself before using this function or do not want to use a , pass NULL.

bg	a background definition. Can be a surface color layer or a character string like 'curv_light' to select a pre-defined layer, see collayer.bg for valid strings.
morph_data_both	numeric vector or NULL, the data to visualize on both hemispheres. This must be a single vector with length equal to the sum of the vertex counts of the left and the right hemisphere. The data for the left hemisphere must come first. If this is given, 'morph_data_lh' and 'morph_data_rh' must be NULL.
style	character string, a rendering style, e.g., 'default', 'shiny' or 'semitransparent'.

Value

list of coloredmeshes. The coloredmeshes used for the visualization.

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other morphometry visualization functions: [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
morph_data_lh = subject.morph.native(subjects_dir, 'subject1', 'thickness', 'lh');
morph_data_rh = NULL;
vis.symmetric.data.on.subject(subjects_dir, 'subject1', morph_data_lh, morph_data_rh);

## End(Not run)
```

vis.tracts

Visualize white matter tracts (streamlines) on the cortical surface.

Description

Draws white matter streamlines (tractography data) as lines, with an optional semi-transparent brain surface as context. The streamlines can come from a tract file (TRK or TCK format), from a directory of such files (a tract atlas with one file per bundle, e.g. the XTRACT atlas, see [download_xtract_tracts](#)), or from in-memory data. This is the fsbrain equivalent of the plot_tracts function of the Python package yabplot.

Usage

```
vis.tracts(
  tracts,
  bundle_values = NULL,
  subjects_dir = NULL,
  template_id = "fs_LR_32",
  context = list(surface = "midthickness", alpha = 0.08, color = "#B0B0B0"),
  coords = "ras",
  transform_matrix = NULL,
  max_tracks = Inf,
  skip_tracks = 0L,
  bbox = NULL,
  tract_color = NULL,
  tract_makecmap_options = mkco.seq(),
  tract_width = 1,
  tract_width_range = NULL,
  tract_width_scale = c("none", "value"),
  color_by_orientation = FALSE,
  views = c("sd_lateral_lh", "sd_medial_lh", "sd_lateral_rh", "sd_medial_rh"),
  rgloptions = rglo(),
  rglactions = list(),
  style = "default",
  draw_colorbar = FALSE,
  silent = FALSE
)
```

Arguments

tracts	the tracts to draw. One of: a character string, the path to a tract file (TRK or TCK format) or to a directory containing one file per bundle (a tract atlas); a vector of tract file paths; an fs.tracts instance (freesurferformats); an (n, 3) matrix of coordinates (a single streamline); or a list of such objects. A list of (n, 3) matrices is interpreted as a single bundle (one tractogram), a list of fs.tracts instances as one bundle per entry. For the atlas case, the bundle names are the file names without extension.
bundle_values	numeric vector or NULL, one value per bundle, in the order of the bundles (or named with the bundle names). The values are mapped to colors, see the details.
subjects_dir	character string or NULL, the subjects directory that contains the template. If NULL, the standard fsbrain locations are searched, see resolve.template.subjects.dir .
template_id	character string, the template or subject to draw the context surface for. Defaults to 'fs_LR_32', the HCP-style surface space that is widely used for tractography data. Note that the template meshes are not part of this package, see download_fs_LR_32_meshes .
context	named list or NULL, the semi-transparent brain surface drawn around the tracts. Entries are 'surface' (the surface name, default 'midthickness'), 'alpha' (the transparency, default 0.08) and 'color' (default '#B0B0B0'). Set to NULL to draw the tracts without any context surface.

coords	character string or NULL, the coordinate system of TRK files, see read.tract.bundles . Ignored for in-memory data.
transform_matrix	a 4x4 numeric matrix or NULL, a transformation that is applied to all tract coordinates after reading them. Note that tract data and template surfaces have to be in the same space for the overlay to be correct: the XTRACT atlas, for example, is defined in MNI152 space, while the fs_LR_32 and fsaverage templates use an fsaverage-like (MNI305) space, which is close but not identical, and both are close enough for the overlay to look right without a transformation. Use this parameter if you have an accurate transformation, e.g. from a registration of the template to MNI152 space.
max_tracks	numeric, the maximum number of streamlines to read per file, see read.tract.bundles . This is the way to plot a subset of a whole-brain tractogram, which can contain millions of streamlines and cannot be drawn (or held in memory) as a whole.
skip_tracks	integer, the number of streamlines to skip per file, see read.tract.bundles .
bbox	numeric vector of length 6 or NULL, a bounding box to select the streamlines that pass through a region, see read.tract.bundles .
tract_color	vector of hex color strings, the color(s) of the tracts. Either a single color which is used for all bundles, or one color per bundle. Ignored if bundle_values is given. If both tract_color and bundle_values are NULL and color_by_orientation is FALSE, all tracts are drawn in a single default color.
tract_makecmap_options	named list of colormap options, see mkco.seq , used to map bundle_values to colors.
tract_width	a single positive number, the line width in pixels.
tract_width_range	numeric vector of length 2 or NULL, the range of the line widths used when the widths are scaled by the bundle values, see tract_width_scale. Defaults to c(0.5, 1.5) * tract_width.
tract_width_scale	character string, either 'none' (all lines have the same width, the default) or 'value' (the line width encodes the bundle values, which requires bundle_values to be given).
color_by_orientation	logical, whether to color the segments by their direction instead of by bundle, see the details. The colors then do not encode the bundle values, so no colorbar is drawn.
views	vector of character strings, the views to render, see brainviews . Pass NULL to only compute the renderables without rendering anything (e.g. to pass them to export).
rgloptions	named list, options for <code>rgl::par3d()</code> , see rglo .
rglactions	named list, actions to perform, see rglactions .
style	character string, the rendering style for the tracts, see material3d . The context surface is always rendered with its own fixed style, so that the requested transparency is used.

<code>draw_colorbar</code>	logical or one of the character strings 'vertical' or 'horizontal', whether to draw a colorbar for the bundle values. Note that the headless scimesh renderer backend does not support colorbars in this function, use export instead (see the return value).
<code>silent</code>	logical, whether to suppress the progress messages.

Details

The streamlines are drawn with the line renderable of the connectome code, see [fs.coloredpaths](#), so the result is a regular fsbrain renderable list which works with both renderer backends (rgl and the headless scimesh renderer) and can be passed to [export](#) to create publication quality figures with a colorbar.

The colors can be defined in three ways: (1) as one value per bundle via `bundle_values` (e.g. a tractometry measure like the mean fractional anisotropy of each bundle), which is mapped to colors with `tract_makecmap_options` and produces a colorbar; (2) as a single color for all bundles via `tract_color`; or (3) per segment, based on the direction of the segment, via `color_by_orientation`, which is the classic DTI look (red/green/blue for left-right/anterior-posterior/superior-inferior).

Value

invisible named list of renderables: the entry `tracts` are the lines (an `fs.coloredpaths` instance), and `context_lh` and `context_rh` the context surfaces (if context is not NULL). The list can be passed to [export](#).

See Also

Other tracts functions: [download_xtract_tracts\(\)](#), [read.tract.bundles\(\)](#)

Examples

```
## Not run:
# Download the XTRACT tract atlas (see the function documentation):
download_xtract_tracts("xtract_tiny");

atlas_dir = file.path(get_optional_data_filepath("tracts"), "xtract_tiny");
bundles = read.tract.bundles(atlas_dir);

# One color per bundle, colors follow a colormap, with a colorbar in the figure:
values = seq(0.2, 0.8, length.out = length(bundles));
names(values) = names(bundles);
tracts = vis.tracts(bundles, bundle_values = values,
  views = c("sd_lateral_lh", "sd_lateral_rh"));

# Or a publication quality figure with 3 views and a colorbar:
export(tracts, view_angles = c("sd_lateral_lh", "sd_dorsal", "sd_rostral"),
  draw_colorbar = "horizontal", colorbar_legend = "Mean FA",
  output_img = "tracts.png");

# Draw a whole-brain tractogram from MRtrix (a subset, it is huge):
vis.tracts("~/data/sub-01_streamlines.tck.gz", max_tracks = 20000);
```

```
## End(Not run)
```

vis.volume.clusters	<i>Visualize clusters of a volume inside a translucent anatomical mesh.</i>
---------------------	---

Description

Visualize the clusters of a volume, e.g., the supra-threshold voxels of a statistical map, as 3D iso-surfaces inside a single semi-transparent anatomical mesh (the *context*, typically the cortex of the same subject). A cluster is rendered as a set of nested iso-surface shells: the outermost shell is the iso-surface at the threshold (the cluster boundary), the inner ones sit at the higher iso-levels, and the innermost shell is at the cluster peak. This way the shape of the cluster and the location of its peak are visible at the same time, and the voxel structure of the data does not show up as staircase artifacts.

The shells are colored with a diverging colormap that encodes the iso-level, i.e., they use exactly the colors that the colorbar shows, and they become more opaque towards the core of the cluster, so that the core is visible through the outer shells. Positive and negative clusters can be rendered together, they use the two halves of the diverging colormap.

To render the clusters without a context mesh, pass `context = NULL`. To show the location of the clusters within a specific subject, use the `context` parameter to select the surface (e.g., "white" or "pial"); the surfaces of the fsaverage template subject can be downloaded with [download_fsaverage](#) if needed.

Usage

```
vis.volume.clusters(
  subjects_dir = NULL,
  subject_id = "fsaverage",
  volume,
  threshold = NULL,
  threshold_quantile = 0.95,
  max_level = NULL,
  max_level_quantile = 0.99,
  num_levels = 4L,
  positive = TRUE,
  negative = TRUE,
  smoothing = 2L,
  downsample = 1L,
  frame = 1L,
  backend = "auto",
  context = list(surface = "white", alpha = 0.05),
  alpha_range = c(0.08, 1),
  makecmap_options = mkco.cluster(),
  views = c("sd_lateral_lh", "sd_medial_lh", "sd_lateral_rh", "sd_medial_rh"),
  rgloptions = rglo(),
```

