Package: neurosurf (via r-universe)

June 3, 2026

Type Package

Title Data Structures and Visualization for Surface-Based Neuroimaging Data

Version 0.1.0

Description A comprehensive toolkit for working with surface-based neuroimaging data represented as triangle meshes. The package provides classes and methods for creating, manipulating, and visualizing 3D surface geometries (e.g., cortical surfaces), with support for various file formats including FreeSurfer and GIFTI. Key features include: surface smoothing, curvature computation, neighborhood graph construction, geodesic distance calculations, searchlight analysis for surface-based machine learning, and interactive 3D visualization via HTMLWidgets. The package facilitates advanced surface-based analyses through specialized data structures for representing surface geometry and associated functional data.

Imports Matrix, igraph, rgl, assertthat, digest, neuroim2, colorplane, FNN, readr, stringr, plyr, Rvcg, methods, gplots, gifti, htmlwidgets, deflist, fastmap, crayon, png, Rcpp

LinkingTo Rcpp, RcppArmadillo, roptim

License GPL (>= 2)

RoxygenNote 7.3.3

Language en-US

Collate 'all_generic.R' 'all_class.R' 'Arith.R' 'IO.R' 'ROI.R' 'RcppExports.R' 'Searchlight.R' 'curv.R' 'example_helpers.R' 'fetch_surfaces.R' 'gaussian_splat.R' 'geodesic.R' 'geometry.R' 'neighborhood.R' 'neuro_surface.R' 'neurosurf.R' 'roi_boundaries.R' 'sdf_alignment.R' 'snapshot_surface.R' 'spec.R' 'surface_montage.R' 'surface_plot.R' 'surface_set.R' 'surfwidget.R' 'testdata.R' 'view_surface.R' 'vol_to_surf.R' 'vol_to_surf_sdf.R' 'zzz.R'

Suggests callr, testthat, covr, bench, knitr, rmarkdown, webshot2, ggplot2, viridis, albersdown, rappdirs

URL https://github.com/bbuchsbaum/neurosurf

BugReports https://github.com/bbuchsbaum/neurosurf/issues

VignetteBuilder knitr

Encoding UTF-8

Config/Needs/website albersdown

Config/pak/sysreqs cmake libfreetype6-dev libglpk-dev libglu1-mesa-dev make texlive libicu-dev libpng-dev libuv1-dev libxml2-dev libgl1-mesa-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-06-03 16:32:25 UTC

RemoteUrl https://github.com/bbuchsbaum/neurosurf

RemoteRef HEAD

RemoteSha 1a150090205b36071ae19f2c8ab06f30139ba5f9

Contents

`[,NeuroSurfaceVector,missing,missing,ANY-method`
Extract All Data from NeuroSurfaceVector

Description

Extracts all data from a NeuroSurfaceVector as a matrix.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,missing,missing,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	the object
`i`	first index
`j`	second index
`...`	additional args
`drop`	dimension

Value

A numeric matrix containing all data

`[,NeuroSurfaceVector,missing,numeric,ANY-method`
Subset NeuroSurfaceVector by Column

Description

Extracts columns (time points) from a NeuroSurfaceVector.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,missing,numeric,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	the object
`i`	first index
`j`	second index
`...`	additional args
`drop`	dimension

Value

A numeric matrix or vector of extracted column data

`[,NeuroSurfaceVector,numeric,missing,ANY-method`
Subset NeuroSurfaceVector by Row

Description

Extracts rows (vertices) from a NeuroSurfaceVector.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,numeric,missing,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	the object
`i`	first index
`j`	second index
`...`	additional args
`drop`	dimension

Value

A numeric matrix or vector of extracted row data

[,NeuroSurfaceVector,numeric,numeric,ANY-method
Subset NeuroSurfaceVector

Description

Extracts a subset of data from a NeuroSurfaceVector.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,numeric,numeric,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	the object
i	first index (rows/vertices)
j	second index (columns/time points)
...	additional args
drop	whether to drop dimensions

Value

A numeric matrix or vector of the extracted data subset

[,ROISurface,numeric,missing,ANY-method
Subset an ROISurface Object

Description

Subsets an 'ROISurface' object by selecting specific vertex indices.

Usage

```
## S4 method for signature 'ROISurface,numeric,missing,ANY'
x[i, j, drop]
```

Arguments

x	The ROISurface object to subset.
i	A numeric vector specifying the indices within the ROI to select.
j	Missing (not used for this signature).
drop	Missing or ANY (ignored, always returns an ROISurface).

Value

A new ROISurface object containing only the selected vertices and their associated data from the original ROI.

[[,NeuroSurfaceVector,numeric-method
Extract Data from NeuroSurfaceVector

Description

Extracts data from a NeuroSurfaceVector using bracket notation.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,numeric'
x[[i]]
```

Arguments

x	the object
i	first index

Value

A numeric vector of data values for the specified column

`add_atlas_outline` *Add an atlas outline layer to a surface plot*

Description

This helper adds an outline-only layer to an existing `"neurosurf_plot"` object using ROI labels. It configures sensible defaults for boundary aesthetics (including a light halo and a small depth offset) so that atlas outlines remain legible over filled statistical maps.

Usage

```
add_atlas_outline(
  x,
  labels,
  rois = NULL,
  label = "atlas",
  outline_col = "black",
  outline_lwd = 1.5,
  outline_offset = 0.5,
  outline_halo = TRUE,
  outline_halo_col = "white",
  outline_halo_lwd = NULL,
  outline_lty = c("solid", "dashed"),
  ...
)
```

Arguments

`x`	A `"neurosurf_plot"` object created by `surface_plot`.
`labels`	Numeric vector or list of vectors containing ROI labels for each vertex. If a single vector is supplied, it is split across hemispheres based on vertex counts.
`rois`	Optional numeric vector of ROI ids to outline. If `NULL`, all ROIs present in `labels` are outlined.
`label`	Optional character label for this outline layer.
`outline_col`	Colour to use for ROI boundaries. Defaults to `"black"`.
`outline_lwd`	Numeric line width for boundaries. Defaults to `1.5`.
`outline_offset`	Numeric depth offset along surface normals to avoid z-fighting. Defaults to `0.5`.
`outline_halo`	Logical; if `TRUE`, draws a thicker halo under the main line for better visibility. Defaults to `TRUE`.
`outline_halo_col`	Colour for the halo. Defaults to `"white"`.
`outline_halo_lwd`	Numeric line width for the halo. If `NULL`, slightly larger than `outline_lwd`.
`outline_lty`	Line type for boundaries: `"solid"` or `"dashed"`.
`...`	Additional arguments passed through to `add_surface_layer`, allowing fine control over line colour, width, offset, and halo appearance.

Value

A modified "neurosurf_plot" object.

Examples

```
fs <- load_fsaverage_std8("inflated")
p  <- surface_plot(fs$lh, fs$rh)
# roi_labels <- ... # per-vertex ROI ids
# p <- add_atlas_outline(p, roi_labels)
```

---

add_surface_layer *Add a data layer to a surface plot*

Description

Add a data layer to a surface plot

Usage

```
add_surface_layer(
  x,
  data,
  cmap = "viridis",
  alpha = 1,
  irange = NULL,
  color_range = NULL,
  thresh = NULL,
  vertices = NULL,
  smoothing = c("auto", "nearest"),
  smoothing_steps = 20,
  as_outline = FALSE,
  zero_transparent = TRUE,
  show_colorbar = TRUE,
  label = NULL,
  outline_col = "black",
  outline_lwd = 1.5,
  outline_offset = 0,
  outline_halo = FALSE,
  outline_halo_col = NULL,
  outline_halo_lwd = NULL,
  outline_rois = NULL,
  outline_lty = c("solid", "dashed"),
  hemi = c("both", "left", "right")
)
```

Arguments

x	A "neurosurf_plot" object created by surface_plot.
data	Numeric vector or list specifying vertex-wise data. If a vector, it should have length equal to the total number of vertices across hemispheres and is split left-to-right. If a list, it may contain elements named "left" and/or "right".
cmap	Character string naming a colour map. The value is passed through to colorRampPalette to generate colours.
alpha	Numeric in $[0, 1]$ controlling layer opacity.
irange	Optional numeric vector of length 2 giving the minimum and maximum data values to map to the colour scale. Alias for color_range.
color_range	Optional numeric vector of length 2 giving the minimum and maximum data values to map to the colour scale. If NULL, the range of data (ignoring NA) is used.
thresh	Optional numeric threshold band passed to the colour mapper. A scalar is expanded to c(-abs(thresh), abs(thresh)).
vertices	Optional vector or list of vertex ids corresponding to the supplied data when it is defined on a subset of vertices. Use a list with elements left/right for hemisphere-specific subsets.
smoothing	One of "auto" (default) or "nearest" when using sparse data. "auto" fills missing vertices by nearest neighbour then applies smoothing iterations; "nearest" performs only nearest-neighbour fill.
smoothing_steps	Integer number of smoothing iterations applied when smoothing = "auto". Ignored otherwise.
as_outline	Logical; if TRUE, the data are treated as labels and are intended to be visualised as region outlines rather than a filled map. When as_outline = TRUE, the layer does not contribute to the filled vertex colours; instead its ROI boundaries are rendered as line overlays using findBoundaries.
zero_transparent	Logical; if TRUE, zeros are turned into NA so they render as transparent.
show_colorbar	Logical; if TRUE, this layer will contribute a colour bar when using a figure-level drawing helper.
label	Optional character label identifying the layer (for legends and colour bars).
outline_col	Colour to use for ROI boundaries when as_outline = TRUE. May be a single colour name/hex code or the special value "auto", in which case boundaries are coloured by ROI using the layer's cmap. Ignored for non-outline layers.
outline_lwd	Numeric line width to use when drawing ROI boundaries for outline layers. Ignored for non-outline layers.
outline_offset	Numeric scalar giving a small depth offset applied to boundary coordinates along the surface normals. This helps avoid z-fighting with the underlying mesh. A value around 0.5–1 is often sufficient for standardised cortical meshes.
outline_halo	Logical; if TRUE, draws a two-pass boundary with a thicker halo under a thinner main line to improve legibility.

outline_halo_col
: Colour used for the halo when outline_halo = TRUE. Defaults to a light colour if NULL.

outline_halo_lwd
: Numeric line width for the halo. If NULL, a slightly larger width than outline_lwd is used.

outline_rois
: Optional numeric vector of ROI ids to plot boundaries for when as_outline = TRUE. If NULL, boundaries are drawn for all ROIs present in the data.

outline_lty
: Line type for boundaries, one of "solid" or "dashed". Dashed lines are approximated by drawing alternating short segments along the boundary polyline.

hemi
: One of "both", "left", or "right", indicating which hemispheres the supplied data apply to when a single numeric vector is given.

Value

A modified "neurosurf_plot" object.

Examples