```

    rglaactions = list(),
    silent = FALSE
)

```

Arguments

<code>subjects_dir</code>	string or NULL. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier. If NULL (the default), the subject directory is searched in the package cache, SUBJECTS_DIR and FREESURFER_HOME, see find.subjectsdir.of . If the subject is the template subject 'fsaverage' and it cannot be found, it is downloaded into the package cache.
<code>subject_id</code>	string. The subject identifier, used to load the context mesh. Defaults to 'fsaverage', the template subject for which the surfaces can be downloaded automatically.
<code>volume</code>	a 3D numerical array (or an <code>fs.volume</code> instance) with the data, e.g., a statistical map. Values are interpreted as signed, i.e., the clusters are the regions in which the value is above threshold or below <code>-threshold</code> . See the note on the coordinate system.
<code>threshold</code>	single positive number or NULL. The cluster threshold, applied to the absolute voxel values. If NULL (the default), it is computed from the data as the <code>threshold_quantile</code> quantile of the absolute values of the non-zero voxels, and the value is reported unless <code>silent</code> is TRUE.
<code>threshold_quantile</code>	single value between 0 and 1, the quantile of the absolute non-zero voxel values used as the threshold if <code>threshold</code> is NULL. Defaults to 0.95.
<code>max_level</code>	single positive number or NULL, the highest iso-level, i.e., the level of the innermost shell of every cluster, in the units of the volume data. If NULL (the default), a separate value is computed for the positive and the negative clusters, namely the <code>max_level_quantile</code> quantile of the absolute values of the voxels beyond the threshold, so that a single extreme voxel does not dominate the visualization.
<code>max_level_quantile</code>	single value between 0 and 1, the quantile of the supra-threshold voxel values used to compute <code>max_level</code> if that is NULL. Defaults to 0.99.
<code>num_levels</code>	positive integer, the number of nested shells per cluster, i.e., per sign. The outermost shell is at the threshold, the innermost at <code>max_level</code> . Defaults to 10. Pass 1 to render a single opaque shell per sign, see the note on the visual style.
<code>positive</code>	logical, whether to render the clusters above threshold. Defaults to TRUE.
<code>negative</code>	logical, whether to render the clusters below <code>-threshold</code> . Defaults to TRUE. At least one of <code>positive</code> and <code>negative</code> must be TRUE.
<code>smoothing</code>	non-negative integer, the number of 3x3x3 box blur passes applied to the volume before the shells are extracted. This removes the staircase artifacts of the marching cubes algorithm and is what makes the clusters look like smooth blobs. Defaults to 2. Set to 0 to extract the shells from the raw volume.
<code>downsample</code>	positive integer, a factor by which the volume is subsampled before the shells are extracted. Values larger than 1 reduce the number of triangles (and thus the

	rendering time) considerably, at the cost of a less detailed surface, which can be useful because a cluster is rendered as <code>num_levels</code> meshes. The coordinates of the subsampled shells are corrected, so the result remains aligned with the original volume. Defaults to 1. Note that the volume is smoothed before subsampling if <code>smoothing</code> is larger than 0, otherwise the shells are prone to aliasing artifacts.
frame	positive integer, the frame (4th dimension) to use for a 4D volume. Defaults to 1.
backend	character string, the backend used to extract the iso-surfaces, one of 'auto' (the default: use the <code>Rvcg</code> package if it is installed, and fall back to <code>misc3d</code> otherwise), 'Rvcg' or 'misc3d'. <code>Rvcg</code> is recommended, it is faster, which matters here because one mesh per shell is extracted.
context	NULL or the definition of the context mesh to render the clusters in, typically a semi-transparent cortex of the same subject. Supported values are a character string (the surface name, e.g., 'white' or 'pial'), a named list of options for the context mesh (entries 'surface', 'color', 'alpha', 'style', 'subjects_dir' and 'subject_id', see vis.subcortical.region.values), or a pre-built mesh (an <code>fs.coloredmesh</code> instance, or a hemilist of such instances).
alpha_range	numerical vector of length 2, the alpha value of the outermost shell (at the threshold) and of the innermost shell (at the cluster peak), each between 0 and 1. The values in between are interpolated, separately for the positive and the negative clusters. Defaults to <code>c(0.08, 1.0)</code> , i.e., the boundary of a cluster is much more transparent than its core. See the note on the visual style.
makecmap_options	named list of parameters to pass to makecmap . Must not include the unnamed first parameter, which is derived from the data. The entry 'range' can be used to fix the range of the colormap, otherwise a symmetric range which covers all rendered iso-levels is used. Defaults to mkco.cluster .
views	list of character strings, the views to visualize. Available views are 'sd_<angle>' (a static view, see vis.subject.morph.native), 't4'/t9' (a 2x2 or 3x3 light-box of views) and 'si'/sr' (single interactive / rotating, requires the <code>rgl</code> renderer backend). Defaults to the 4 standard lateral and medial views. Pass NULL to skip the rendering entirely and only compute the meshes (which are also returned).
rgloptions	option list passed to par3d . Defaults to the package default, see rglo .
rglactions	named list, passed to the visualization functions, see rglactions . The key 'no_vis' is useful to prepare the scene and add other data to it later. Note that saving the rendered scene to an image file is done with export , which takes the return value of this function and supports merging several views into one image.
silent	logical, whether to suppress the console messages which report the threshold, the levels and the backend used. Defaults to FALSE.

Value

the list of renderable meshes, invisibly: the context meshes (if any) first, then the cluster shells, ordered from the outermost to the innermost shell of each sign. Passing this list to [export](#) saves the scene as an image, including the colorbar of the cluster values.

Coordinate system

The cluster shells are extracted in the voxel space of the volume and transformed to surface RAS with `index2ras_tkr`, so they are aligned with surface renderings of the same subject. This requires the volume to be in the standard FreeSurfer orientation with 1 mm voxels (a *conformed* volume), which is the case for volumes read with `subject.volume`; volumes in other spaces (like a 2 mm MNI template) have to be transformed to the subject space outside of fsbrain first.

Visual style

A single iso-surface contains, by definition, the threshold value at all of its vertices, so a color gradient *within* one shell is not meaningful. The value information is therefore encoded in the *nesting*: each shell is drawn in the color of its own iso-level (the outer shell in the color of the threshold, the innermost one in the color of the peak), and the shells are semi-transparent, with the transparency decreasing towards the core (see `alpha_range`). This results in a soft glow which is brightest at the location of the maximum. Setting `num_levels = 1` instead renders one fully opaque shell per sign (the cluster boundary), which is the cleanest option if only the extent of the clusters matters. Note that the backfaces of the shells are drawn (`back = 'filled'`): with culled backfaces, the rgl renderer would composite the nested transparent shells in the wrong order, so the innermost shell would end up on top of the outer ones.

Performance

A cluster is rendered as `num_levels` meshes, so the number of iso-surfaces that have to be extracted is `num_levels` times the number of signs (20 for the defaults). For a conformed volume of 256^3 voxels this takes about 30 seconds with the Rvcg backend, most of which is the marching cubes extraction. Setting `downsample = 2` reduces this to a few seconds (and the number of triangles by roughly a factor of 8), which is usually not visible for the smooth blobs that the default smoothing produces; it is a good choice for larger volumes, for small `num_levels` it matters less.

See Also

`volvis.shells` to visualize a whole volume (not just the clusters) as nested shells, `volvis.voxels` for a voxel-based rendering, `vis.volume.on.surface` to combine a volume with a morphometry-colored surface, and `vis.subcortical.region.values` for the related atlas version with one value per region.

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

Other volume visualization: `vis.volume.on.surface()`, `volvis.lb()`, `volvis.lb.with.surface()`, `volvis.lightbox()`, `volvis.slices.with.surface()`

Examples

```
## Not run:
fsbrain::download_optional_data();
fsbrain::download_fsaverage(accept_freesurfer_license = TRUE);
```

```

subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");

# Use the brain volume of the demo subject as a fake statistical map: bright voxels are
# 'positive', dark ones are 'negative'.
brain = subject.volume(subjects_dir, 'subject1', 'brain');
stat = (brain - 110) / 30;

# Render the clusters inside a semi-transparent cortex of the fsaverage template subject:
cm = vis.volume.clusters(subjects_dir, 'fsaverage', stat, threshold = 1.5,
  rglactions = list('no_vis' = TRUE));
export(cm, colorbar_legend = 'value', output_img = 'clusters.png');

# A single opaque shell per cluster, which only shows the cluster extent:
cm_simple = vis.volume.clusters(subjects_dir, 'fsaverage', stat, threshold = 1.5,
  num_levels = 1, rglactions = list('no_vis' = TRUE));

# Without any context mesh, and with a fixed colormap range:
cm_ctx = vis.volume.clusters(subjects_dir, 'fsaverage', stat, threshold = 1.5,
  context = NULL, makecmap_options = list('colFn' = squash::blueorange, 'n' = 100L,
    'symm' = TRUE, 'range' = c(-2.0, 2.0)), rglactions = list('no_vis' = TRUE));

## End(Not run)

```

vis.volume.on.surface *Visualize a brain volume overlaid on a cortical surface in 3D.*

Description

Render a brain volume (as an isosurface or as voxels) together with a cortical surface mesh colored by morphometry data in the same interactive 3D scene. The volume coordinates are transformed to surface RAS space using the FreeSurfer [vox2ras_tkr](#) matrix (more precisely [index2ras_tkr](#), see there for the R-index versus CRS distinction), ensuring proper spatial alignment of volume and surface.

Usage

```

vis.volume.on.surface(
  subjects_dir,
  subject_id,
  volume = "brain",
  volume_mode = "contour",
  volume_level = 80,
  volume_color = "#666666",
  volume_alpha = 0.3,
  measure = "thickness",
  surface = "white",
  hemi = "both",
  views = c("t4"),

```

```

    surface_style = "semitransparent",
    rgloptions = rglo(),
    makecmap_options = mkco.seq(),
    cortex_only = FALSE,
    render_every = 20L,
    ...
)

```

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	numeric 3D array or character string. Either a 3D volume array with voxel intensities, or the name of a volume file to load from the subject's <code>mri/</code> directory (e.g., "brain", "aseg", "aparc+aseg"). Background voxels should have value NA or 0.
volume_mode	character string, one of "contour" or "voxels". The rendering mode for the volume: "contour" creates a smooth isosurface using the <code>misc3d</code> package (requires the optional dependency <code>misc3d</code>), "voxels" renders individual foreground voxels as small cubes. Defaults to "contour".
volume_level	numeric scalar, the intensity threshold for contour mode. Only voxels with intensity values $\geq$ this threshold contribute to the isosurface. Ignored in voxel mode. Defaults to 80 (suitable for a raw brain.mgz with values in range 0-255).
volume_color	character string, the color for the volume rendering. Defaults to "#666666" (medium gray).
volume_alpha	numeric in range 0..1, the transparency (alpha) of the volume overlay. Lower values make the volume more transparent, revealing the surface beneath. Defaults to 0.3.
measure	character string or NULL, the morphometry data to use for coloring the surface. E.g., "thickness", "sulc", "area", or "curv". Pass NULL to render the surface in a single plain color without morphometry overlay. Defaults to "thickness".
surface	character string, the display surface. E.g., "white", "pial", or "inflated". Defaults to "white".
hemi	character string, one of 'lh', 'rh', or 'both'. The hemisphere to display. Defaults to "both".
views	list of strings, the view configuration. For single interactive view use <code>c("si")</code> . For 4-angle tiled view use <code>c("t4")</code> . For 9-angle tiled view use <code>c("t9")</code> . Defaults to <code>c("t4")</code> .
surface_style	character string, a rendering style for the cortical surface, e.g., 'default', 'shiny' or 'semitransparent'. Use 'semitransparent' to make the surface partially see-through, which works well for volume overlays. Defaults to "semitransparent".
rgloptions	named list, parameters passed to par3d . Example: <code>rgloptions = list("windowRect"=c(50,50,800,800))</code> . Defaults to the result of rglo .

makecmap_options	named list of parameters to pass to makecmap . Should include at least a colormap function as name 'colFn'. Defaults to the result of mkco.seq .
cortex_only	logical, whether to mask the medial wall (i.e., only render cortical vertices). Defaults to FALSE.
render_every	integer, for voxel mode only: render every Nth foreground voxel. Higher values improve performance but reduce density. Set to 1 to render all voxels (may be slow). Defaults to 20.
...	extra parameters passed to the volume rendering backend (e.g., lit=FALSE for unlit voxels).

Value

invisible named list with entries: "surface" (the coloredmeshes from the surface rendering, a hemilist of coloredmesh instances) and "volume" (the volume rendering result: a Triangles3D object for contour mode, or a list of coloredvoxels for voxel mode).

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vislayout.from.coloredmeshes\(\)](#)

Other volume visualization: [vis.volume.clusters\(\)](#), [volvis.lb\(\)](#), [volvis.lb.with.surface\(\)](#), [volvis.lightbox\(\)](#), [volvis.slices.with.surface\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
# Contour overlay of brain volume on thickness-colored surface:
vis.volume.on.surface(subjects_dir, 'subject1', 'brain',
  volume_mode='contour', volume_level=60, measure='thickness',
  views=c('si'), surface_style='semitransparent');
# Voxel overlay of aseg segmentation ventricles:
aseg = subject.volume(subjects_dir, 'subject1', 'aseg');
ventricle_mask = vol.mask.from.segmentation(aseg, c(4,14,15,43));
vis.volume.on.surface(subjects_dir, 'subject1', ventricle_mask,
  volume_mode='voxels', measure=NULL, views=c('si'),
  volume_color='red');

## End(Not run)
```

vislayout.from.coloredmeshes

Visualize coloredmeshes from several angles and combine the images into a new figure.

Description

Create a tight layout view of coloredmeshes from several angles. Creates separate `sd_<angle>` images, then crops and finally merges them into a single output image with image magick. The coloredmeshes to pass to this function are usually obtained by running any `vis*` function (like `vis.subject.morph.native`, `vis.subject.morph.standard`, `vis.subject.label`, `vis.subject.annot`, and others). That means you can use this function to visualize all kinds of data, e.g., morphometry data in native and standard space, labels, and brain atlases.

Usage

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

Arguments

- | | |
|---------------|--|
| coloredmeshes | list of coloredmesh. A coloredmesh is a named list as returned by the <code>coloredmesh.from*</code> functions (like <code>coloredmesh.from.morph.native</code>). It has the entries 'mesh' of type <code>tmesh3d</code> , a 'col', which is a color specification for such a mesh. The <code>vis*</code> functions (like <code>vis.subject.morph.native</code>) all return a list of coloredmeshes. |
| view_angles | list of strings. See <code>get.view.angle.names</code> for all valid strings. |
| rgloptions | option list passed to <code>par3d</code> . Example: <code>rgloptions = list("windowRect"=c(50,50,1000,1000))</code> . |
| 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 <code>material3d</code>), e.g., <code>list("shininess"=50, specular="black", alpha=0.5)</code> . Use the magic word 'from_mesh' to use the 'style' field of each coloredmesh instead of a single, global style. In that case, you will have to make sure your meshes have such a field, if not, the style 'default' is used as a fallback for those which don't. |

output_img	string, path to the output file. Defaults to "fsbrain_arranged.png"
silent	logical, whether to suppress all messages
grid_like	logical, whether to arrange the images in a grid-like fashion. If FALSE, they will all be merged horizontally. Passed to arrange.brainview.images .
background_color	hex color string (like '#FFFFFF'), the color to use for the background. Ignored if 'transparency_color' is not NULL. To get a transparent background, use 'transparency_color' instead of this parameter. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!
transparency_color	hex color string (like '#FFFFFF'), the temporary background color that will get mapped to transparency, or NULL if you do not want a transparent background. If used, it can be any color that does not occur in the foreground. Try '#FFFFFF' (white) or '#000000' (black) if in doubt. WARNING: Do not use color names (like 'gray'), as their interpretation differs between rgl and image magick!