```
# Requires FreeSurfer-like surface files
# sp <- surface_plot(left = "lh.pial", right = "rh.pial")
# sp <- add_surface_layer(sp, data = rnorm(163842))
```

---

add_vector_layer *Add a vector field overlay*

Description

Add a vector field overlay

Usage

```
add_vector_layer(
  x,
  vectors,
  vertices = NULL,
  scale = NULL,
  color = "red",
  alpha = 0.8,
  lwd = 1.5,
  hemi = c("both", "left", "right")
)
```

Arguments

x	A "neurosurf_plot" object.
vectors	Matrix (n x 3) of XYZ vectors or a list with left/right matrices. When supplying a single matrix for both hemispheres, rows are split left-to-right to match the vertex ordering.
vertices	Optional vector or list of vertex ids matching the rows of vectors when defined on a subset of vertices. For both hemispheres, supply a list to avoid ambiguity.
scale	Optional numeric scalar. If NULL, a heuristic scale is derived from the mesh extent and vector magnitudes.
color	Colour for the vectors (single value or vector).
alpha	Numeric in $[0, 1]$ for vector opacity.
lwd	Numeric line width for the glyphs.
hemi	One of "both", "left", or "right" when a single vectors matrix is provided.

Value

A modified "neurosurf_plot" object.

Examples

```
# Requires FreeSurfer-like surface files
# sp <- surface_plot(left = "lh.pial", right = "rh.pial")
# vectors <- matrix(rnorm(163842 * 3), ncol = 3)
# sp <- add_vector_layer(sp, vectors = vectors)
```

---

adjacency *Get Adjacency Graph*

Description

Get Adjacency Graph

Usage

```
adjacency(x, attr, ...)

## S4 method for signature 'SurfaceGeometry,numeric'
adjacency(x, attr)

## S4 method for signature 'SurfaceGeometry,character'
adjacency(x, attr)

## S4 method for signature 'SurfaceGeometry,missing'
adjacency(x)
```

Arguments

x	Graph structure
attr	Character; edge attribute for weights in igraph object. If absent, weights are 0 or 1.
...	Additional arguments

Value

A sparse adjacency matrix of class dgCMatrix

AFNISurfaceFileDescriptor-class *AFNISurfaceFileDescriptor*

Description

This class supports the AFNI 1D file format for surface-based data

Value

An object of class AFNISurfaceFileDescriptor.

---

apply_surface_sampler *Apply a precomputed surface sampler to a volume*

Description

Apply a precomputed surface sampler to a volume

Usage

```
apply_surface_sampler(
  sampler,
  vol,
  fun = c("avg", "nn", "mode"),
  sigma = 8,
  fill = 0
)
```

Arguments

sampler	A sampler object returned by surface_sampler().
vol	A NeuroVol with the same grid as the template used to build the sampler.
fun	Aggregation function: "avg", "nn", or "mode".
sigma	Bandwidth for Gaussian weights when fun = "avg".
fill	Value used when no valid voxels fall within dthresh.

Value

NeuroSurface with mapped data.

Examples

```
# Requires surface sampler and volume data
# sampler <- surface_sampler(geometry, vol)
# result <- apply_surface_sampler(sampler, vol)
```

Arith,NeuroSurface,NeuroSurface-method

Arithmetic Operations for NeuroSurface Objects

Description

Arithmetic Operations for NeuroSurface Objects

Usage

```
## S4 method for signature 'NeuroSurface,NeuroSurface'
Arith(e1, e2)

## S4 method for signature 'NeuroSurface,numeric'
Arith(e1, e2)

## S4 method for signature 'numeric,NeuroSurface'
Arith(e1, e2)

## S4 method for signature 'NeuroSurface,NeuroSurfaceVector'
Arith(e1, e2)
```

Arguments

e1	the left operand
e2	the right operand

Value

NeuroSurface object with arithmetic operation results

Arith,NeuroSurfaceVector,NeuroSurfaceVector-method *Arithmetic Operations for NeuroSurfaceVector Objects*

Description

Arithmetic Operations for NeuroSurfaceVector Objects

Usage

```
## S4 method for signature 'NeuroSurfaceVector,NeuroSurfaceVector'
Arith(e1, e2)

## S4 method for signature 'NeuroSurfaceVector,numeric'
Arith(e1, e2)

## S4 method for signature 'numeric,NeuroSurfaceVector'
Arith(e1, e2)

## S4 method for signature 'NeuroSurfaceVector,NeuroSurface'
Arith(e1, e2)
```

Arguments

e1	NeuroSurfaceVector object or numeric value
e2	NeuroSurfaceVector object or numeric value

Value

NeuroSurfaceVector object with arithmetic operation results

as *Coercion Methods for NeuroSurface Objects*

Description

Methods to convert NeuroSurface objects to other types.

Value

The converted object (e.g., a numeric vector when converting to "vector").

as.matrix,ROISurfaceVector-method *Convert Surface Data to Matrix*

Description

Converts surface data to a matrix representation.

Usage

```
## S4 method for signature 'ROISurfaceVector'
as.matrix(x)

## S4 method for signature 'NeuroSurfaceVector'
as.matrix(x)

## S4 method for signature 'BilatNeuroSurfaceVector'
as.matrix(x)
```

Arguments

x	the object to convert to a matrix

Value

A numeric matrix with vertices as rows and time points/features as columns

as.vector,NeuroSurface-method *Convert Surface Data to Vector*

Description

Converts surface data to a numeric vector.

Usage

```
## S4 method for signature 'NeuroSurface'
as.vector(x)
```

Arguments

x	the object to convert to a vector

Value

A numeric vector of surface data values

`BilatNeuroSurfaceVector-class`
Bilateral NeuroSurface Vector Class

Description

Represents surface data for both left and right hemispheres.

Details

The BilatNeuroSurfaceVector class provides a convenient container for organizing and analyzing data from both hemispheres of the brain simultaneously. It holds two NeuroSurfaceVector objects, one for each hemisphere, allowing researchers to:

- Analyze bilateral symmetric or asymmetric patterns in brain data
- Compare corresponding regions across hemispheres
- Represent whole-brain surface data with proper hemisphere separation
- Apply operations to both hemispheres while maintaining their distinct identities

This class is particularly useful for studies examining interhemispheric differences, bilateral effects, or any analysis that requires maintaining separate representations of the two hemispheres while treating them as parts of a unified whole.

Value

An object of class `BilatNeuroSurfaceVector`.

Slots

`left` NeuroSurfaceVector instance for the left hemisphere

`right` NeuroSurfaceVector instance for the right hemisphere

See Also

`NeuroSurfaceVector`

Examples

```
# Create two simple tetrahedron meshes (one for each hemisphere)
# Left hemisphere
lh_vertices <- c(
  0, 0, 0,
  -1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
# Right hemisphere
```

```
rh_vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)

triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d objects
lh_mesh <- rgl::mesh3d(vertices = lh_vertices, triangles = triangles)
rh_mesh <- rgl::mesh3d(vertices = rh_vertices, triangles = triangles)

# Create graph representations
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
lh_graph <- igraph::graph_from_edgelist(edges)
rh_graph <- igraph::graph_from_edgelist(edges)

# Create SurfaceGeometry objects
lh_geometry <- new("SurfaceGeometry",
                   mesh = lh_mesh,
                   graph = lh_graph,
                   hemi = "left")
rh_geometry <- new("SurfaceGeometry",
                   mesh = rh_mesh,
                   graph = rh_graph,
                   hemi = "right")

# Define indices for all vertices
indices <- 1:4

# Create Matrix data for each hemisphere
# Each has 4 vertices and 2 measures
require(Matrix)
lh_data <- Matrix(
  c(0.5, 1.2, 0.8, 0.6,   # Measure 1 values
    0.7, 0.3, 1.5, 0.9),  # Measure 2 values
  nrow = 4, ncol = 2,
  byrow = FALSE
)

rh_data <- Matrix(
  c(0.4, 1.1, 0.7, 0.5,   # Measure 1 values (slightly different from left)
    0.8, 0.4, 1.6, 1.0),  # Measure 2 values (slightly different from left)
```

```
  nrow = 4, ncol = 2,
  byrow = FALSE
)

# Create NeuroSurfaceVector objects for each hemisphere
lh_nsv <- new("NeuroSurfaceVector",
              geometry = lh_geometry,
              indices = indices,
              data = lh_data)

rh_nsv <- new("NeuroSurfaceVector",
              geometry = rh_geometry,
              indices = indices,
              data = rh_data)

# Create the BilatNeuroSurfaceVector object
bilat_nsv <- new("BilatNeuroSurfaceVector",
                 left = lh_nsv,
                 right = rh_nsv)

# Now you can access each hemisphere separately:
# bilat_nsv@left@data  # Left hemisphere data
# bilat_nsv@right@data # Right hemisphere data
```

`clear_geodesic_cache` *Clear geodesic cache*

Description

Clear geodesic cache

Usage

```
clear_geodesic_cache()
```

Value

TRUE (invisibly)

Examples

```
clear_geodesic_cache()
```

cluster_threshold *Apply Cluster-Extent Threshold to Surface Data*

Description

Apply Cluster-Extent Threshold to Surface Data

Usage

```
cluster_threshold(x, threshold, size, ...)

## S4 method for signature 'NeuroSurfaceVector'
cluster_threshold(x, threshold, size = 10, index = 1)

## S4 method for signature 'NeuroSurface'
cluster_threshold(x, threshold, size = 10)
```

Arguments

x	Object to threshold
threshold	the two-element threshold range to use to define connected components
size	the minimum size of the connected components to keep
...	Additional arguments
index	the index/column of the underlying data matrix to cluster

Value

A thresholded surface object of the same class as x

See Also

conn_comp

ColorMappedNeuroSurface *ColorMappedNeuroSurface*

Description

This function creates a ColorMappedNeuroSurface object, which represents a single set of data values associated with nodes on a surface geometry, with pre-defined color mapping parameters.

Usage

```
ColorMappedNeuroSurface(geometry, indices, data, cmap, irange, thresh)
```

Arguments

geometry	A SurfaceGeometry object representing the underlying surface structure.
indices	An integer vector specifying the indices of valid surface nodes.
data	A numeric vector of data values corresponding to the surface nodes.
cmap	A character string specifying the colormap to use for mapping the data values.
irange	A numeric vector of length 2 specifying the range of values to map.
thresh	A numeric value specifying the threshold for the colormap.

Details

This object bundles the surface geometry, data, and specific color mapping parameters ('cmap', 'irange', 'thresh'). This is useful for ensuring consistent visualization across different plots or for saving a predefined view. The actual application of the color map happens during rendering (e.g., when using 'plot()").

Value

An instance of class ColorMappedNeuroSurface.

See Also

SurfaceGeometry, NeuroSurface

Examples

```
# Load a sample surface geometry
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surf_geom <- read_surf_geometry(surf_file)

# Get vertex count and generate some random data
n_verts <- nrow(coords(surf_geom))
set.seed(123)
vertex_data <- rnorm(n_verts)

# Define indices (all vertices in this case)
vertex_indices <- 1:n_verts

# Define color mapping parameters
my_cmap <- colorRampPalette(c("blue", "white", "red"))(256) # Blue-white-red colormap
my_irange <- c(-2, 2) # Map data values from -2 to 2 onto the colormap
my_thresh <- c(-1, 1) # Define thresholds (e.g., for transparency later)

# Create the ColorMappedNeuroSurface object
mapped_surf <- ColorMappedNeuroSurface(geometry = surf_geom,
                                       indices = vertex_indices,
                                       data = vertex_data,
                                       cmap = my_cmap,
                                       irange = my_irange,
                                       thresh = my_thresh)
```

```
# Print the object summary
print(mapped_surf)

# The object can now be plotted, and the plotting function will use
# the stored cmap, irange, and thresh parameters by default.
# plot(mapped_surf) # Requires rgl package
```

`ColorMappedNeuroSurface-class` *ColorMappedNeuroSurface*

Description

A three-dimensional surface consisting of a set of triangle vertices with one value per vertex, mapped to colors using a specified colormap and range.

Details

This class extends `NeuroSurface` by adding color mapping functionality. The `cmap` slot contains a vector of hex color codes that define the colormap. The `irange` slot specifies the range of data values to be mapped to the colormap. The `thresh` slot defines the visibility thresholds: data values below `thresh[1]` or above `thresh[2]` are visible, while values in between are not visible.

The color mapping process works as follows:

1. Data values are linearly mapped to the range [0,1] based on `irange`
2. These normalized values are used to interpolate colors from the `cmap`
3. Values falling between the thresholds in `thresh` are marked as invisible

This approach is particularly useful for visualizing statistical maps (e.g., t-statistics) where researchers are often interested in highlighting values above or below certain significance thresholds.

Value

An object of class `ColorMappedNeuroSurface`

Slots

`geometry` The surface geometry, an instance of `SurfaceGeometry` or `SurfaceSet`

`indices` An `integer` vector specifying the subset of valid surface nodes encoded in the `geometry` object

`data` The 1-D vector of data values at each vertex of the mesh

`cmap` A character vector of hex color codes representing the colormap

`irange` A numeric vector of length 2 specifying the low and high values for color mapping

`thresh` A numeric vector of length 2 specifying the low and high thresholds for visibility

See Also

view_surface, plot-methods

Examples

```
# First create a simple tetrahedron mesh
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define indices for all vertices
indices <- 1:4

# Create data values for each vertex
vertex_data <- c(-1.5, -0.5, 0.8, 2.0)  # example values for the vertices

# Define a simple colormap (blue to red)
colormap <- c("#0000FF", "#FFFFFF", "#FF0000")

# Define intensity range for mapping data to colors
intensity_range <- c(-2.0, 2.0)

# Define thresholds (values between -0.5 and 0.5 will be invisible)
thresholds <- c(-0.5, 0.5)

# Create the ColorMappedNeuroSurface object
```

```
colored_surface <- new("ColorMappedNeuroSurface",
                       geometry = geometry,
                       indices = indices,
                       data = vertex_data,
                       cmap = colormap,
                       irange = intensity_range,
                       thresh = thresholds)

# In this example:
# - Vertex 1 (-1.5) will be visible and colored blue (below lower threshold)
# - Vertex 2 (-0.5) will be exactly at the lower threshold
# - Vertex 3 (0.8) will be visible and colored light red (above upper threshold)
# - Vertex 4 (2.0) will be visible and colored bright red (above upper threshold)
```

Compare,NeuroSurface,NeuroSurface-method

Comparison Operations for NeuroSurface Objects

Description

Comparison Operations for NeuroSurface Objects

Usage

```
## S4 method for signature 'NeuroSurface,NeuroSurface'
Compare(e1, e2)
```

Arguments

e1	NeuroSurface object
e2	NeuroSurface object

Value

NeuroSurface object with comparison results

`Compare,NeuroSurface,numeric-method`
Comparison Operations for NeuroSurface Objects

Description

Comparison Operations for NeuroSurface Objects

Usage

```
## S4 method for signature 'NeuroSurface,numeric'
Compare(e1, e2)
```

Arguments

`e1`	the left operand
`e2`	the right operand

Value

NeuroSurface object with comparison results

`Compare,NeuroSurfaceVector,NeuroSurfaceVector-method`
Comparison Operations for NeuroSurfaceVector Objects

Description

Comparison Operations for NeuroSurfaceVector Objects

Usage

```
## S4 method for signature 'NeuroSurfaceVector,NeuroSurfaceVector'
Compare(e1, e2)

## S4 method for signature 'NeuroSurfaceVector,numeric'
Compare(e1, e2)

## S4 method for signature 'numeric,NeuroSurfaceVector'
Compare(e1, e2)
```

Arguments

`e1`	NeuroSurfaceVector object or numeric value
`e2`	NeuroSurfaceVector object or numeric value

Value

NeuroSurfaceVector object with comparison results

`compute_hull_world_cpp`
Compute boundary hull points in world space (C++)

Description

Compute boundary hull points in world space (C++)

Usage

```
compute_hull_world_cpp(vol, dims, grid2world, thresh, n_points, mask)
```

Arguments

`vol`	flattened numeric array (dim = dims)
`dims`	integer vector length 3 (nx, ny, nz)
`grid2world`	4x4 matrix mapping voxel indices to world coords
`thresh`	intensity threshold
`n_points`	max number of hull points (approximate)
`mask`	optional logical vector same length as vol; if empty, ignored

Value

numeric matrix (n_points x 3) of (x,y,z) in world coords

`conn_comp,NeuroSurfaceVector-method`
Compute Connected Components on a Surface

Description

This method identifies connected components on a NeuroSurface object based on a given threshold.

Usage