Value

named list, see [arrange.brainview.images](#) for details

See Also

Other visualization functions: [highlight.vertices.on.subject\(\)](#), [highlight.vertices.on.subject.spheres\(\)](#), [vis.color.on.subject\(\)](#), [vis.data.on.fsaverage\(\)](#), [vis.data.on.subject\(\)](#), [vis.labeldata.on.subject\(\)](#), [vis.mask.on.subject\(\)](#), [vis.region.values.on.subject\(\)](#), [vis.rglwidget\(\)](#), [vis.subcortical.region.values\(\)](#), [vis.subject.annot\(\)](#), [vis.subject.label\(\)](#), [vis.subject.morph.native\(\)](#), [vis.subject.morph.standard\(\)](#), [vis.subject.pre\(\)](#), [vis.symmetric.data.on.subject\(\)](#), [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
# Use any vis function to get coloredmeshes. You can visualize morphometry,
# labels, an atlas, whatever. You can suppress the view unless you need it.
coloredmeshes = vis.subject.morph.native(subjects_dir, "subject1", "thickness",
  cortex_only=TRUE, rglactions=list("clip_data"=c(0.05, 0.95)),
  views=NULL);
# The meshes contain the surface, data, and color information and can be
# visualized. You could adapt the rendering style while doing so:
vislayout.from.coloredmeshes(coloredmeshes, style='shiny');
# You could change the rendering style on a per-mesh basis.
coloredmeshes[[1]]$style = list("shininess"=50, alpha=0.5);
vislayout.from.coloredmeshes(coloredmeshes, style='from_mesh');

## End(Not run)
```

vol.boundary.box	<i>Compute 3D bounding box of a volume.</i>
------------------	---

Description

Compute the axis-aligned foreground bounding box of a 3D volume, i.e. the inner foreground area that must be retained if you want to remove all background from the corners of the volume. The foreground is determined by thresholding, such that all values greater than 0 are considered foreground. See [vol.boundary.mask](#) for details.

Usage

```
vol.boundary.box(volume, threshold = 0L, apply = FALSE)
```

Arguments

volume	a 3D image volume
threshold	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value will be considered foreground voxels.
apply	logical, whether to directly apply the bounding box and return the resulting volume instead.

Value

named list with 2 entries: from is an integer vector of length 3, defining the minimal (x,y,z) foreground indices. to is an integer vector of length 3, defining the maximal (x,y,z) foreground indices.

See Also

Other volume utility: [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

vol.boundary.box.apply	<i>Apply a boundary box to a volume, returning the inner volume part</i>
------------------------	--

Description

Apply a boundary box to a volume, returning the inner volume part

Usage

```
vol.boundary.box.apply(volume, bbox)
```

Arguments

volume	a 3D image volume
bbox	the boundary box

Value

a 3D image volume, the inner volume part, resulting from the application of the boundary box

vol.hull	<i>Retain only the outer hull voxels of the foreground.</i>
----------	---

Description

Filters the *foreground* voxel in the volume by keeping only an outer border of voxels, and setting the inner core voxels to NA. This is a utility function for voxel-based visualization. The goal is to remove the inner voxels, which will not be visible anyways, and thus to dramatically reduce the number of triangles that will need to be computed for the mesh.

Usage

```
vol.hull(volume, thickness = 1L, axes = c(2L))
```

Arguments

volume	numeric 3d array, must contain foreground voxel and background voxels. The latter must have value NA. This function assumes that a solid foreground object surrounded by background exists in the volume.
thickness	integer, the width of the border in voxels, i.e., how many of the voxels in each upright column to keep at the top and at the bottom.
axes	integer vector, the axes to use. Valid values in the vector are 1L, 2L and 3L. You will have to use all 3 axes if you do not want any holes in the object. (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	<i>Turn volume into an ImageMagick image stack.</i>
----------------	---

Description

Create an image from each slice along the axis, then stack those into an ImageMagick image stack.

Usage

```
vol.imagestack(volume, axis = 1L, intensity_scale = 255)
```

Arguments

volume	a 3D image volume. Can be numeric, or something that can be read directly by <code>magick::image_read</code> 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 <code>NULL</code> for data that can be read directly by <code>magick::image_read</code> , and to 1 for intensity data that requires no scaling. Defaults to 255, which is suitable for 8 bit image data.

Value

a vectorized ImageMagick image, containing one subimage per slice. This can be interpreted as an animation or whatever.

See Also

Other volume utility: [vol.boundary.box\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

vol.intensity.to.color	<i>Convert integer intensity image to RGB color string form.</i>
------------------------	--

Description

Convert a gray-scale image defined by intensity values in range $[0, 1]$ to an image with identical dimensions that contains an R color string (like #222222) at each position. The color strings are computed from the intensities, by taking the intensity value as the value for all three RGB channels. I.e., the output is still gray-scale, but defined in RGB space. To make it clear, this function does **not** apply a colormap. It only changes the representation of the data, not the resulting colors.

Usage

```
vol.intensity.to.color(volume, scale = NULL)
```

Arguments

volume	numeric array, typically a 3D image with intensities in range $[0, 1]$. This function now also supports numeric matrices (2D images, slices) and numeric vectors (1D).
scale	numeric or character string, a scaling to apply to the values. Defaults to NULL, which means <i>no scaling</i> 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 <i>outside</i> the range $[0, 1]$, so that data in range $[0.3, 0.5]$ would not be rescaled to $[0, 1]$.

Value

array (or matrix, or vector) of RGB color strings. All of them will represent gray values.

Examples

```
vol.intensity.to.color(c(0.0, 0.5, 1.0));
# output: "#000000" "#808080" "#FFFFFF"
vol.intensity.to.color(c(20, 186, 240), scale="normalize");
vol.intensity.to.color(c(20, 186, 240), scale=255);
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize");
vol.intensity.to.color(c(0.0, 0.5, 0.8), scale="normalize_if_needed");
```

```
vol.mask.from.segmentation
```

Extract subset from a volume by value.

Description

Extract subset from a volume by value, set all other voxel values to NA. Typically used to extract a brain structure (like corpus callosum) from a volume segmentation (like the `mri/aseg.mgz` file of a subject). You should consider passing the volume and the include values as integers.

Usage

```
vol.mask.from.segmentation(volume, include_values)
```

Arguments

volume	numeric 3D array
include_values	numerical vector, the intensity values which qualify a voxel to be part of the result (without being set to NA)

Value

numerical array with same dimensions as the input volume. All values which are not part of `include_values` replaced with NA.

vol.merge	<i>Merge background volume and overlay to new colors.</i>
-----------	---

Description

Merge background volume and overlay to new colors.

Usage

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

Arguments

volume	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range $[0, 1]$.
overlay_colors	3D array of color strings, values which are not part of the overlay (and should display background in the result) must have NA instead of a color string. Must have same dimensions as the volume.
bbox_threshold	numerical, the threshold intensity used to separate background and foreground. All voxels with intensity values greater than this value in the background volume will be considered foreground voxels. Background-only slices at the borders of the volume will be discarded (in the merged, final image). Pass NULL to use the full image without applying any bounding box.
forced_overlay_color	NULL or an rgb color string, like '#FF0000' for red. If NULL, the activation colors will be used as foreground colors. Otherwise, the given color will be for all foreground vertices.

Value

3D array of color strings, the merged colors

See Also

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.overlay.colors.from.activation\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

```
vol.overlay.colors.from.activation
```

Generate colors for a 3D volume, based on the activation data and a colormap.

Description

Applies the colormap function to the data, then sets the alpha value (transparency) to full in all areas without any activation. Feel free to clip data or whatever before passing it, so that all your no-activation data has the same value.

Usage

```
vol.overlay.colors.from.activation(  
    volume,  
    colormap_fn = squash::blueorange,  
    no_act_source_value = 0  
)
```

Arguments

volume	a 3D array, the activation data (or p-values, effect sizes, or whatever)
colormap_fn	function, a colormap function
no_act_source_value	numerical scalar, the value from the data in 'volume' that means no activation. The output colors for this value will be set to NA. Set to NULL to not change anything.

Value

a 3D matrix of color strings, with the same dimensions as the input volume

See Also

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.planes\(\)](#), [vol.slice\(\)](#)

```
vol.overlay.colors.from.colortable
```

Compute voxel colors based on colortable.

Description

Use the intensity values of the voxels in volume and lookup the respective colors in a colortable.

Usage

```
vol.overlay.colors.from.colortable(
  volume,
  colortable,
  ignored_struct_indices = c(),
  ignored_struct_names = c("unknown", "Unknown")
)
```

Arguments

volume	numeric 3D array, the values should be integers present in the struct_index column of the colortable. All other values will be assigned NA as a color.
colortable	a colortable, as returned by read.fs.colortable , or a character string representing a path to a colortable file.
ignored_struct_indices	integer vector, struct_index entries in the colortable that should be ignored
ignored_struct_names	vector of character strings, struct_name entries in the colortable that should be ignored. Can be combined with ignored_struct_indices.

Value

character string 3D array, the colors. Voxels in the volume which were not matched by the colortable are set to NA in it.

```
vol.planes
```

Translate names and indices of planes.

Description

Translate names and indices of 3D image planes. The names only make sense if the data in the volume is in the default FreeSurfer conformed orientation.

Usage

```
vol.planes(plane = NULL)
```

Arguments

plane NULL, a plane index, or a plane name.

Value

if plane is NULL, all available planes and their indices as a named list. If plane is an integer (a plane index), its name. If plane is an characters string (a plane name), its index.

See Also

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagystack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation](#)
[vol.slice\(\)](#)

vol.slice	<i>Extract a slice of a 3D image stack.</i>
-----------	---

Description

Extracts one or more 2D slices from a 3D image (or a frame of a 4D image). To display the result, you can use [volvis.lightbox](#).

Usage

```
vol.slice(  
  volume,  
  slice_index = NULL,  
  frame = 1L,  
  axis = 1L,  
  rotation = 0L,  
  flip = NULL  
)
```

Arguments

volume	a 3D or 4D image volume. Note that empty dimensions will be dropped before any processing, and the remaining volume must have 3 or 4 dimensions.
slice_index	positive integer or vector of positive integers, the index into the slices (for the axis). A <i>slice</i> in the sense of this function is any 2D image plane extracted from the 3D volume (no matter the axis). If NULL, the slice in the middle of the volume is used. One can pass the magic character string 'all' to use all slice indices along the axis.
frame	positive integer, optional. The frame (time point) to use, only relevant for 4D volumes. The last (i.e. 4th) dimension is assumed to be the time dimension in that case.
axis	positive integer, the axis to use when indexing the slices. Defaults to 1.

rotation	integer, rotation in degrees. Defaults to 0 (no rotation). Must be a multiple of 90L if given.
flip	NULL or one of the character strings 'vertically' or 'horizontally'. Note that flipping <i>horizontally</i> means that the image will be mirrored along the central <i>vertical</i> axis. If NULL is passed, nothing is flipped. Flipping occurs after rotation.

Value

slice data. If slice_index is a scalar, a numerical 2D matrix (a 2D image from the stack). Otherwise, a numerical 3D array that contains the selected 2D images.

See Also

Other volume utility: [vol.boundary.box\(\)](#), [vol.imagestack\(\)](#), [vol.merge\(\)](#), [vol.overlay.colors.from.activation](#) [vol.planes\(\)](#)

vol.vox.from.crs	<i>Compute R voxel index for FreeSurfer CRS voxel index.</i>
------------------	--

Description

Performs a vox2vos transform from FreeSurfer to R indices.

Usage

```
vol.vox.from.crs(fs_crs, add_affine = FALSE)
```

Arguments

fs_crs	integer vector of length 3, Freesurfer indices for column, row, and slice (CRS).
add_affine	logical, whether to add 1 to the output vector as the 4th value

Value

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

Examples

```
# Get voxel intensity data on the command line, based
# on the FreeSurfer (zero-based) CRS voxel indices:
# `mri_info --voxel 127 100 100 ~/data/tim_only/tim/mri/brain.mgz`
# (the result is: 106.0)
#
# That should be identical to:
# our_crs = vol.vox.from.crs(c(127,100,100), add_affine = FALSE);
# brain$data[our_crs[1], our_crs[2], our_crs[3]]; # gives 106
```

volvis.contour	<i>Visualize contour of a volume.</i>
----------------	---------------------------------------

Description

Compute a smoothed surface from the voxel intensities in the given volume and render it. Requires the misc3d package to be installed, which is an optional dependency.

Usage

```
volvis.contour(volume, level = 80, show = TRUE, frame = 1L, color = "white")
```

Arguments

volume	a 3D brain volume
level	numeric, intensity threshold for the data. Voxels with intensity value smaller than level will be ignored when creating the contour surface.
show	logical, whether to display the triangles. Defaults to TRUE.
frame	integer, the frame to show in case of a 4D input volume. Can also be the character string 'all' to draw the contents of all frames at once. Useful to plot white matter tracts from DTI data, where each tract is stored in a different frame.
color	the color to use when plotting. Can be a vector of colors when plotting all frames of a 4D image (one color per frame).

Value

the rendered triangles (a Triangles3D instance) with coordinates in surface RAS space if any, NULL otherwise. This will be a list if you pass a 4D volume and select 'all' frames.

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain');
# Plot all voxels of the brain:
volvis.contour(brain);

## End(Not run)
```

volvis.lb	<i>Show continuous 3D voxel/volume data as a lightbox, optionally with a background brain volume and colormap.</i>
-----------	--

Description

This function is the main way to visualize 3D volume images that contain raw MRI scans or statistical results.