```
## S4 method for signature 'NeuroSurfaceVector'
conn_comp(x, threshold, index = 1)

## S4 method for signature 'NeuroSurface'
conn_comp(x, threshold)
```

Arguments

x	A NeuroSurface object representing the surface data.
threshold	A numeric vector of length 2 specifying the lower and upper thresholds for including vertices in the components.
index	the index/column of the underlying data matrix to cluster

Details

This method computes connected components on the surface by:

1. Thresholding the surface data using the provided threshold values.
2. Creating a subgraph of the surface mesh containing only the vertices that pass the threshold.
3. Identifying connected components in this subgraph.
4. Assigning component indices and sizes to the original vertices.

Vertices that do not pass the threshold are assigned a value of 0 in both output surfaces.

Value

A list containing two NeuroSurface objects:

index	A NeuroSurface object where each vertex is labeled with its component index.
size	A NeuroSurface object where each vertex is labeled with the size of its component.

See Also

NeuroSurface, components

Examples

```
# Load a sample surface from the package
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surf_geom <- read_surf_geometry(surf_file)

# Create random data for the surface with some clusters
n_vertices <- nrow(coords(surf_geom))
set.seed(123)
random_data <- rnorm(n_vertices, mean = 0, sd = 1)

# Create a few clusters of higher values
cluster_centers <- sample(1:n_vertices, 5)
g <- graph(surf_geom)

# For each cluster center, set nearby vertices to higher values
for (center in cluster_centers) {
  # Get neighbors within 2 steps
  neighbors <- unlist(igraph::neighborhood(g, 2, center))
  random_data[neighbors] <- random_data[neighbors] + 2
}
```

```
# Create a NeuroSurface object
neuro_surf <- NeuroSurface(geometry = surf_geom,
                           indices = 1:n_vertices,
                           data = random_data)

# Find connected components with threshold c(-Inf, 1.5)
# This will identify clusters where values are >= 1.5
components <- conn_comp(neuro_surf, c(-Inf, 1.5))

# Check the number of components found
max(series(components$index, seq_len(n_vertices)))

# Check the size of the largest component
max(series(components$size, seq_len(n_vertices)))

# Count vertices in components of size >= 10
sum(series(components$size, seq_len(n_vertices)) >= 10)
```

`coords,ROISurface-method`

Extract Vertex Coordinates

Description

Extracts the 3D coordinates of vertices from a surface object.

Usage

```
## S4 method for signature 'ROISurface'
coords(x)

## S4 method for signature 'SurfaceGeometry'
coords(x)

## S4 method for signature 'igraph'
coords(x)

## S4 method for signature 'NeuroSurfaceVector'
coords(x)

## S4 method for signature 'NeuroSurface'
coords(x)
```

Arguments

`x`	the object to extract coordinates from

Value

A numeric matrix with three columns (x, y, z) and one row per vertex

---

`curv_cols` *Convert Curvature Values to Binary Colors for Visualization*

Description

This function maps a vector of surface curvature values (e.g., mean curvature) to a binary color scheme, typically used to distinguish gyri (outward folds) from sulci (inward folds) on a brain surface visualization.

Usage

```
curv_cols(vals, incol = "#B3B3B3", outcol = "#404040")
```

Arguments

`vals`	A numeric vector containing curvature values for each vertex on the surface.
`incol`	A character string specifying the hex color code to represent vertices with curvature values *greater than* the median curvature. Default is "#B3B3B3" (light gray).
`outcol`	A character string specifying the hex color code to represent vertices with curvature values *less than or equal to* the median curvature. Default is "#404040" (dark gray).

Details

Surface curvature provides information about the local shape of the surface. Mean curvature is often used, where positive values typically indicate outward curvature (gyri) and negative values indicate inward curvature (sulci). This function simplifies the curvature map into two colors based on whether the value is above or below the median curvature, providing a quick visual distinction between these features. Note the default coloring assigns ‘incol‘ to values *above* the median and ‘outcol‘ to values *at or below* the median. You might need to adjust ‘incol‘ and ‘outcol‘ depending on the specific interpretation of curvature values in your data (e.g., if positive values represent sulci).

Value

A character vector of the same length as ‘vals‘, containing hex color codes based on the binary classification of curvature values relative to the median.

See Also

`curvature`, `view_surface`

Examples

```
# Generate some example curvature values
set.seed(123)
curvature_values <- rnorm(100, mean = 0, sd = 0.1)

# Get binary colors using default light/dark gray
gray_colors <- curv_cols(curvature_values)
table(gray_colors)

# Use different colors (e.g., red for above median, blue for below)
red_blue_colors <- curv_cols(curvature_values, incol = "#FF0000", outcol = "#0000FF")
table(red_blue_colors)
```

---

curv_cols_smooth *Convert Curvature Values to Smooth Gradient Colors*

Description

Maps surface curvature values to a continuous grayscale gradient, producing a FreeSurfer-style sulcal shading that is visually smoother than the binary mapping of curv_cols.

Usage

```
curv_cols_smooth(
  vals,
  light = "#C8C8C8",
  dark = "#4D4D4D",
  quantiles = c(0.05, 0.95),
  sharpness = 6
)
```

Arguments

vals	A numeric vector of curvature values for each vertex.
light	Hex color for the lightest shade (gyral crowns). Default "#D4D4D4".
dark	Hex color for the darkest shade (sulcal fundi). Default "#3A3A3A".
quantiles	Length-2 numeric vector of lower and upper quantiles used to clamp extreme values before rescaling. Default c(0.05, 0.95).
sharpness	Non-negative number controlling how crisp the gyral/sulcal split is. Larger values push vertices toward the light/dark extremes (a more binary, FreeSurfer-like contrast); smaller values give a softer gradient. 0 reproduces a plain linear ramp. Default 6.

Details

Values are clamped to the range defined by `quantiles` to prevent outliers from washing out the colour map. The clamped values are then centred on their median (the gyral/sulcal boundary) and passed through a logistic contrast curve governed by `sharpness` before being interpolated between `dark` (sulcal fundi) and `light` (gyral crowns).

A plain linear ramp leaves most vertices near mid-grey, so the fold pattern tends to disappear once surface lighting is applied. Pushing values toward the two extremes with a logistic curve keeps the sulci clearly dark and the gyri clearly light, which reads as anatomical folding under lighting much the way FreeSurfer / Connectome Workbench curvature overlays do.

Value

A character vector of hex color codes the same length as `vals`.

See Also

`curv_cols`, `curvature`, `view_surface`

Examples

```
set.seed(1)
curv <- rnorm(500, sd = 0.1)
cols <- curv_cols_smooth(curv)
head(cols)

# Softer, more continuous gradient
cols_soft <- curv_cols_smooth(curv, sharpness = 2)
```

---

curvature *Compute Surface Curvature Vector*

Description

Compute Surface Curvature Vector

Usage

```
curvature(x, ...)

curvature(x, ...)

## S4 method for signature 'SurfaceGeometry'
curvature(x)
```

Arguments

x	Object to compute curvature from
...	Additional arguments

Value

A numeric vector of curvature values, one per vertex

---

data_reader,SurfaceGeometryMetaInfo-method *Create a Column Reader for Surface Data*

Description

Creates a column reader object for accessing surface data.

Usage

```
## S4 method for signature 'SurfaceGeometryMetaInfo'
data_reader(x)

## S4 method for signature 'NIMLSurfaceDataMetaInfo'
data_reader(x)
```

Arguments

x	A SurfaceGeometryMetaInfo object

Value

A `ColumnReader` object for accessing surface data columns

---

debug_surfwidget *Debugging Helper for surfwidget*

Description

This function helps diagnose issues with surfwidget rendering by checking common problems and providing debugging information.

Usage

```
debug_surfwidget(x, verbose = TRUE)
```

Arguments

x	The surface object you're trying to visualize
verbose	Logical, whether to print detailed information

Value

Invisibly returns a list of diagnostic information

Examples

```
# Load a surface and check it
surf_file <- system.file("extdata", "std.8_lh.smoothwm.asc", package="neurosurf")
surf <- read_surf(surf_file)
debug_surfwidget(surf)
```

---

draw_surface_plot *Draw a static multi-panel surface figure*

Description

This is a convenience wrapper around render_surface_plot that arranges rendered panels into a single static figure using the **grid** graphics system. It supersamples, crops whitespace, and preserves per-panel aspect ratios to avoid tiny or distorted brains when assembled.

Usage

```
draw_surface_plot(
  x,
  colorbar = TRUE,
  cbar_location = c("bottom", "right"),
  cbar_kws = list()
)
```

Arguments

x	A "neurosurf_plot" object.
colorbar	Logical; if TRUE, draws one or more colour bars for non-outline layers that have show_colorbar = TRUE.
cbar_location	Location of colour bars relative to the panel layout. Currently supports "bottom" (default) or "right".
cbar_kws	Optional list of graphical parameters for colour bars (e.g., bar_height, title_cex, label_cex, n_ticks).

Value

A grob object that can be drawn with `grid::grid.draw()`.

Examples

```
# Requires FreeSurfer-like surface files
# sp <- surface_plot(left = "lh.pial", right = "rh.pial")
# g <- draw_surface_plot(sp)
# grid::grid.draw(g)
```

---

faces *Extract Faces from a Surface Object*

Description

Extracts the triangle faces from a surface object, providing a standardized interface across different surface representations.

Usage

```
faces(x, ...)