Usage

```
volvis.lb(
    volume,
    background = NULL,
    colFn = viridis::viridis,
    colortable = NULL,
    no_act_source_value = 0,
    bbox_threshold = NULL,
    bbox_of_volume = TRUE,
    ...
)
```

Arguments

volume	numerical 3D array of per-voxel data, typically activation data, a raw MRI image, or a segmentation to show. Can also be a filename if the file can be loaded as such a volume with read.fs.volume .
background	numerical 3D array or 3D array of color strings, the background volume. Typically a raw brain volume. Dimensions and space must match those of the 'volume' for an array. Can also be a single file name as a character string. Can also be a single color name, like '#FEFEFE' but the string then must start with '#' (color names like 'red' are not allowed, they would be treated as file names). If a color string, be sure to use the ... parameter to set the same color as background_color for the tiles.
colFn	a colormap function, passed to <code>vol.overlay.colors.from.activation</code> and used as colormap for the 'volume' data. Pass NULL to derive gray-scale values from the raw data (only recommended with single-color backgrounds). Note that the colormap is not used for the the background data (if any), which will be shown in grayscale (unless it is a 3D array of color strings).
colortable	optional, only makes sense for categorical 'volume' data like segmentations. If not NULL, a colortable as returned by read.fs.colortable , or a character string representing a path to a colortable file (like "FREESURFER_HOME/FreeSurferColorLUT.txt").
no_act_source_value	numerical value, passed to <code>vol.overlay.colors.from.activation</code> . Specifies the value which is treated as transparent in the 'volume' parameter data (where you will see the background). If you need more control, e.g., you want to treat

one or more ranges of values as NA, you should load the 'volume' data first, modify it as needed, as pass it to this function afterwards. Set this parameter to NULL to disable it. Only for 'colFn', ignored if a 'colortable' is used.

- `bbox_threshold` numerical scalar, passed on to `vol.merge`. If set, voxels with intensities smaller than this threshold will be dropped at the outside of the image. If `bbox_of_volume` parameter is TRUE (the default), this threshold applies to the 'volume', otherwise to the 'background'. Set to NULL to disable bounding box and show the full image.
- `bbox_of_volume` logical, whether the bounding box is computed on the volume (foreground), which typically is what you want. Leave alone if in doubt.
- `...` extra parameters to be passed to `volvis.lightbox`, can be used to select specific slices, set the `background_color` for the border between and around the image tiles, etc.

Note

This function should be preferred over manually calling `volvis.lightbox`.

See Also

Other volume visualization: `vis.volume.clusters()`, `vis.volume.on.surface()`, `volvis.lb.with.surface()`, `volvis.lightbox()`, `volvis.slices.with.surface()`

Examples

```
## Not run:
volume = subject.volume(subjects_dir, subject_id, 'brain');
volvis.lb(volume);
volvis.lb("~/study1/subject1/mri/brain.mgz");
volvis.lb("~/study1/subject1/mri/brain.mgz", bbox_threshold = 1L);
volvis.lb("~/study1/subject1/mri/brain.mgz", background = "~/data/study1/subject1/mri/T1.mgz");
volvis.lb("~/study1/subject1/mri/brain.mgz", background = "#FEFEFE", background_color="#FEFEFE");
ct = file.path(find.freesurferhome(mustWork = T), "FreeSurferColorLUT.txt"); # ct = "color table"
volvis.lb("~/study1/subject1/mri/aseg.mgz", background="~/study1/subject1/mri/T1.mgz",
  colortable = ct, colFn=NULL, axis=2L);
volvis.lb("~/study1/subject1/mri/aseg.mgz", background = "~/study1/subject1/mri/T1.mgz",
  colortable = ct, colFn=NULL, bbox_threshold = 0);

## End(Not run)
```

volvis.lb.with.surface

Visualize volume slices with surface mesh contours overlaid in light-box view.

Description

Creates a lightbox view of 2D MRI slices with the cortical surface boundary contours drawn on top. This is the fsbrain equivalent of what FreeSurfer's freeview or tkmedit show — useful for quality assessment of surface reconstruction against the underlying MRI volume. The surface mesh is loaded from the subject's FreeSurfer directory and its intersection with each displayed slice plane is computed and drawn as colored contour lines.

Usage

```
volvis.lb.with.surface(
    subjects_dir,
    subject_id,
    volume = "brain",
    surface = "white",
    hemi = "both",
    surface_color = "#FF0000",
    surface_lwd = 1,
    slices = -5,
    axis = 1L,
    scale = 1L,
    per_row = 5L,
    per_col = NULL,
    border_geometry = "5x5",
    background_color = "#000000",
    silent = TRUE
)
```

Arguments

subjects_dir	character string, the FreeSurfer SUBJECTS_DIR.
subject_id	character string, the subject identifier.
volume	numeric 3D array or character string. Either a 3D brain volume (e.g., from T1.mgz or brain.mgz), or the name of a volume file to load from the subject's mri/ directory. Defaults to "brain".
surface	character string or vector of strings, the surface(s) to use for contour extraction. One or more of "white", "pial", or "inflated". If a vector, contours for all surfaces are drawn. Defaults to "white".
hemi	character string, one of 'lh', 'rh', or 'both'. Which hemisphere surface(s) to overlay. Defaults to "both".
surface_color	character string or character vector, the color(s) for the surface contour lines. If a single color, all surfaces and hemispheres use it. If a vector of length length(surface), each surface gets its own color (both hemispheres use it). If a vector of length 2 * length(surface), colors are assigned as (lh_surf1, rh_surf1, lh_surf2, rh_surf2, ...). Defaults to "#FF0000" (red).
surface_lwd	numeric, line width for the contour lines (passed as lwd to segments). Defaults to 1.

slices	passed to volvis.lightbox . A negative integer N means "use every Nth slice". A numeric vector gives explicit slice indices (1-based). Defaults to -5 (every 5th slice).
axis	integer, the slice axis. 1 = sagittal, 2 = coronal, 3 = axial (in volume CRS convention). Defaults to 1L.
scale	integer >= 1, the factor by which to upscale the slice images in both dimensions using nearest-neighbor interpolation (keeps the MRI pixels sharp, which is good for QA). The surface contour lines are drawn on a high-resolution overlay and downsampled to the final size, so they appear thin and smooth. Defaults to 1L (no upscaling, previous behavior).
per_row	integer, number of slice images per row in the lightbox grid. Defaults to 5L.
per_col	integer, number of rows. If NULL, computed from per_row. Defaults to NULL.
border_geometry	character string, geometry for borders between tiles, passed to image_border . Defaults to "5x5".
background_color	character string, background color for borders and empty tiles. Defaults to "#000000".
silent	logical, whether to suppress messages. Defaults to TRUE.

Value

a magick image instance containing the lightbox with surface contours overlaid. Can be saved with [image_write](#) or displayed interactively.

See Also

Other volume visualization: [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [volvis.lb\(\)](#), [volvis.lightbox\(\)](#), [volvis.slices.with.surface\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir <- fsbrain::get_optional_data_filepath("subjects_dir");

# Axial slices with white surface contours in red:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
  volume="brain", surface="white", axis=3L,
  surface_color="#FF0000");
magick::image_write(img, "~/fsbrain-qa_axial.png");

# Coronal slices in green:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
  volume="brain", surface="white", axis=2L,
  surface_color="#00FF00");

# Sagittal slices: left hemisphere red, right hemisphere blue:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
```

```

    volume="brain", surface="white", axis=1L,
    surface_color=c("#FF0000", "#0000FF"));

# Single hemisphere, pial surface, every 5th slice:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
    volume="brain", surface="pial", hemi="lh",
    axis=3L, slices=-5, surface_color="#FF8800");

# Both white and pial surfaces overlaid: white in red, pial in yellow:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
    volume="brain", surface=c("white", "pial"), axis=3L,
    surface_color=c("#FF0000", "#FFFF00"));

# White (lh red, rh blue) + pial (lh green, rh orange):
img <- volvis.lb.with.surface(subjects_dir, "subject1",
    volume="brain", surface=c("white", "pial"), axis=3L,
    surface_color=c("#FF0000", "#0000FF", "#00FF00", "#FF8800"));

# 2x upscaled slices for larger images with thin, smooth contours:
img <- volvis.lb.with.surface(subjects_dir, "subject1",
    volume="brain", surface="white", axis=3L, scale=2L);

## End(Not run)

```

volvis.lightbox

Draw a lightbox view from volume slices.

Description

A lightbox is a single image that holds a set of subimages, arranged in a grid. The images can have a small border or spacing between them. Consecutive subimages will be appear the same row of the grid.

If overlay_colors are given, the volume will be used as the background, and it will only be visible where overlay_colors has transparency.

Usage

```

volvis.lightbox(
  volume,
  slices = -5,
  axis = 1L,
  per_row = 5L,
  per_col = NULL,
  border_geometry = "5x5",
  background_color = "#000000",
  arrange_single_image = FALSE,
  scale = 1L
)

```

Arguments

volume	3D array, can be numeric (gray-scale intensity values) or color strings. If numeric, the intensity values must be in range $[0, 1]$.
slices	slice index definition. If a vector of integers, interpreted as slice indices. If a single negative interger $-n$, interpreted as every n th slice, starting at slice 1. The character string 'all' or the value NULL will be interpreted as <i>all slices</i> .
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 <code>magick::image_border</code> to define the borders to add to each image tile. The default value adds 5 pixels, both horizontally and vertically.
background_color	string, a valid ImageMagick color string such as "white" or "#000080". The color to use when extending images (e.g., when creating the border). Defaults to black.
arrange_single_image	logical, whether to apply the given arrangement (from parameters per_row and per_column) even if a single slice (a 2D image) is passed as volume. Defaults to FALSE, which prevents that background tiles are added to fill the row up to per_row images. This also prevents the border from getting added to a single image, so all you see is the raw image. Set to TRUE if you want to arrange even a single image in a row with a border.
scale	integer ≥ 1 , the factor by which to upscale the slice images in both dimensions using nearest-neighbor interpolation. This keeps the MRI pixels sharp (no smoothing) while making the images larger. Defaults to 1L (no upscaling).

Value

a magick image instance

Note

You should, in most cases, not call this function directly. Use `volvis.lb` instead, which has a more intuitive interface.

See Also

`volvis.lb`

Other volume visualization: `vis.volume.clusters()`, `vis.volume.on.surface()`, `volvis.lb()`, `volvis.lb.with.surface()`, `volvis.slices.with.surface()`

volvis.shells

*Visualize a volume as nested, semi-transparent iso-surface shells.***Description**

Visualize the iso-surfaces (*shells*) of a volume at several levels as nested, semi-transparent meshes, like the nested contour lines of a topographic map. This is useful to show the shape and the internal structure of a volume at the same time, e.g., for a statistical map: the outer shells (low levels) are rendered more transparently than the inner ones (high levels), so that the inner structure remains visible. The volume is extracted in voxel space and transformed to surface RAS with the [index2ras_tkr](#) matrix, so the result is spatially aligned with surface renderings of the same subject (see [vis.subject.morph.native](#)). To combine the shells with a surface rendering, use the `rglactions` parameter of the surface function (key 'no_vis') to keep the rendering window open, or call this function with `views = NULL` and render the returned meshes yourself.

Usage

```
volvis.shells(
  volume,
  levels = NULL,
  num_levels = 4L,
  level_type = "quantile",
  level_range = c(0.2, 0.95),
  frame = 1L,
  colors = NULL,
  alphas = NULL,
  palette = "grey_context",
  alpha_range = NULL,
  smoothing = 1L,
  downsample = 1L,
  cut_away = NULL,
  cut_fraction = 0.5,
  backend = "auto",
  views = c("sd_lateral_lh", "sd_medial_lh", "sd_lateral_rh", "sd_medial_rh"),
  rgloptions = rglo(),
  rglactions = list(),
  silent = FALSE
)
```

Arguments

<code>volume</code>	a 3D numerical array (or an <code>fs.volume</code> instance), the volume to visualize. Values 0 and NA are treated as background when the levels are computed automatically, and the shells are the iso-surfaces at the computed (or given) levels.
<code>levels</code>	numerical vector or NULL. The iso-levels at which to extract shells, in the units of the volume data. Must be ascending (the first level is the outermost shell). If

	NULL (the default), the levels are computed automatically, see <code>num_levels</code> and <code>level_type</code> .
<code>num_levels</code>	positive integer, the number of shells to create. Ignored if <code>levels</code> is given. Defaults to 5.
<code>level_type</code>	character string, how to compute the levels if <code>levels</code> is NULL. One of 'quantile' (the default) or 'linear'. For 'quantile', each shell contains the same fraction of the foreground voxels, for 'linear', the levels are equally spaced within the range given by <code>level_range</code> . In both cases only voxels with finite, non-zero values are considered.
<code>level_range</code>	numerical vector of length 2, the range from which the automatic levels are taken. For <code>level_type</code> = 'quantile', this is interpreted as quantiles (values between 0 and 1) of the foreground values, for <code>level_type</code> = 'linear' as fractions of the range of the foreground values. Defaults to <code>c(0.2, 0.95)</code> , i.e., the innermost shell is at the 95th percentile of the voxel values, so that a few extreme voxels do not dominate the visualization.
<code>frame</code>	positive integer, the frame (4th dimension) to use for a 4D volume. Defaults to 1.
<code>colors</code>	character vector of colors, one per shell (outermost shell first). Defaults to NULL, in which case the colors are taken from the palette.
<code>alphas</code>	numerical vector of alpha values, one per shell (outermost shell first), values between 0 and 1. Defaults to NULL, in which case the alpha values are taken from the palette.
<code>palette</code>	character string, the color and transparency recipe used for the shells if <code>colors</code> and/or <code>alphas</code> are not given, one of 'grey_context' (the default), 'sequential' or 'viridis'. See shell.palette for the details.
<code>alpha_range</code>	numerical vector of length 2 or NULL, the alpha value of the outermost and of the innermost shell. If NULL (the default), the range defined by the palette is used. Ignored if <code>alphas</code> is given.
<code>smoothing</code>	non-negative integer, the number of 3x3x3 box blur passes applied to the volume before the shells are extracted. This removes the staircase artifacts of the marching cubes algorithm and leads to much nicer surfaces. Defaults to 1. Set to 0 to disable and extract the shells from the raw volume.
<code>downsample</code>	positive integer, a factor by which the volume is subsampled before the shells are extracted. Values larger than 1 reduce the number of triangles (and thus the rendering time) considerably, at the cost of a less detailed surface. The coordinates of the subsampled shells are corrected, so the result is still aligned with the original volume. Defaults to 1. Note that the volume is smoothed before subsampling if <code>smoothing</code> is larger than 0, otherwise the shells are prone to aliasing artifacts.
<code>cut_away</code>	character string or NULL. If not NULL, the volume is cut open along one of the six anatomical directions, so that one can look inside: the part of the volume in the given direction is removed. Valid values are 'left', 'right', 'posterior', 'anterior', 'inferior' and 'superior' (in FreeSurfer RAS convention, i.e., +x is right, +y is anterior, +z is superior). The caps of the cut are not closed, so the shells are open at the cut plane. Defaults to NULL, i.e., no cut.

cut_fraction	numerical value between 0 and 1, the position of the cut along the axis given by cut_away, as a fraction of the bounding box of the shells. Defaults to 0.5, i.e., the volume is cut in the middle.
backend	character string, the backend used to extract the iso-surfaces. One of 'auto' (the default: use the Rvcg package if it is installed, and fall back to misc3d otherwise), 'Rvcg' or 'misc3d'. Rvcg is the recommended backend: it is faster and returns vertex-welded meshes (i.e., meshes that need much less memory) with normals. Note that at least one of the two optional packages must be installed.
views	list of character strings, the views to visualize. Available views are 'sd_<angle>' (a static view, see vis.subject.morph.native), 't4'/'t9' (a 2x2 or 3x3 light-box of views) and 'si'/'sr' (single interactive / rotating, requires the rgl renderer backend). Defaults to the 4 standard lateral and medial views. Pass NULL to skip the rendering entirely and only compute the meshes (which are also returned).
rgloptions	option list passed to par3d . Defaults to the package default, see rglo .
rglactions	named list, passed to the visualization functions, see rglactions . The key 'no_vis' is useful to prepare the scene and add other data to it later. Note that saving the rendered scene to an image file is done with export , which takes the return value of this function and supports merging several views into one image.
silent	logical, whether to suppress the console messages which report the levels and the backend used. Defaults to FALSE.

Value

the list of `fs.coloredmesh` instances, one per shell (from the outermost to the innermost), invisibly. Note that the colorbar of the rendering functions is not used here, the color of a shell encodes its level, which is reported in the console unless `silent` is TRUE.

Note

Both renderers are used with the mesh's own style (see [get.rglstyle](#)), which sets the alpha value of the shell. Note that backfaces are rendered for the shells (`back = 'filled'`): if they were culled, the rgl renderer would composite the nested transparent shells in the wrong order (the innermost shell would end up on top of the outer ones).

See Also

[volvis.contour](#) for a single iso-surface, [volvis.voxels](#) for a voxel-based rendering of a volume, and [vis.volume.on.surface](#) to combine a volume with a surface rendering.

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain');
# Show the 5 inner shells of the brain, cut open from the right:
shells = volvis.shells(brain, num_levels = 5, downsample = 2, cut_away = 'right', views = NULL);
# Combine the 4 standard views into a single image:
view_angles = c('sd_lateral_lh', 'sd_medial_lh', 'sd_lateral_rh', 'sd_medial_rh');
```

```
fsbrain::export(shells, view_angles = view_angles, output_img = 'shells.png');

## End(Not run)
```

volvis.slices.with.surface

Export individual volume slices with surface contours to image files.

Description

Creates individual 2D slice images with surface contour overlays and exports them to separate files — ideal for browsing through slices one by one. This is the single-slice export counterpart of [volvis.lb.with.surface](#), which arranges slice images in a lightbox grid. Supports optional slice index labelling and automatic skipping of empty slices (i.e., slices where no surface contour intersects the plane).

Usage

```
volvis.slices.with.surface(
  subjects_dir,
  subject_id,
  volume = "brain",
  surface = "white",
  hemi = "both",
  surface_color = "#FF0000",
  surface_lwd = 1,
  slices = -5,
  axis = 1L,
  scale = 1L,
  silent = TRUE,
  output_dir = ".",
  output_prefix = NULL,
  image_format = "png",
  label_slices = FALSE,
  label_color = "white",
  label_size = 20,
  label_gravity = "northwest",
  skip_empty = FALSE
)
```

Arguments

<code>subjects_dir</code>	character string, the FreeSurfer SUBJECTS_DIR.
<code>subject_id</code>	character string, the subject identifier.
<code>volume</code>	numeric 3D array or character string. Either a 3D brain volume, or the name of a volume file to load from the subject's <code>mri/</code> directory. Defaults to "brain".

surface	character string or vector of strings, the surface(s) to use for contour extraction. One or more of "white", "pial", or "inflated". Defaults to "white".
hemi	character string, one of 'lh', 'rh', or 'both'. Which hemisphere surface(s) to overlay. Defaults to "both".
surface_color	character string or character vector, the color(s) for the surface contour lines. See volvis.lb.with.surface for the color assignment rules. Defaults to "#FF0000" (red).
surface_lwd	numeric, line width for the contour lines. Defaults to 1.
slices	passed to volvis.lightbox . A negative integer N means "use every Nth slice". A numeric vector gives explicit slice indices (1-based). Defaults to -5.
axis	integer, the slice axis. 1 = sagittal, 2 = coronal, 3 = axial (in volume CRS convention). Defaults to 1L.
scale	integer >= 1, the factor by which to upscale the slice images in both dimensions using nearest-neighbor interpolation (keeps the MRI pixels sharp, which is good for QA). The surface contour lines are drawn on a high-resolution overlay and downsampled to the final size, so they appear thin and smooth. Defaults to 1L (no upscaling, previous behavior).
silent	logical, whether to suppress messages. Defaults to TRUE.
output_dir	character string, directory in which to write the slice image files. Created if it does not exist. Defaults to "." (current working directory).
output_prefix	character string, prefix for output filenames. If NULL, auto-derived as "<subject_id>". Defaults to NULL.
image_format	character string, image format for output files (e.g., "png", "jpg"). Passed to image_write . Defaults to "png".
label_slices	logical, whether to annotate each slice image with its slice index (e.g., "slice 42") in the top-left corner. Uses image_annotate . Defaults to FALSE.
label_color	character string, color for the slice label text. Defaults to "white".
label_size	integer, font size for the slice label. Defaults to 20.
label_gravity	character string, gravity for label placement, passed to image_annotate . Defaults to "northwest".
skip_empty	logical, whether to skip slices where no surface contour intersects the slice plane. Defaults to FALSE.

Value

invisible character vector of file paths that were written.

See Also

Other volume visualization: [vis.volume.clusters\(\)](#), [vis.volume.on.surface\(\)](#), [volvis.lb\(\)](#), [volvis.lb.with.surface\(\)](#), [volvis.lightbox\(\)](#)

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir <- fsbrain::get_optional_data_filepath("subjects_dir");

# Export all sagittal slices with white surface contours:
volvis.slices.with.surface(subjects_dir, "subject1",
  volume="brain", surface="white", axis=1L,
  output_dir=~ /qa_slices");

# Export with slice labels, skipping empty slices:
volvis.slices.with.surface(subjects_dir, "subject1",
  volume="brain", surface="white", axis=3L,
  output_dir=~ /qa_slices", label_slices=TRUE,
  skip_empty=TRUE);

# Both white and pial surfaces, different colors:
volvis.slices.with.surface(subjects_dir, "subject1",
  volume="brain", surface=c("white","pial"), axis=3L,
  surface_color=c("#FF0000","#FFFF00"),
  output_dir=~ /qa_slices");

# 2x upscaled slices for larger images with thin, smooth contours:
volvis.slices.with.surface(subjects_dir, "subject1",
  volume="brain", surface="white", axis=3L, scale=2L,
  output_dir=~ /qa_slices");

## End(Not run)
```

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 1-based R array indices of the voxels using the [index2ras_tkr](#) matrix (see [vox2ras_tkr](#) for the CRS-based variant). This means that the position of the rendered data fits to the surface coordinates (in files like surf/1h.white), and that you can call this function while an active surface rendering window is open (e.g., from calling [vis.subject.morph.native](#)), to superimpose the surface and volume data. **On coloring the voxels** (using *rgl materials*): Note that you can call this function several times for the active plot, and color the voxels differently by passing different material properties in each call. Alternatively, check the `voxelcol` parameter.

Usage

```
volvis.voxels(volume, render_every = 1, voxelcol = NULL, ...)
```

Arguments

volume	numeric 3d array, voxels which should not be plotted must have value NA. Take care not to plot too many.
render_every	integer, how many to skip before rendering the next one (to improve performance and/or see deeper structures). Use higher values to see a less dense representation of your data that usually still allows you to see the general shape, but at lower computational burden. Set to 1 to render every (foreground) voxel.
voxelcol	character string or a <i>voxel coloring</i> . A <i>voxel coloring</i> 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 <code>triangles3d</code> . Example: <code>color = "#0000ff"</code> , <code>lit=FALSE</code> .

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
brain = subject.volume(subjects_dir, 'subject1', 'brain');
# Plot all voxels of the brain:
brain[which(brain==0L, arr.ind = TRUE)] = NA; # mark background
brain = vol.hull(brain); # remove inner triangles
volvis.voxels(brain);

## End(Not run)
```

vox2ras_tkr

*The FreeSurfer default vox2ras_tkr matrix.***Description**

Applying this matrix to a FreeSurfer CRS index of a conformed volume will give you the RAS coordinates of the voxel in surface coordinates, i.e., in the coordinates used in surface file like `lh.white`. The central voxel is 127,127,127 when using zero-based indices (or 128,128,128 when using one-based indices), meaning its surface RAS coordinates are 0.0, 0.0, 0.0. The returned matrix is the inverse of the `ras2vox_tkr` matrix.

Usage

```
vox2ras_tkr()
```

Value

numeric 4x4 matrix, the FreeSurfer vox2ras_tkr matrix.

See Also

Other surface and volume coordinates: [index2ras_tkr\(\)](#), [ras2vox_tkr\(\)](#)

Examples

```
# Compute surface RAS coordinate of voxel with CRS (0L, 0L, 0L):
vox2ras_tkr() %*% c(0, 0, 0, 1);
# Show that voxel with CRS (128,128,128) is at the
# origin (0.0, 0.0, 0.0) of the surface RAS coordinate system:
(vox2ras_tkr() %*% c(128, 128, 128, 1))[1:3];
```

```
write.group.morph.standard
```

Write standard space group data to a standard FreeSurfer directory structure.

Description

Write standard space group data to a standard FreeSurfer directory structure.