## S4 method for signature 'SurfaceGeometry'
faces(x)

## S4 method for signature 'NeuroSurface'
faces(x)

## S4 method for signature 'NeuroSurfaceVector'
faces(x)
```

Arguments

x	An object representing a surface.
...	Additional arguments passed to methods.

Value

A matrix where each row represents a triangular face, containing the 1-based vertex indices that form the triangle.

See Also

vertices, nodes

Examples

```
geom <- example_surface_geometry()
face_data <- faces(geom)
num_faces <- nrow(face_data)
```

`find_all_neighbors` *Find Node Neighbors in a Surface Mesh*

Description

Identifies all neighboring nodes within a specified radius for a given surface mesh.

Usage

```
find_all_neighbors(
  surf,
  radius,
  edgeWeights,
  nodes = NULL,
  distance_type = c("euclidean", "geodesic", "spherical")
)
```

Arguments

`surf`	A SurfaceGeometry object or igraph object representing the mesh
`radius`	Numeric; the spatial radius within which to search for neighbors. Must be positive.
`edgeWeights`	Numeric vector; weights for edges used in distance computation. Length must equal the number of edges.
`nodes`	Integer vector; subset of nodes to find neighbors for. If NULL, uses all nodes
`distance_type`	Character; type of distance metric to use: "euclidean", "geodesic", or "spherical"

Details

The function supports three distance metrics: Euclidean, geodesic, and spherical. For spherical distances, the surface is assumed to be a sphere. The internal k-nearest-neighbor search is capped at `vcount(g) - 1` to avoid requesting more neighbors than exist in the graph.

Value

A list of matrices, each containing neighbor information:

`i`	Source node index
`j`	Neighbor node index
`d`	Distance between nodes

Examples

```
# Load a sample inflated surface from the package
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surf <- read_surf_geometry(surf_file)

# Create edge weights (using uniform weights for simplicity)
g <- graph(surf)
edge_weights <- rep(1, length(igraph::E(g)))

# Find neighbors within a 10mm radius for the first 5 vertices
neighbors <- find_all_neighbors(surf, radius = 10,
                                edgeWeights = edge_weights,
                                nodes = 1:5,
                                distance_type = "geodesic")

# Check the number of neighbors found for the first vertex
nrow(neighbors[[1]])

# Look at the first few neighbors of the first vertex
head(neighbors[[1]])
```

---

find_nearest_vertex *Find the nearest surface vertex to a 3D point*

Description

Find the nearest surface vertex to a 3D point

Usage

```
find_nearest_vertex(surface, point)
```

Arguments

surface	A SurfaceGeometry.
point	Numeric vector of length 3.

Value

Integer vertex index (1-based) of the closest vertex.

Examples

```
geom <- example_surface_geometry()
find_nearest_vertex(geom, c(0.9, 0, 0))
```

`find_roi_boundaries` *Find boundaries of ROIs on a surface mesh*

Description

This function identifies boundaries between regions of interest (ROIs) defined on a triangular surface mesh. It is a translation of the core parts of Stuart Oldham's `findROIboundaries` MATLAB function, adapted for use with neurosurf objects.

Usage

```
find_roi_boundaries(
  vertices,
  faces,
  vertex_id,
  boundary_method = c("midpoint", "faces", "edge_vertices"),
  verbose = FALSE,
  use_cpp = TRUE
)
```

Arguments

`vertices`	Numeric matrix of vertex coordinates ($n \times 3$).
`faces`	Integer matrix of face indices ($m \times 3$, 1-based).
`vertex_id`	Integer vector of ROI labels for each vertex (length n).
`boundary_method`	One of `"midpoint"`, `"faces"`, or `"edge_vertices"`.
`verbose`	Logical; if `TRUE`, print progress messages.
`use_cpp`	Logical; if `TRUE`, use optimized C++ implementation (applies to `"edge_vertices"` only).

Details

Three boundary representations are currently supported:

- `"midpoint"` (default): returns crisp single-width contour segments that run *between* differing labels, through the midpoints of the mesh edges that separate them. This is the recommended method for drawing clean ROI/atlas outlines: shared borders are drawn once (no double lines) and adjacent segments join into continuous contours.
- `"faces"`: returns a logical vector indicating which faces lie on a boundary between ROIs.
- `"edge_vertices"`: returns boundary polygons traced through mesh vertices. Borders are traced along vertices on both sides of a boundary, which can read as a thicker double line on coarse meshes.

Value

A list with elements:

`boundary` For `"faces"`: logical vector of length `nrow(faces)` indicating boundary faces. For `"midpoint"`: list of 2×3 coordinate matrices, one per contour segment. For `"edge_vertices"`: list of coordinate matrices ($k \times 3$) giving boundary polygons for each connected ROI component.

`boundary_roi_id` Integer vector giving the ROI id associated with each boundary polygon (empty for `"faces"`).

`roi_components` Integer vector giving the number of connected components for each ROI (indexed parallel to `sort(unique(vertex_id))`).

`boundary_verts` For `"edge_vertices"`: list of integer vectors giving the vertex ids used for each boundary polygon; `NULL` for `"faces"`.

Examples

```
# Simple cube mesh with two ROIs (bottom vs top)
vertices <- matrix(c(
  0,0,0, 1,0,0, 1,1,0, 0,1,0,
  0,0,1, 1,0,1, 1,1,1, 0,1,1
), ncol = 3, byrow = TRUE)

faces <- matrix(c(
  1,2,3, 1,3,4,
  5,6,7, 5,7,8,
  1,2,6, 1,6,5,
  3,4,8, 3,8,7,
  1,4,8, 1,8,5,
  2,3,7, 2,7,6
), ncol = 3, byrow = TRUE)

roi <- c(1,1,1,1, 2,2,2,2)

b <- find_roi_boundaries(vertices, faces, roi, boundary_method = "edge_vertices")
str(b$boundary)
```

---

findBoundaries *Find Boundaries Between Regions on a Surface*

Description

This generic function identifies boundaries between different regions or parcellations on a surface. The implementation depends on the class of the input object.

Usage

```
findBoundaries(x, method = "midpoint", ...)

## S4 method for signature 'NeuroSurface'
findBoundaries(x, method = c("midpoint", "edge_vertices", "faces"), ...)
```

Arguments

x	A NeuroSurface object whose data slot contains integer ROI labels for each vertex.
method	Boundary method passed to find_roi_boundaries. One of "midpoint" (default, crisp single-width contours), "edge_vertices", or "faces".
...	Additional arguments passed to find_roi_boundaries.

Details

This function provides a high-level interface for finding boundaries between different regions on a surface mesh. It typically returns coordinates and metadata describing the boundaries between regions.

Value

An object containing boundary information. The specific structure depends on the method implementation.

A list as returned by find_roi_boundaries.

See Also

find_roi_boundaries

find_roi_boundaries

Examples

```
# Requires surface with ROI labels
# boundaries <- findBoundaries(labeled_surface)
```

FreesurferAsciiSurfaceFileDescriptor-class

FresurferAsciiSurfaceFileDescriptor

Description

This class supports the Freesurfer Ascii file format for surface geometry

Value

An object of class `FreesurferAsciiSurfaceFileDescriptor`.

Examples

```
# Create a FreesurferAsciiSurfaceFileDescriptor object
fs_ascii_descriptor <- new("FreesurferAsciiSurfaceFileDescriptor")

# This descriptor would typically be used when loading Freesurfer ASCII surfaces
# Example of a path to a Freesurfer ASCII surface file:
# fs_ascii_file <- "/path/to/subject/surf/lh.pial.asc"

# In practice, this descriptor would be used in code like:
# surface_geo <- loadSurfaceGeometry(fs_ascii_file, descriptor = fs_ascii_descriptor)
```

FreesurferBinarySurfaceFileDescriptor-class *FresurferBinarySurfaceFileDescriptor*

Description

This class supports the Freesurfer binary file format for surface geometry

Value

An object of class `FreesurferBinarySurfaceFileDescriptor`.

Examples

```
# Create a FreesurferBinarySurfaceFileDescriptor object
fs_binary_descriptor <- new("FreesurferBinarySurfaceFileDescriptor")

# This descriptor would typically be used when loading Freesurfer binary surfaces
# Example of a path to a Freesurfer binary surface file:
# fs_binary_file <- "/path/to/subject/surf/lh.pial"

# In practice, this descriptor would be used in code like:
# surface_geo <- loadSurfaceGeometry(fs_binary_file, descriptor = fs_binary_descriptor)
```

FreesurferSurfaceGeometryMetaInfo-class *FreesurferSurfaceGeometryMetaInfo Class*

Description

The 'FreesurferSurfaceGeometryMetaInfo' class extends 'SurfaceGeometryMetaInfo' to specifically handle meta information for Freesurfer-formatted brain surface geometries.

Details

This class inherits all slots from the parent 'SurfaceGeometryMetaInfo' class and is specialized for working with Freesurfer surface files (e.g., .asc, .pial, .white, .inflated). It maintains the same structure but is used specifically when the surface data originates from Freesurfer processing pipelines.

Value

An object of class `FreesurferSurfaceGeometryMetaInfo`.

Examples

```
fs_meta <- new("FreesurferSurfaceGeometryMetaInfo",
               header_file = "lh.white.asc",
               data_file = "lh.white.asc",
               file_descriptor = new("FileFormat"),
               vertices = 140000L,
               faces = 279998L,
               embed_dimension = 3L,
               label = "white",
               hemi = "lh")
```

gaussian_splat *Gaussian splats on surface meshes*

Description

Create per-vertex Gaussian falloff maps centred on coordinates or vertices. Distances can be measured either in straight-line Euclidean space or along the mesh using geodesic paths (Dijkstra on edge lengths).

Usage

```
gaussian_splat(surface, center, sigma, use_geodesic = FALSE)

gaussian_splat_vertex(surface, vertex_idx, sigma, use_geodesic = FALSE)

gaussian_splat_multi(
  surface,
  centers,
  sigmas,
  weights = NULL,
  use_geodesic = FALSE
)
```

Arguments

surface	A SurfaceGeometry.
center	Numeric vector of length 3 giving the centre coordinate.
sigma	Positive numeric standard deviation of the Gaussian kernel.
use_geodesic	Logical; when TRUE use along-surface distances.
vertex_idx	Integer index (1-based) of the centre vertex.
centers	Numeric matrix with three columns; each row is a centre.
sigmas	Numeric vector of length 1 or nrow(centers).
weights	Optional numeric vector of length 1 or nrow(centers) used to weight each centre (defaults to 1).

Value

A NeuroSurface with values $\exp(-d^2/(2*sigma^2))$ at every vertex. For gaussian_splat_multi the per-centre maps are summed (after weighting).

Examples

```
geom <- example_surface_geometry()
g1 <- gaussian_splat(geom, center = c(0, 0, 0), sigma = 1)
v_idx <- find_nearest_vertex(geom, c(0.1, 0, 0))
g2 <- gaussian_splat_vertex(geom, vertex_idx = v_idx, sigma = 1)
g_multi <- gaussian_splat_multi(
  geom,
  centers = rbind(c(0, 0, 0), c(1, 0, 0)),
  sigmas = c(0.5, 1)
)
```

`geodesic_distance_matrix` *All-pairs geodesic distance matrix (chunked)*

Description

All-pairs geodesic distance matrix (chunked)

Usage

```
geodesic_distance_matrix(
  geometry,
  vertices = NULL,
  targets = NULL,
  weights = NULL,
  mode = c("sparse", "dense"),
  chunk_size = 2000,
  cache = TRUE,
  cache_key = NULL,
  algorithm = c("dijkstra")
)
```

Arguments

`geometry`	`SurfaceGeometry`, `LabeledNeuroSurface`, `NeuroSurface`, or an `igraph`.
`vertices`	Optional integer vector of source vertices. Defaults to all vertices.
`targets`	Optional integer vector of target vertices. Defaults to `vertices` (square matrix).
`weights`	Optional numeric edge weights (defaults to `E(g)$dist`).
`mode`	Output mode: `"sparse"` (default, `dgCMatrix`) or `"dense"`.
`chunk_size`	Number of source vertices per Dijkstra batch.
`cache`	Logical; cache symmetric results when `targets` is NULL or identical to `vertices`.
`cache_key`	Optional manual cache key.
`algorithm`	Only "dijkstra" is currently implemented.

Value

A distance matrix (sparse or dense) of size `length(vertices)` x `length(targets)`.

Examples

```
geom <- example_surface_geometry()
dmat <- geodesic_distance_matrix(geom, vertices = 1:2)
```

`geodesic_distances` *Geodesic distances from sources to targets*

Description

Convenience wrapper returning a numeric vector/matrix.

Usage

```
geodesic_distances(
  geometry,
  sources,
  targets = NULL,
  weights = NULL,
  algorithm = c("dijkstra"),
  chunk_size = 2000
)
```

Arguments

`geometry`	`SurfaceGeometry`, `LabeledNeuroSurface`, `NeuroSurface`, or an `igraph`.
`sources`	Integer vector of source vertices.
`targets`	Optional target vertices (defaults to `sources`).
`weights`	Optional numeric edge weights (defaults to `E(g)$dist`).
`algorithm`	Only "dijkstra" is currently implemented.
`chunk_size`	Number of source vertices per Dijkstra batch.

Value

Numeric vector if one source, otherwise a matrix.

Examples

```
geom <- example_surface_geometry()
d <- geodesic_distances(geom, sources = 1, targets = 2:4)
```

`geometry` *Extract Geometry from Surface Object*

Description

Extracts the geometric representation from a surface object.

Usage

```
geometry(x)

## S4 method for signature 'NeuroSurface'
geometry(x)

## S4 method for signature 'NeuroSurfaceVector'
geometry(x)
```

Arguments

`x`	A surface object

Value

A geometry object representing the surface structure

See Also

`vertices`, `nodes`

Examples

```
# Load a sample surface
surf_file <- system.file("extdata", "std.8_lh.smoothwm.asc", package = "neurosurf")
surf <- read_surf_geometry(surf_file)
# Create a NeuroSurface with some data
ns <- NeuroSurface(surf, indices = 1:100, data = rnorm(100))
# Extract the geometry
geom <- geometry(ns)
class(geom)
```

get_surface *Retrieve a geometry from a SurfaceSet*

Description

Retrieve a geometry from a SurfaceSet

Usage

```
get_surface(x, label = NULL)
```

Arguments

x	SurfaceSet
label	Label of the desired surface variant. Defaults to the set's default.

Value

A 'SurfaceGeometry' object.

Examples

```
geom <- example_surface_geometry()
ss <- surface_set(inflated = geom)
get_surface(ss, "inflated")
```

GIFTISurfaceDataMetaInfo-class *GIFTISurfaceDataMetaInfo*

Description

This class contains meta information for surface-based data for the GIFTI data format

Value

An object of class `GIFTISurfaceDataMetaInfo`.

Slots

`info` the underlying `gifti` object returned by `readgii`

Examples

```
# First create a SurfaceDataMetaInfo parent object
meta_info <- new("SurfaceDataMetaInfo",
                 header_file = "rscan01_lh.gii",
                 data_file = "rscan01_lh.gii",
                 file_descriptor = new("FileFormat"),
                 node_count = 40000L,
                 nels = 1L,
                 label = "thickness")

# Use the example file included in the package
example_gii_file <- system.file("extdata", "rscan01_lh.gii", package = "neurostyle")

if (file.exists(example_gii_file) && requireNamespace("gifti", quietly = TRUE)) {
  # Load the actual GIFTI file
  gifti_obj <- gifti::readgii(example_gii_file)

  # Create GIFTISurfaceDataMetaInfo object with the real file
  gifti_data_meta <- new("GIFTISurfaceDataMetaInfo",
                         header_file = meta_info@header_file,
                         data_file = meta_info@data_file,
                         file_descriptor = meta_info@file_descriptor,
                         node_count = meta_info@node_count,
                         nels = meta_info@nels,
                         label = meta_info@label,
                         info = gifti_obj)
}
```

GIFTISurfaceFileDescriptor-class

GIFTISurfaceFileDescriptor

Description

This class supports the GIFTI file format for surface-based data

Value

An object of class `GIFTISurfaceFileDescriptor`.

GIFTISurfaceGeometryMetaInfo-class
GIFTISurfaceGeometryMetaInfo

Description

This class contains meta information for surface-based geometry for the GIFTI data format

Value

An object of class `GIFTISurfaceGeometryMetaInfo`.

Slots

`info` the underlying `gifti` object returned by `readgii`

Examples

```
# First create a SurfaceGeometryMetaInfo parent object
meta_info <- new("SurfaceGeometryMetaInfo",
                 header_file = "rscan01_lh.gii",
                 data_file = "rscan01_lh.gii",
                 file_descriptor = new("FileFormat"),
                 vertices = 40000L,
                 faces = 79998L,
                 embed_dimension = 3L,
                 label = "white",
                 hemi = "lh")

# Use the example file included in the package
example_gii_file <- system.file("extdata", "rscan01_lh.gii", package = "neurostyle")

if (file.exists(example_gii_file) && requireNamespace("gifti", quietly = TRUE)) {
  # Load the actual GIFTI file
  gifti_obj <- gifti::readgii(example_gii_file)

  # Create GIFTISurfaceGeometryMetaInfo object with the real file
  gifti_meta <- new("GIFTISurfaceGeometryMetaInfo",
                    header_file = meta_info@header_file,
                    data_file = meta_info@data_file,
                    file_descriptor = meta_info@file_descriptor,
                    vertices = meta_info@vertices,
                    faces = meta_info@faces,
                    embed_dimension = meta_info@embed_dimension,
                    label = meta_info@label,
                    hemi = meta_info@hemi,
                    info = gifti_obj)
}
```

graph *extract* igraph *object*

Description

extract igraph object

Usage

```
graph(x, ...)