Usage

```
write.group.morph.standard(
  subjects_dir,
  subjects_list,
  data,
  measure_name,
  hemi = "both",
  fwhm = "10",
  template_subject = "fsaverage",
  format = "mgh",
  create_dirs = TRUE,
  template_lh_numverts = NULL
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	vector of strings. The subject identifiers.
data	the data matrix

measure_name	character string, the data part of the generated file names, e.g., 'thickness' or 'area'.
hemi	string, one of 'lh', 'rh' or 'both'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
fwhm	string. Smoothing as string, e.g. '10' or '25'.
template_subject	string. Template subject name, defaults to 'fsaverage'.
format	string. One of 'mgh', 'mgz', 'curv'. Defaults to 'mgh'.
create_dirs	logical, whether to create missing (sub) directories which occur in the 'filepaths'.
template_lh_numverts	positive integer, the vertex count of the left hemi of the template subject, only used if 'hemi' is 'both'. If hemi is both and this is unspecified (left at the default value NULL), the template subject needs to exist in the 'subjects_dir' to determine the vertex count of the left hemisphere, so that the data can be split into the lh and rh files at the correct index.

See Also

[write.group.morph.standard.sf](#) and [write.group.morph.standard.mf](#)

Examples

```
## Not run:
dm = matrix(rnorm(325684 * 6, 5.0, 1.2), ncol = 6);
subjects = paste("subject", seq(6), sep="");
write.group.morph.standard("/tmp/groupdata", subjects, dm,
  "rand", template_lh_numverts = 325684 / 2);

## End(Not run)
```

```
write.group.morph.standard.mf
```

Write per-vertex standard space data for a group of subjects to given file names.

Description

Write per-vertex standard space data for a group of subjects to given file names.

Usage

```
write.group.morph.standard.mf(
  filepaths_hl,
  data_hl,
  format = "auto",
  create_dirs = TRUE
)
```

Arguments

filepaths_hl	hemilist of vectors of character strings, the full paths to the output files, including file names and extension.
data_hl	hemilist of numerical matrix or data.frame, the morph data for the hemispheres of all subjects. See <code>groupmorph.split.hemilist</code> to get this format if you have a full matrix or dataframe for both hemispheres.
format	character string, a valid format spec for <code>freesurferformats::write.fs.morph</code> , e.g., "auto" to derive from filename, "mgh", "mgz", "curv" or others.
create_dirs	logical, whether to create missing (sub) directories which occur in the 'filepaths'.

See Also

[write.group.morph.standard.sf](#) to write the data to a single stacked file instead.

`write.group.morph.standard.sf`

Reshape and write combined per-vertex data for a group to a single MGH file.

Description

Write morphometry data for a group into a single MGH or MGZ file. In neuroimaging, the first 3 dimensions in the resulting 4D volume file are space, and the 4th is the time/subject dimension.

Usage

```
write.group.morph.standard.sf(filepath, data)
```

Arguments

filepath	character string, path to the target file, should end with '.mgh' or '.mgz'.
data	numerical 2D matrix, with the rows identifying the subjects and the columns identifying the vertices.

Note

The file will contain no information on the subject identifiers. The data can be for one or both hemispheres. See [group.morph.standard.sf](#) to read the data back into R.

Examples

```
## Not run:
# create per-vertex data for 5 subjects.
mat = matrix(rnorm(5 * 163842, 3.0, 0.5), nrow=5, ncol = 163842);
fsbrain::write.group.morph.standard.sf("~/group_pvd.mgz", mat);

## End(Not run)
```

```
write.region.aggregated
```

Write data aggregated over regions to morphometry file for group.

Description

Given an atlas, a subjects list and a measure, aggregate the measure over each region (e.g., mean) and write an output morphometry file in which the value for all region vertices is set to the aggregated value.

Usage

```
write.region.aggregated(
  subjects_dir,
  subjects_list,
  measure,
  hemi,
  atlas,
  agg_fun = mean,
  outfile_morph_name = "",
  format = "mgz"
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subjects_list	string vector. A vector of subject identifiers that match the directory names within subjects_dir.
measure	string. Name of the vertex-wise measure of morphometry data file. E.g., "area" or "thickness". Used to construct the name of the morphometry file to be loaded.
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
agg_fun	function. An R function that aggregates data, typically max, mean, min or something similar. Note: this is NOT a string, put the function name without quotes. Defaults to mean.
outfile_morph_name	string. The measure part of the output file name. E.g., 'agg_thickness' will write the file '<subject>/surf/<hemi>.agg_thickness.mgh'. Defaults to 'agg_<measure>'.
format	string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

See Also

Other output functions: [write.region.values\(\)](#), [write.region.values.fsaverage\(\)](#)

write.region.values	<i>Write one value per atlas region for a subject.</i>
---------------------	--

Description

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

Usage

```
write.region.values(
    subjects_dir,
    subject_id,
    hemi,
    atlas,
    region_value_list,
    outfile_morph_name,
    format = "mgz",
    do_write_file = TRUE,
    output_path = NULL,
    value_for_unlisted_regions = NaN
)
```

Arguments

subjects_dir	string. The FreeSurfer SUBJECTS_DIR, i.e., a directory containing the data for all your subjects, each in a subdir named after the subject identifier.
subject_id	string. The subject identifier
hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
region_value_list	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.
outfile_morph_name	string. The measure part of the output file name. E.g., 'agg_thickness' will write the file '<subject>/surf/<hemi>.agg_thickness.mgh'.
format	string. A morphometry file format. One of 'mgh', 'mgz' or 'curv.' The output file name extension will be set accordingly. Defaults to 'mgz'.

`do_write_file` logical. Whether to write the data to a file on the disk. If FALSE, the data are only returned (and the `outfile_morph_name` parameter gets ignored). Default to TRUE.

`output_path` string, path to the output directory. If omitted, defaults to the 'surf' directory of the subject (i.e., '`<subjects_dir>/<subject_id>/surf/'`).

`value_for_unlisted_regions` numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in `region_value_list`. Defaults to NaN.

Value

a named list with the following entries: "data": a vector containing the data. "file_written": string, path to the file that was written, only exists if `do_write = TRUE`.

See Also

Other output functions: `write.region.aggregated()`, `write.region.values.fsaverage()`

Examples

```
## Not run:
fsbrain::download_optional_data();
subjects_dir = fsbrain::get_optional_data_filepath("subjects_dir");
region_value_list = list("bankssts"=0.9, "precuneus"=0.7);
write.region.values(subjects_dir, 'subject1', 'lh', 'aparc',
  region_value_list, 'pvalues.mgz', do_write_file = FALSE);

## End(Not run)
```

```
write.region.values.fsaverage
```

Write one value per atlas region for a template subject.

Description

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

Usage

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

```

    template_subjects_dir = NULL,
    show_freeview_tip = FALSE,
    value_for_unlisted_regions = NaN
)

```

Arguments

hemi	string, one of 'lh' or 'rh'. The hemisphere name. Used to construct the names of the annotation and morphometry data files to be loaded.
atlas	string. The atlas name. E.g., "aparc", "aparc.2009s", or "aparc.DKTatlas". Used to construct the name of the annotation file to be loaded.
region_value_list	named list. A list in which the names are atlas regions, and the values are the value to write to all vertices of that region.
output_file	string or NULL. Path of the output file, including file name and extension. The format is determined from the (absence of a) file extension. If NULL, no file will be written.
template_subject	string, template subject name. Defaults to 'fsaverage'.
template_subjects_dir	string, the path to the subjects directory containing the template subject directory. If this is NULL, the function will try to find it using the environment, see the function find.subjectsdir.of for details. Defaults to NULL.
show_freeview_tip	logical, whether to print the freeview command on howto use the overlay to the console. (Only happens if the output_file is not NULL.)
value_for_unlisted_regions	numeric scalar. The value to assign to vertices which are part of atlas regions that are not listed in region_value_list. Defaults to NaN.

Value

a named list with the following entries: "data": a vector containing the data. "file_written": string, path to the file that was written, only exists if do_write = TRUE.

See Also

Other output functions: [write.region.aggregated\(\)](#), [write.region.values\(\)](#)

Index

- * **3d utility functions**
 - highlight.points.spheres, [108](#)
 - highlight.vertices.spheres, [112](#)
 - vertex.coords, [180](#)
- * **aggregation functions**
 - group.agg.atlas.native, [81](#)
 - group.agg.atlas.standard, [82](#)
 - group.morph.agg.standard.vertex, [93](#)
 - subject.atlas.agg, [157](#)
- * **atlas functions**
 - get.atlas.region.names, [76](#)
 - group.agg.atlas.native, [81](#)
 - group.agg.atlas.standard, [82](#)
 - group.annot, [84](#)
 - group.label.from.annot, [89](#)
 - label.from.annotdata, [119](#)
 - label.to.annot, [120](#)
 - regions.to.ignore, [143](#)
 - spread.values.over.annot, [150](#)
 - spread.values.over.hemi, [152](#)
 - spread.values.over.subject, [153](#)
 - subject.annot, [154](#)
 - subject.atlas.agg, [157](#)
 - subject.label.from.annot, [161](#)
 - subject.lobes, [162](#)
- * **color functions**
 - alphablend, [7](#)
 - desaturate, [49](#)
- * **colorbar functions**
 - coloredmesh.plot.colorbar.separate, [36](#)
 - combine.colorbar.with.brainview.animation, [41](#)
 - combine.colorbar.with.brainview.image, [42](#)
- * **coloredmesh functions**
 - coloredmesh.from.annot, [27](#)
 - coloredmesh.from.color, [28](#)
 - coloredmesh.from.label, [30](#)
 - coloredmesh.from.mask, [31](#)
 - coloredmesh.from.morph.native, [32](#)
 - coloredmesh.from.morph.standard, [33](#)
 - coloredmesh.from.morphdata, [35](#)
 - coloredmeshes.from.color, [38](#)
 - Triangles3D.to.coloredmesh, [178](#)
- * **coloredpaths functions**
 - fs.coloredpaths, [65](#)
- * **concatination functions**
 - group.concat.measures.native, [85](#)
 - group.concat.measures.standard, [86](#)
- * **connectome functions**
 - subject.region.centroids, [168](#)
 - vis.connectome, [186](#)
- * **fs_LR 32k template functions**
 - download_fs_LR_32_atlases, [54](#)
 - download_fs_LR_32_labels, [55](#)
 - download_fs_LR_32_meshes, [56](#)
- * **global aggregation functions**
 - group.morph.agg.native, [90](#)
 - group.morph.agg.standard, [92](#)
 - group.multimorph.agg.native, [98](#)
 - group.multimorph.agg.standard, [100](#)
- * **group visualization functions**
 - vis.data.on.group.native, [192](#)
 - vis.data.on.group.standard, [193](#)
 - vis.group.annot, [201](#)
 - vis.group.coloredmeshes, [202](#)
 - vis.group.morph.native, [203](#)
 - vis.group.morph.standard, [204](#)
- * **hemilist functions**
 - hemilist, [104](#)
 - hemilist.derive.hemi, [105](#)
 - hemilist.from.prefixed.list, [105](#)
 - hemilist.get.combined.data, [106](#)
 - hemilist.unwrap, [107](#)
 - hemilist.wrap, [107](#)

- is.hemilist, 117
- * **interal**
 - geodesic.circles, 74
- * **internals**
 - images.dimmax, 113
- * **label data functions**
 - group.label, 88
 - labeldata.from.mask, 121
 - mask.from.labeldata.for.hemi, 125
 - subject.label, 160
- * **label functions**
 - apply.label.to.morphdata, 10
 - apply.labeldata.to.morphdata, 11
 - subject.lobes, 162
 - subject.mask, 163
 - vis.labeldata.on.subject, 206
 - vis.subject.label, 222
- * **mask functions**
 - coloredmesh.from.mask, 31
 - mask.from.labeldata.for.hemi, 125
 - vis.mask.on.subject, 208
- * **mesh data functions**
 - group.surface, 102
- * **metadata functions**
 - demographics.to.fsgd.file, 47
 - read.md.demographics, 138
 - read.md.subjects, 139
 - report.on.demographics, 143
- * **morphometry data functions**
 - apply.label.to.morphdata, 10
 - apply.labeldata.to.morphdata, 11
 - group.morph.native, 95
 - group.morph.standard, 96
 - subject.morph.native, 165
 - subject.morph.standard, 166
- * **morphometry visualization functions**
 - vis.data.on.fsaverage, 190
 - vis.data.on.subject, 194
 - vis.subject.morph.native, 224
 - vis.subject.morph.standard, 226
 - vis.symmetric.data.on.subject, 230
- * **output functions**
 - write.region.aggregated, 271
 - write.region.values, 272
 - write.region.values.fsaverage, 273
- * **quality check functions**
 - qc.for.group, 133
 - qc.from.regionwise.df, 134
- * **region-based visualization functions**
 - vis.region.values.on.subject, 213
 - vis.subcortical.region.values, 217
 - vis.subject.annot, 221
- * **surface and volume coordinates**
 - index2ras_tkr, 114
 - ras2vox_tkr, 137
 - vox2ras_tkr, 267
- * **surface color layer**
 - collayer.bg, 20
 - collayer.bg.atlas, 21
 - collayer.bg.meancurv, 22
 - collayer.bg.sulc, 23
 - collayer.from.annot, 24
 - collayer.from.annotdata, 24
 - collayer.from.mask.data, 25
 - collayer.from.morphlike.data, 26
 - collayers.merge, 27
- * **surface mesh functions**
 - face.edges, 62
 - label.border, 117
 - mesh.vertex.neighbors, 126
 - subject.surface, 170
 - vis.path.along.verts, 210
- * **surface visualization functions**
 - highlight.vertices.on.subject, 109
 - highlight.vertices.on.subject.spheres, 110
 - vis.color.on.subject, 181
- * **tracts functions**
 - download_xtract_tracts, 58
 - read.tract.bundles, 141
 - vis.tracts, 232
- * **visualization functions**
 - highlight.vertices.on.subject, 109
 - highlight.vertices.on.subject.spheres, 110
 - vis.color.on.subject, 181
 - vis.data.on.fsaverage, 190
 - vis.data.on.subject, 194
 - vis.labeldata.on.subject, 206
 - vis.mask.on.subject, 208
 - vis.region.values.on.subject, 213
 - vis.rglwidget, 215
 - vis.subcortical.region.values, 217
 - vis.subject.annot, 221
 - vis.subject.label, 222
 - vis.subject.morph.native, 224