## S4 method for signature 'SurfaceGeometry'
graph(x)

## S4 method for signature 'NeuroSurface'
graph(x, ...)

## S4 method for signature 'NeuroSurfaceVector'
graph(x, ...)
```

Arguments

x	the object to extract the graph from
...	extra args

Value

An igraph object representing the surface mesh connectivity

indices,ROISurface-method *Extract Vertex Indices*

Description

Extracts the vertex indices from surface objects.

Usage

```
## S4 method for signature 'ROISurface'
indices(x)

## S4 method for signature 'ROISurfaceVector'
indices(x)
```

```
## S4 method for signature 'NeuroSurfaceVector'
indices(x)

## S4 method for signature 'NeuroSurface'
indices(x)
```

Arguments

x	the object to extract indices from

Value

An integer vector of vertex indices

LabeledNeuroSurface-class *LabeledNeuroSurface Class*

Description

Represents a 3D surface with labeled vertices, extending NeuroSurface.

Details

The LabeledNeuroSurface class provides a way to represent anatomical parcellations or other categorical divisions of a brain surface. Each vertex is assigned a label based on its data value, which typically represents a region ID. The class maintains a mapping between these IDs and human-readable labels, along with colors for visualization.

This is particularly useful for displaying anatomical atlases or the results of clustering algorithms on the brain surface.

Value

An object of class `LabeledNeuroSurface`.

Slots

`labels` Character vector of label annotations

`cols` Character vector of hex color codes for labels

See Also

NeuroSurface

Examples

```
# Create a simple tetrahedron mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
               mesh = mesh,
               graph = graph,
               hemi = "left")

# Define indices for all vertices
indices <- 1:4

# Create data values for each vertex
vertex_data <- c(1, 2, 1, 3)  # Values representing region IDs

# Define the unique labels for the regions
labels <- c("Region A", "Region B", "Region C")

# Define colors for each label (in the same order as labels)
colors <- c("#FF0000", "#00FF00", "#0000FF")  # Red, Green, Blue

# Create the LabeledNeuroSurface object
labeled_surface <- new("LabeledNeuroSurface",
                      geometry = geometry,
                      indices = indices,
                      data = vertex_data,
                      labels = labels,
                      cols = colors)
```

```
# In this example, vertices are labeled as follows:
# - Vertices 1 and 3 belong to "Region A" (Red)
# - Vertex 2 belongs to "Region B" (Green)
# - Vertex 4 belongs to "Region C" (Blue)
```

`laplacian` *Compute Graph Laplacian*

Description

Compute Graph Laplacian

Usage

```
laplacian(x, normalized, weights, ...)

## S4 method for signature 'SurfaceGeometry,missing,missing'
laplacian(x)

## S4 method for signature 'SurfaceGeometry,missing,numeric'
laplacian(x, weights)
```

Arguments

`x`	Object to compute Laplacian from
`normalized`	Logical; whether to normalize the Laplacian
`weights`	Edge weights for weighted Laplacian matrix
`...`	Additional arguments

Value

A sparse Laplacian matrix of class `dgCMatrix`

`left` *Get Left Hemisphere*

Description

Get Left Hemisphere

Usage

```
left(x)

## S4 method for signature 'BilatNeuroSurfaceVector'
left(x)
```

Arguments

x	Surface object

Value

Left hemisphere of the surface

Examples

```
# Requires bilateral surface data
# lh <- left(bilat_surface)
```

---

length,ROISurface-method

Get Length of Surface Object

Description

Returns the number of vertices in a surface object.

Usage

```
## S4 method for signature 'ROISurface'
length(x)
```

Arguments

x	the object to extract the length from

Value

An integer giving the number of vertices

load_data,NeuroSurfaceVectorSource-method *load_data*

Description

Loads surface geometry data from a source object.

Usage

```
## S4 method for signature 'NeuroSurfaceVectorSource'
load_data(x)

## S4 method for signature 'NeuroSurfaceSource'
load_data(x)

## S4 method for signature 'FreesurferSurfaceGeometryMetaInfo'
load_data(x)

## S4 method for signature 'GIFTISurfaceGeometryMetaInfo'
load_data(x)

## S4 method for signature 'SurfaceGeometrySource'
load_data(x)
```

Arguments

x	the object to load data from

Value

A `SurfaceGeometry` object containing the loaded mesh data

load_fsaverage *Fetch fsaverage surfaces*

Description

This is a high-level wrapper that mirrors the API of neuromaps' `fetch_fsaverage()` but is currently limited to the `"std.8"` decimated fsaverage surfaces that ship with neurosurf. The function name uses "load" rather than "fetch" to follow common R idioms.

Usage

```
load_fsaverage(
  density = "std.8",
  surf = c("smoothwm", "pial", "inflated", "white", "sphere")
)
```

Arguments

density	Character string specifying surface density. At present only "std.8" is supported.
surf	Character string specifying which surface to load. One of "smoothwm", "pial", "inflated", "white", or "sphere". Defaults to "smoothwm".

Value

A named list with elements "lh" and "rh", each a SurfaceGeometry instance.

Examples

```
fs <- load_fsaverage(density = "std.8", surf = "inflated")
if (interactive()) {
  show_surface_plot(fs$lh, fs$rh, views = c("lateral", "medial"))
}
```

---

load_fsaverage_bundle *Load a bundle of fsaverage surface variants as a SurfaceSet*

Description

Load a bundle of fsaverage surface variants as a SurfaceSet

Usage

```
load_fsaverage_bundle(
  density = "std.8",
  surfs = c("smoothwm", "pial", "inflated", "white", "sphere"),
  default_label = "smoothwm"
)
```

Arguments

density	Surface density; currently only \"std.8\" is supported.
surfs	Character vector of surface labels to include (e.g., c(\"smoothwm\",\"pial\",\"inflated\",\"white
default_label	Which label to treat as default when none is specified.

Value

A named list with elements \"lh\" and \"rh\", each a SurfaceSet containing the requested variants.

Examples

```
bundle <- load_fsaverage_bundle()
lh_set <- bundle$lh
rh_set <- bundle$rh
```

---

load_fsaverage_std8 *Load fsaverage std.8 surfaces packaged with neurosurf*

Description

This convenience helper loads the FreeSurfer fsaverage surfaces that ship with neurosurf (the std.8 decimated variant) and returns them as SurfaceGeometry objects.

Usage

```
load_fsaverage_std8(
  surf = c("smoothwm", "pial", "inflated", "white", "sphere")
)
```

Arguments

surf	Character string specifying which surface to load. One of "smoothwm", "pial", "inflated", "white", or "sphere". Defaults to "smoothwm".

Value

A named list with elements "lh" and "rh", each a SurfaceGeometry instance for the requested surface type.

Examples

```
fs <- load_fsaverage_std8("smoothwm")
if (interactive()) {
  show_surface_plot(fs$lh, fs$rh, views = c("lateral", "medial"))
}
```

loadFSSurface *load Freesurfer ascii surface*

Description

load Freesurfer ascii surface

Usage

```
loadFSSurface(meta_info, surf_to_world = diag(4))
```

Arguments

meta_info	instance of type FreesurferSurfaceGeometryMetaInfo
surf_to_world	Optional 4x4 affine transformation matrix from surface (tkr-RAS) coordinates to world (scanner RAS) coordinates. Defaults to identity. For proper alignment with volumes, this transform can be computed from FreeSurfer's vox2ras-tkr matrix (available in orig.mgz or via mri_info –vox2ras-tkr).

Details

requires rgl library

Value

a class of type SurfaceGeometry

Examples

```
# Requires FreeSurfer surface file
# meta <- read_meta_info(FreesurferAsciiSurfaceFileDescriptor(), "lh.pial.asc")
# geom <- loadFSSurface(meta)
```

loadGIFTISurface *Load GIFTI surface geometry*

Description

Loads a GIFTI (`.surf.gii`) surface into a SurfaceGeometry. The usual public entry point for reading a surface from disk is read_surf_geometry / read_surf; this function is the GIFTI-specific loader it dispatches to. For convenience it also accepts a file path directly, in which case the header is read internally.

Usage

```
loadGIFTISurface(meta_info)
```

Arguments

meta_info	either a GIFTISurfaceGeometryMetaInfo instance, or a length-one character path to a .surf.gii / .gii file.

Details

requires rgl library

Value

a class of type SurfaceGeometry

Examples

```
# Either pass a path directly ...
# geom <- loadGIFTISurface("lh.midthickness.surf.gii")
# ... or go through the meta-info object:
# meta <- read_meta_info(neurosurf:::GIFTI_SURFACE_DSET, "lh.midthickness.surf.gii")
# geom <- loadGIFTISurface(meta)
```

---

map_values,NeuroSurface,list-method

Map Values for NeuroSurface with List Lookup

Description

Map Values for NeuroSurface with List Lookup

Usage

```
## S4 method for signature 'NeuroSurface,list'
map_values(x, lookup)
```

Arguments

x	a NeuroSurface object
lookup	a list of values to map

Value

A NeuroSurface object with remapped values

map_values,NeuroSurface,matrix-method
Map Values for NeuroSurface with Matrix Lookup

Description

Map Values for NeuroSurface with Matrix Lookup

Usage

```
## S4 method for signature 'NeuroSurface,matrix'
map_values(x, lookup)
```

Arguments

x	a NeuroSurface object
lookup	a matrix with two columns: the first column is the key, and the second column is the value

Value

A NeuroSurface object with remapped values

---

meshToGraph *Construct a Graph from Mesh Vertices and Faces*

Description

This function creates an igraph object representing the connectivity structure of a 3D mesh based on its vertices and triangular faces.

Usage