- vis.subject.morph.standard, 226
- vis.subject.pre, 229
- vis.symmetric.data.on.subject, 230
- vis.volume.clusters, 236
- vis.volume.on.surface, 240
- vislayout.from.coloredmeshes, 243
- * **volume to surface projection functions**
 - subject.vol2surf, 171
 - template.vol2surf, 176
- * **volume utility**
 - vol.boundary.box, 245
 - vol.imagestack, 247
 - vol.merge, 249
 - vol.overlay.colors.from.activation, 250
 - vol.planes, 251
 - vol.slice, 252
- * **volume visualization**
 - vis.volume.clusters, 236
 - vis.volume.on.surface, 240
 - volvis.lb, 255
 - volvis.lb.with.surface, 256
 - volvis.lightbox, 259
 - volvis.slices.with.surface, 264
- alphablend, 7
- alphablend(), 50
- annot.outline, 8, 21, 222
- annot.outline.border.vertices, 9
- apply.label.to.morphdata, 10
- apply.label.to.morphdata(), 12, 95, 97, 163, 164, 166, 167, 207, 224
- apply.labeldata.to.morphdata, 11
- apply.labeldata.to.morphdata(), 11, 95, 97, 163, 164, 166, 167, 207, 224
- apply.transform, 12
- arrange.brainview.images, 13, 244
- arrange.brainview.images.grid, 14, 193, 194, 202, 204, 205
- bg3d, 216
- brainviews, 15, 234
- c, 106
- clip.data, 16, 33, 34, 122, 123
- clip_fun, 17, 17, 122, 144
- cm.cbry, 18
- cm.div, 18
- cm.heat, 19
- cm.qual, 19
- cm.seq, 20
- collayer.bg, 20, 191, 196, 214, 223, 226, 228, 232
- collayer.bg(), 22–27
- collayer.bg.atlas, 21
- collayer.bg.atlas(), 21, 23–27
- collayer.bg.meancurv, 22
- collayer.bg.meancurv(), 21–27
- collayer.bg.sulc, 23
- collayer.bg.sulc(), 21–27
- collayer.from.annot, 24
- collayer.from.annot(), 21–23, 25–27
- collayer.from.annotdata, 24
- collayer.from.annotdata(), 21–24, 26, 27
- collayer.from.mask.data, 25
- collayer.from.mask.data(), 21–27
- collayer.from.morphlike.data, 26
- collayer.from.morphlike.data(), 21–27
- collayers.merge, 20, 22, 23, 27
- collayers.merge(), 21–26
- coloredmesh.from.annot, 27
- coloredmesh.from.annot(), 30–35, 39, 179
- coloredmesh.from.color, 28, 79, 219
- coloredmesh.from.color(), 28, 31–35, 39, 179
- coloredmesh.from.label, 30, 207
- coloredmesh.from.label(), 28, 30, 32–35, 39, 179
- coloredmesh.from.mask, 31
- coloredmesh.from.mask(), 28, 30, 31, 33–35, 39, 126, 179, 209
- coloredmesh.from.morph.native, 32, 60, 198, 243
- coloredmesh.from.morph.native(), 28, 30–32, 34, 35, 39, 179
- coloredmesh.from.morph.standard, 33
- coloredmesh.from.morph.standard(), 28, 30–33, 35, 39, 179
- coloredmesh.from.morphdata, 35
- coloredmesh.from.morphdata(), 28, 30–34, 39, 179
- coloredmesh.from.preloaded.data, 36
- coloredmesh.plot.colorbar.separate, 16, 36, 66, 183, 189, 191, 196, 207, 209, 214, 219, 223, 225, 228, 230, 231
- coloredmesh.plot.colorbar.separate(), 42, 44

- coloredmeshes.from.color, [29, 38](#)
- coloredmeshes.from.color(), [28, 30–35, 179](#)
- colorlist.brain.clusters, [40](#)
- colors.are.grayscale, [40](#)
- colors.have.transparency, [41](#)
- combine.colorbar.with.brainview.animation, [41](#)
- combine.colorbar.with.brainview.animation(), [38, 44](#)
- combine.colorbar.with.brainview.image, [42](#)
- combine.colorbar.with.brainview.image(), [38, 42](#)
- constant.pervertexdata, [44](#)
- cube3D.tris, [44, 46](#)
- cubes3D.tris, [46](#)
- delete_all_optional_data, [46](#)
- demographics.to.fsgd.file, [47, 140](#)
- demographics.to.fsgd.file(), [139, 140, 144](#)
- demographics.to.qdec.table.dat, [48, 137](#)
- desaturate, [49](#)
- desaturate(), [8](#)
- download_fs_LR_32_atlases, [54](#)
- download_fs_LR_32_atlases(), [55, 56](#)
- download_fs_LR_32_labels, [55, 177](#)
- download_fs_LR_32_labels(), [54, 56](#)
- download_fs_LR_32_meshes, [56, 58, 177, 233](#)
- download_fs_LR_32_meshes(), [54, 55](#)
- download_fsaverage, [50, 53, 63, 236](#)
- download_fsaverage3, [51](#)
- download_fsaverage6, [52](#)
- download_fsaverage_atlases, [52, 56, 63, 218](#)
- download_fsaverage_minimal, [53](#)
- download_optional_data, [56, 58, 218](#)
- download_optional_paper_data, [57](#)
- download_xtract_tracts, [58, 141, 232](#)
- download_xtract_tracts(), [142, 235](#)
- export, [59, 189, 199, 219, 234, 235, 238, 263](#)
- export.coloredmesh.ply, [61](#)
- face.edges, [62](#)
- face.edges(), [118, 127, 170, 211](#)
- find.freesurferhome, [63](#)
- find.subjectsdir.of, [63, 218, 237, 274](#)
- fs.coloredmesh, [64, 178](#)
- fs.coloredpaths, [65, 235](#)
- fs.home, [63, 66](#)
- fs.surface.as.adjacencylist, [67](#)
- fs.surface.to.igraph, [67](#)
- fs.surface.to.tmesh3d, [68](#)
- fs.surface.vertex.neighbors, [68](#)
- fs.tracts, [142](#)
- fsaverage.path, [64, 69](#)
- fsbrain.set.default.figsize, [70](#)
- fup, [71](#)
- gen.test.volume, [71](#)
- geod.patches.color.overlay, [72](#)
- geod.vert.neighborhood, [73](#)
- geodesic.circles, [74](#)
- geodesic.dists.to.vertex, [75](#)
- geodesic.path, [75, 211](#)
- get.atlas.region.names, [9, 10, 76, 156](#)
- get.atlas.region.names(), [82, 84, 85, 90, 120, 121, 143, 151, 153–155, 158, 162, 163](#)
- get.fsbrain.renderer.backend, [77](#)
- get.rglstyle, [29, 39, 66, 78, 111, 182, 189, 219, 263](#)
- get.view.angle.names, [16, 60, 79, 188, 192, 194, 199, 201–203, 205, 219, 243](#)
- get_optional_data_filepath, [58, 80, 177](#)
- getIn, [80](#)
- group.agg.atlas.native, [81, 134](#)
- group.agg.atlas.native(), [77, 84, 85, 90, 94, 120, 121, 143, 151, 153–155, 158, 162, 163](#)
- group.agg.atlas.standard, [82](#)
- group.agg.atlas.standard(), [77, 82, 85, 90, 94, 120, 121, 143, 151, 153–155, 158, 162, 163](#)
- group.annot, [84](#)
- group.annot(), [77, 82, 84, 90, 120, 121, 143, 151, 153–155, 158, 162, 163](#)
- group.concat.measures.native, [85](#)
- group.concat.measures.native(), [87](#)
- group.concat.measures.standard, [86](#)
- group.concat.measures.standard(), [86](#)
- group.label, [88](#)
- group.label(), [122, 126, 161](#)
- group.label.from.annot, [89](#)

- group.label.from.annot(), 77, 82, 84, 85, 120, 121, 143, 151, 153–155, 158, 162, 163
- group.morph.agg.native, 90
- group.morph.agg.native(), 93, 99, 101
- group.morph.agg.standard, 92
- group.morph.agg.standard(), 91, 99, 101
- group.morph.agg.standard.vertex, 93
- group.morph.agg.standard.vertex(), 82, 84, 158
- group.morph.native, 95, 192
- group.morph.native(), 11, 12, 97, 166, 167
- group.morph.standard, 96, 194
- group.morph.standard(), 11, 12, 95, 166, 167
- group.morph.standard.sf, 97, 194, 270
- group.multimorph.agg.native, 98
- group.multimorph.agg.native(), 91, 93, 101
- group.multimorph.agg.standard, 100
- group.multimorph.agg.standard(), 91, 93, 99
- group.surface, 102
- groupmorph.split.hemilist, 103
- hasIn, 103
- hemilist, 17, 44, 72, 103, 104, 105–108, 117, 229, 270
- hemilist(), 105–108, 117
- hemilist.derive.hemi, 105
- hemilist.derive.hemi(), 104, 106–108, 117
- hemilist.from.prefixed.list, 105
- hemilist.from.prefixed.list(), 104–108, 117
- hemilist.get.combined.data, 106
- hemilist.get.combined.data(), 104–108, 117
- hemilist.unwrap, 107
- hemilist.unwrap(), 104–106, 108, 117
- hemilist.wrap, 107
- hemilist.wrap(), 104–107, 117
- highlight.points.spheres, 108, 112
- highlight.points.spheres(), 113, 180
- highlight.vertices.on.subject, 109
- highlight.vertices.on.subject(), 112, 182, 191, 196, 207, 209, 214, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- highlight.vertices.on.subject.spheres, 110
- highlight.vertices.on.subject.spheres(), 110, 182, 191, 196, 207, 209, 214, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- highlight.vertices.spheres, 112
- highlight.vertices.spheres(), 108, 180
- image.plot, 37
- image_annotate, 265
- image_border, 258
- image_write, 258, 265
- images.dimmax, 113
- index2ras_tkr, 114, 239, 240, 261, 266
- index2ras_tkr(), 137, 268
- is.fs.coloredmesh, 115
- is.fs.coloredpaths, 115
- is.fs.coloredvoxels, 116
- is.fsbrain, 116
- is.hemilist, 117
- is.hemilist(), 104–108
- label.border, 117
- label.border(), 62, 127, 170, 211
- label.colFn, 118
- label.colFn.inv, 119
- label.from.annotdata, 119
- label.from.annotdata(), 77, 82, 84, 85, 90, 121, 143, 151, 153–155, 158, 162, 163
- label.to.annot, 120
- label.to.annot(), 77, 82, 84, 85, 90, 120, 143, 151, 153–155, 158, 162, 163
- labeldata.from.mask, 121
- labeldata.from.mask(), 88, 126, 161
- legend, 186
- limit_fun, 122, 144
- limit_fun_na, 123, 124, 144
- limit_fun_na_inside, 123, 124
- list_optional_data, 125
- makecmap, 25, 26, 30, 31, 33–36, 188, 191, 196, 200, 207, 209, 214, 219, 223, 226, 228, 230, 231, 238, 242
- mask.from.labeldata.for.hemi, 125, 208
- mask.from.labeldata.for.hemi(), 32, 88, 122, 161, 209
- material3d, 78, 234, 243

- max, [91](#), [92](#), [94](#), [99](#), [101](#)
- mean, [91](#), [93](#), [94](#), [99](#), [101](#)
- mesh.vertex.included.faces(), [62](#), [118](#), [127](#), [170](#), [211](#)
- mesh.vertex.neighbors, [126](#), [206](#)
- mesh.vertex.neighbors(), [62](#), [118](#), [170](#), [211](#)
- mkco.cluster, [128](#), [238](#)
- mkco.div, [128](#), [188](#)
- mkco.heat, [129](#)
- mkco.seq, [129](#), [188](#), [234](#), [242](#)
- numverts.lh, [130](#)
- numverts.rh, [130](#)
- par3d, [16](#), [109](#), [182](#), [183](#), [185](#), [188](#), [191](#), [195](#), [206](#), [209](#), [214](#), [216](#), [219](#), [221](#), [223](#), [225](#), [228](#), [229](#), [231](#), [238](#), [241](#), [243](#), [263](#)
- png, [37](#)
- principal.curvatures, [131](#), [148](#)
- print.fs.coloredmesh, [131](#)
- print.fs.coloredpaths, [132](#)
- print.fs.coloredvoxels, [132](#)
- print.fsbrain, [133](#)
- qc.for.group, [133](#), [136](#)
- qc.for.group(), [135](#)
- qc.from.regionwise.df, [133](#), [134](#), [134](#), [135](#)
- qc.from.regionwise.df(), [134](#)
- qc.from.segstats.table(), [134](#), [135](#)
- qc.from.segstats.tables, [135](#), [136](#)
- qc.vis.failcount.by.region, [135](#)
- qdec.table.skeleton, [48](#), [49](#), [136](#)
- ras2vox_tkr, [137](#)
- ras2vox_tkr(), [114](#), [268](#)
- read.colorcsv, [138](#)
- read.dti.tck, [141](#)
- read.dti.trk, [141](#)
- read.fs.annot, [121](#), [155](#)
- read.fs.colortable, [185](#), [217](#), [251](#), [255](#)
- read.fs.mgh, [172](#), [174](#)
- read.fs.morph, [166](#), [200](#)
- read.fs.surface, [8](#), [10](#), [62](#), [118](#), [127](#), [170](#), [200](#), [211](#), [212](#)
- read.fs.volume, [217](#), [255](#)
- read.md.demographics, [138](#), [144](#)
- read.md.demographics(), [48](#), [140](#), [144](#)
- read.md.subjects, [139](#)
- read.md.subjects(), [48](#), [139](#), [144](#)
- read.md.subjects.from.fsgd, [98](#), [140](#)
- read.table, [139](#)
- read.tract.bundles, [58](#), [141](#), [234](#)
- read.tract.bundles(), [59](#), [235](#)
- regions.to.ignore, [143](#)
- regions.to.ignore(), [77](#), [82](#), [84](#), [85](#), [90](#), [120](#), [121](#), [151](#), [153–155](#), [158](#), [162](#), [163](#)
- renderRglwidget, [216](#)
- report.on.demographics, [139](#), [143](#)
- report.on.demographics(), [48](#), [139](#), [140](#)
- resolve.template.subjects.dir, [233](#)
- rgb, [27](#)
- rglactions, [17](#), [109](#), [122–124](#), [144](#), [182](#), [188](#), [191](#), [192](#), [194](#), [195](#), [201](#), [204](#), [205](#), [207](#), [209](#), [214](#), [219](#), [221](#), [223](#), [225](#), [228](#), [229](#), [231](#), [234](#), [238](#), [263](#)
- rglo, [145](#), [188](#), [216](#), [234](#), [238](#), [241](#), [263](#)
- rglot, [145](#), [145](#)
- rglvoxels, [146](#)
- rglwidget, [215](#), [216](#)
- rglwidgetOutput, [216](#)
- rotationMatrix, [12](#)
- scale01, [147](#)
- segments, [257](#)
- segments3d, [65](#)
- shade3d, [79](#)
- shape.descriptor.names, [147](#), [148](#)
- shape.descriptors, [148](#)
- shell.extract.mesh, [114](#)
- shell.palette, [262](#)
- shift.hemis.apart, [148](#)
- sjd.demo, [149](#)
- sjld, [150](#)
- spheres3d, [146](#)
- spin3d, [185](#)
- spread.values.over.annot, [150](#)
- spread.values.over.annot(), [77](#), [82](#), [84](#), [85](#), [90](#), [120](#), [121](#), [143](#), [153–155](#), [158](#), [162](#), [163](#)
- spread.values.over.hemi, [152](#)
- spread.values.over.hemi(), [77](#), [82](#), [84](#), [85](#), [90](#), [120](#), [121](#), [143](#), [151](#), [154](#), [155](#), [158](#), [162](#), [163](#)
- spread.values.over.subject, [153](#)

- spread.values.over.subject(), [77](#), [82](#), [84](#),
[85](#), [90](#), [120](#), [121](#), [143](#), [151](#), [153](#), [155](#),
[158](#), [162](#), [163](#)
- subject.annot, [8](#), [9](#), [25](#), [120](#), [154](#), [185](#)
- subject.annot(), [77](#), [82](#), [84](#), [85](#), [90](#), [120](#),
[121](#), [143](#), [151](#), [153](#), [154](#), [158](#), [162](#),
[163](#)
- subject.annot.border, [156](#)
- subject.atlas.agg, [157](#)
- subject.atlas.agg(), [77](#), [82](#), [84](#), [85](#), [90](#), [94](#),
[120](#), [121](#), [143](#), [151](#), [153–155](#), [162](#),
[163](#)
- subject.dir.has.core.files, [63](#)
- subject.filepath.morph.native, [158](#)
- subject.filepath.morph.standard, [159](#)
- subject.label, [11](#), [125](#), [160](#)
- subject.label(), [88](#), [122](#), [126](#)
- subject.label.from.annot, [161](#)
- subject.label.from.annot(), [77](#), [82](#), [84](#),
[85](#), [90](#), [120](#), [121](#), [143](#), [151](#), [153–155](#),
[158](#), [163](#)
- subject.lobes, [162](#)
- subject.lobes(), [11](#), [12](#), [77](#), [82](#), [84](#), [85](#), [90](#),
[120](#), [121](#), [143](#), [151](#), [153–155](#), [158](#),
[162](#), [164](#), [207](#), [224](#)
- subject.mask, [163](#)
- subject.mask(), [11](#), [12](#), [163](#), [207](#), [224](#)
- subject.morph.native, [165](#), [179](#)
- subject.morph.native(), [11](#), [12](#), [95](#), [97](#),
[167](#)
- subject.morph.standard, [166](#)
- subject.morph.standard(), [11](#), [12](#), [95](#), [97](#),
[166](#)
- subject.num.verts, [168](#)
- subject.region.centroids, [168](#), [187](#)
- subject.region.centroids(), [189](#)
- subject.surface, [8](#), [10](#), [31](#), [62](#), [64](#), [112](#), [118](#),
[127](#), [170](#), [180](#), [200](#), [211](#), [212](#)
- subject.surface(), [62](#), [118](#), [127](#), [211](#)
- subject.vol2surf, [171](#), [176](#), [177](#)
- subject.vol2surf(), [177](#)
- subject.volume, [174](#), [239](#)
- surface.curvatures, [175](#)
- template.vol2surf, [55](#), [171](#), [176](#)
- template.vol2surf(), [173](#)
- text3d, [144](#)
- tmesh3d, [28](#), [29](#), [31–35](#), [39](#), [64](#)
- tmesh3d.to.fs.surface, [178](#)
- toupper, [71](#)
- triangles3d, [44](#), [146](#), [267](#)
- Triangles3D.to.coloredmesh, [178](#)
- Triangles3D.to.coloredmesh(), [28](#), [30–35](#),
[39](#)
- vdata.split.by.hemi, [179](#)
- vertex.coords, [180](#)
- vertex.coords(), [108](#), [113](#)
- vertex.hemis, [180](#)
- vis.color.on.subject, [21–26](#), [29](#), [181](#)
- vis.color.on.subject(), [110](#), [112](#), [191](#),
[196](#), [207](#), [209](#), [214](#), [216](#), [220](#), [222](#),
[224](#), [226](#), [228](#), [230](#), [232](#), [239](#), [242](#),
[244](#)
- vis.coloredmeshes, [29](#), [39](#), [66](#), [183](#), [200](#), [201](#)
- vis.coloredmeshes.rotating, [184](#)
- vis.colortable.legend, [185](#), [217](#)
- vis.connectome, [168](#), [186](#)
- vis.connectome(), [169](#)
- vis.data.on.fsaverage, [190](#)
- vis.data.on.fsaverage(), [110](#), [112](#), [182](#),
[196](#), [207](#), [209](#), [214](#), [216](#), [220](#), [222](#),
[224](#), [226](#), [228](#), [230](#), [232](#), [239](#), [242](#),
[244](#)
- vis.data.on.group.native, [192](#)
- vis.data.on.group.native(), [194](#),
[202–205](#)
- vis.data.on.group.standard, [193](#)
- vis.data.on.group.standard(), [193](#),
[202–205](#)
- vis.data.on.subject, [111](#), [194](#), [230](#)
- vis.data.on.subject(), [110](#), [112](#), [182](#), [191](#),
[207](#), [209](#), [214](#), [216](#), [220](#), [222](#), [224](#),
[226](#), [228](#), [230](#), [232](#), [239](#), [242](#), [244](#)
- vis.dti.trk, [197](#)
- vis.export.from.coloredmeshes, [198](#)
- vis.fs.surface, [200](#)
- vis.group.annot, [201](#), [203](#)
- vis.group.annot(), [193](#), [194](#), [203–205](#)
- vis.group.coloredmeshes, [202](#)
- vis.group.coloredmeshes(), [193](#), [194](#), [202](#),
[204](#), [205](#)
- vis.group.morph.native, [203](#)
- vis.group.morph.native(), [193](#), [194](#), [202](#),
[203](#), [205](#)
- vis.group.morph.standard, [204](#)
- vis.group.morph.standard(), [193](#), [194](#),
[202–204](#)

- `vis.labeldata.on.subject`, 206
- `vis.labeldata.on.subject()`, 11, 12, 110, 112, 163, 164, 182, 191, 196, 209, 214, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.mask.on.subject`, 208
- `vis.mask.on.subject()`, 32, 110, 112, 126, 182, 191, 196, 207, 214, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.path.along.verts`, 210
- `vis.path.along.verts()`, 62, 118, 127, 170
- `vis.paths`, 211, 212
- `vis.paths.along.verts`, 212
- `vis.region.values.on.subject`, 136, 213, 218, 219
- `vis.region.values.on.subject()`, 110, 112, 182, 191, 196, 207, 209, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.rglwidget`, 215
- `vis.rglwidget()`, 110, 112, 182, 191, 196, 207, 209, 214, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.seg.legend`, 217
- `vis.subcortical.region.values`, 53, 56, 79, 183, 217, 238, 239
- `vis.subcortical.region.values()`, 110, 112, 182, 191, 196, 207, 209, 214–216, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.subject.annot`, 221, 243
- `vis.subject.annot()`, 110, 112, 182, 191, 196, 207, 209, 214–216, 220, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.subject.label`, 222, 243
- `vis.subject.label()`, 11, 12, 110, 112, 163, 164, 182, 191, 196, 207, 209, 215, 216, 220, 222, 226, 228, 230, 232, 239, 242, 244
- `vis.subject.morph.native`, 37, 60, 61, 79, 128, 129, 145, 173, 198, 210, 216, 224, 238, 243, 261, 263, 266
- `vis.subject.morph.native()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 228, 230, 232, 239, 242, 244
- `vis.subject.morph.standard`, 226, 243
- `vis.subject.morph.standard()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 230, 232, 239, 242, 244
- `vis.subject.pre`, 229
- `vis.subject.pre()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 228, 232, 239, 242, 244
- `vis.symmetric.data.on.subject`, 230
- `vis.symmetric.data.on.subject()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242, 244
- `vis.tracts`, 58, 59, 141, 232
- `vis.tracts()`, 59, 142
- `vis.volume.clusters`, 236
- `vis.volume.clusters()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 228, 230, 232, 242, 244, 256, 258, 260, 265
- `vis.volume.on.surface`, 239, 240, 263
- `vis.volume.on.surface()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 228, 230, 232, 239, 244, 256, 258, 260, 265
- `vislayout.from.coloredmeshes`, 79, 243
- `vislayout.from.coloredmeshes()`, 110, 112, 182, 191, 196, 207, 209, 215, 216, 220, 222, 224, 226, 228, 230, 232, 239, 242
- `vol.boundary.box`, 245
- `vol.boundary.box()`, 247, 249, 250, 252, 253
- `vol.boundary.box.apply`, 245
- `vol.boundary.mask`, 245
- `vol.hull`, 246
- `vol.imagestack`, 247
- `vol.imagestack()`, 245, 249, 250, 252, 253
- `vol.intensity.to.color`, 247
- `vol.mask.from.segmentation`, 248
- `vol.merge`, 249
- `vol.merge()`, 245, 247, 250, 252, 253
- `vol.overlay.colors.from.activation`, 250
- `vol.overlay.colors.from.activation()`, 245, 247, 249, 252, 253
- `vol.overlay.colors.from.colortable`, 251

`vol.planes`, 251
`vol.planes()`, 245, 247, 249, 250, 253
`vol.slice`, 252
`vol.slice()`, 245, 247, 249, 250, 252
`vol.tkreg.affine`, 173
`vol.vox.from.crs`, 253
`volvis.contour`, 114, 178, 254, 263
`volvis.lb`, 255, 260
`volvis.lb()`, 239, 242, 258, 260, 265
`volvis.lb.with.surface`, 256, 264, 265
`volvis.lb.with.surface()`, 239, 242, 256, 260, 265
`volvis.lightbox`, 252, 256, 258, 259, 265
`volvis.lightbox()`, 239, 242, 256, 258, 265
`volvis.shells`, 239, 261
`volvis.slices.with.surface`, 264
`volvis.slices.with.surface()`, 239, 242, 256, 258, 260
`volvis.voxels`, 239, 263, 266
`vox2ras_tkr`, 114, 173, 240, 266, 267
`vox2ras_tkr()`, 114, 137

`which`, 121
`write.group.morph.standard`, 268
`write.group.morph.standard.mf`, 98, 269, 269
`write.group.morph.standard.sf`, 98, 269, 270, 270
`write.region.aggregated`, 271
`write.region.aggregated()`, 273, 274
`write.region.values`, 272
`write.region.values()`, 272, 274
`write.region.values.fsaverage`, 273
`write.region.values.fsaverage()`, 272, 273