```
meshToGraph(vertices, nodes)
```

Arguments

vertices	A numeric matrix with 3 columns representing the x, y, and z coordinates of vertices. Each row corresponds to a vertex.
nodes	A numeric matrix where each row represents a triangular face, containing 0-based indices of three vertices that form the face.

Details

The function converts a triangular mesh into a graph representation where:

- Vertices of the graph correspond to vertices of the mesh
- Edges of the graph correspond to the edges of triangular faces in the mesh

The function performs the following steps:

1. Extracts all unique edges from the triangular faces
2. Creates an undirected graph from these edges
3. Simplifies the graph to remove duplicate edges and loops
4. Calculates Euclidean distances between connected vertices
5. Adds vertex coordinates and edge distances as attributes to the graph

Note that the input `nodes` matrix should use 0-based indexing (starting from 0), as the function will increment indices by 1 when creating the graph.

Value

An `igraph` object representing the mesh connectivity. The graph has the following attributes:

- Vertex attributes: "x", "y", and "z" coordinates from the vertices matrix
- Edge attribute: "dist" (Euclidean distance between connected vertices)

See Also

SurfaceGeometry, graph_from_edgelist

Examples

```
# Create a simple cube mesh with 8 vertices
vertices <- matrix(c(
  0, 0, 0,  # vertex 1
  1, 0, 0,  # vertex 2
  1, 1, 0,  # vertex 3
  0, 1, 0,  # vertex 4
  0, 0, 1,  # vertex 5
  1, 0, 1,  # vertex 6
  1, 1, 1,  # vertex 7
  0, 1, 1   # vertex 8
), ncol = 3, byrow = TRUE)

# Define triangular faces with 0-based indices
faces <- matrix(c(
  # bottom face (z=0)
  0, 1, 2,
  0, 2, 3,
  # top face (z=1)
  4, 5, 6,
  4, 6, 7,
```

```
  # front face (y=0)
  0, 1, 5,
  0, 5, 4,
  # back face (y=1)
  2, 3, 7,
  2, 7, 6,
  # left face (x=0)
  0, 3, 7,
  0, 7, 4,
  # right face (x=1)
  1, 2, 6,
  1, 6, 5
), ncol = 3, byrow = TRUE)

# Create the graph representation of the mesh
graph <- meshToGraph(vertices, faces)

# Examine the graph properties
cat("Number of vertices:", igraph::vcount(graph), "\n")
cat("Number of edges:", igraph::ecount(graph), "\n")

# Plot the graph if igraph is available
if (interactive() && requireNamespace("igraph", quietly = TRUE) &&
    requireNamespace("rgl", quietly = TRUE)) {
  # First visualize the mesh
  rgl::open3d()
  mesh <- rgl::tmesh3d(
    vertices = t(vertices),
    indices = t(faces) + 1,  # rgl uses 1-based indexing
    homogeneous = FALSE
  )
  rgl::shade3d(mesh, col = "lightblue")
  rgl::title3d(main = "Original Mesh")

  # Visualize the graph connections using the 3D coordinates
  rgl::open3d()
  # Plot vertices
  rgl::points3d(vertices[,1], vertices[,2], vertices[,3], size = 10, col = "red")
  # Plot edges
  edges <- igraph::as_edgelist(graph)
  for (i in 1:nrow(edges)) {
    v1 <- edges[i, 1]
    v2 <- edges[i, 2]
    coords <- vertices[c(v1, v2), ]
    rgl::lines3d(coords[,1], coords[,2], coords[,3], col = "black", lwd = 2)
  }
  rgl::title3d(main = "Graph Representation")
}
```

neighbor_graph *Construct Neighborhood Graph from Surface Mesh*

Description

This generic function constructs a neighborhood graph from a surface mesh using edge weights. It allows for flexible definition of neighborhoods based on edge radius and custom edge weights.

Usage

```
neighbor_graph(x, radius, edgeWeights = missing(), nodes = missing(), ...)

## S4 method for signature 'igraph,numeric,missing,missing'
neighbor_graph(
  x,
  radius,
  edgeWeights = NULL,
  nodes = NULL,
  distance_type = c("geodesic", "euclidean", "spherical")
)

## S4 method for signature 'SurfaceGeometry,numeric,missing,missing'
neighbor_graph(
  x,
  radius,
  edgeWeights = NULL,
  nodes = NULL,
  distance_type = c("geodesic", "euclidean", "spherical")
)

## S4 method for signature 'SurfaceGeometry,numeric,numeric,missing'
neighbor_graph(
  x,
  radius,
  edgeWeights,
  distance_type = c("geodesic", "euclidean", "spherical")
)

## S4 method for signature 'SurfaceGeometry,numeric,numeric,integer'
neighbor_graph(
  x,
  radius,
  edgeWeights,
  nodes,
  distance_type = c("geodesic", "euclidean", "spherical")
)
```

```
## S4 method for signature 'SurfaceGeometry,numeric,missing,integer'
neighbor_graph(
  x,
  radius,
  nodes,
  distance_type = c("geodesic", "euclidean", "spherical")
)
```

Arguments

x	An object representing a surface mesh. The specific class depends on the method implementation.
radius	Numeric. The edge radius defining the neighborhood extent.
edgeWeights	Numeric vector. Custom weights for edges, used to define edge distances.
nodes	Integer vector. The subset of nodes to use in graph construction. If NULL, all nodes are used.
...	Additional arguments passed to methods.
distance_type	the type of distance metric to use

Details

The neighborhood graph is constructed by considering edges within the specified radius and applying the provided edge weights. This function is particularly useful in neuroimaging analyses for defining local connectivity on brain surfaces.

Value

An object representing the constructed neighborhood graph. The specific class depends on the method implementation.

See Also

graph, vertices

Examples

```
geom <- example_surface_geometry()
graph <- neighbor_graph(geom, radius = 1)
igraph::vcount(graph)



# Build a neighbor graph from a simple SurfaceGeometry
geom <- example_surface_geometry()
g_geom <- neighbor_graph(geom, radius = 1)
igraph::vcount(g_geom)
```

neurosurf *neurosurf: Data structures and IO for surface-based neuroimaging data.*

Description

Data structures and IO for surface-based neuroimaging data.

Author(s)

Maintainer: Bradley R Buchsbaum <brad.buchsbaum@gmail.com> [copyright holder]

See Also

Useful links:

- https://github.com/bbuchsbaum/neurosurf
- Report bugs at https://github.com/bbuchsbaum/neurosurf/issues

neurosurf_download_testdata *Download optional test data for neurosurf*

Description

Downloads larger test data files that are not included in the CRAN package due to size constraints. These files are hosted on GitHub releases and are useful for running the full test suite or exploring additional file formats.

Usage

```
neurosurf_download_testdata(
  files = "all",
  destdir = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

files	Character vector of files to download. Use "all" to download all available test files. Available files: • "rscan01_lh.gii" - GIFTI surface file (30 MB) • "rscan01_lh.niml.dset" - NIML surface dataset (22 MB) • "rscan01_rh.niml.dset" - NIML surface dataset (22 MB)

destdir	Destination directory. Defaults to extdata within the installed package location. If the package location is not writable, defaults to rappdirs::user_cache_dir("neurosurf").
overwrite	Logical; if TRUE, re-download files even if they already exist. Default is FALSE.
quiet	Logical; if TRUE, suppress progress messages.

Details

The test data files are hosted on GitHub releases for the neurosurf package. This function requires an internet connection to download the files.

If you are running tests locally and want the full test suite, run neurosurf_download_testdata("all") once after installing the package.

Value

Invisibly returns the paths to downloaded files.

Examples

```
# Download all test data
neurosurf_download_testdata("all")

# Download only the GIFTI file
neurosurf_download_testdata("rscan01_lh.gii")
```

NeuroSurface *Construct a NeuroSurface Object*

Description

This function creates a new NeuroSurface object, which represents a single set of data values associated with nodes on a surface geometry.

Usage

```
NeuroSurface(geometry, indices, data)
```

Arguments

geometry	A SurfaceGeometry or SurfaceSet object representing the underlying surface structure.
indices	An integer vector specifying the indices of valid surface nodes.
data	A numeric vector of data values corresponding to the surface nodes.

Details

The NeuroSurface object is designed to store and manipulate a single set of data values associated with nodes on a surface. This can represent various neuroimaging measures such as cortical thickness, functional activation, or any other node-wise metric.

The length of the `data` vector should match the length of the `indices` vector. Nodes not included in the `indices` vector are considered to have no data or to be invalid.

Value

A new object of class `NeuroSurface` containing:

`geometry`	The input `SurfaceGeometry` object.
`indices`	The input integer vector of valid node indices.
`data`	The input numeric vector of data values.

See Also

`SurfaceGeometry`, `NeuroSurfaceVector`

Examples

```
surf_geom <- example_surface_geometry()
indices <- seq_len(nrow(coords(surf_geom)))  # all vertices
data_values <- rnorm(length(indices))        # Example data
neuro_surf <- NeuroSurface(surf_geom, indices, data_values)
```

NeuroSurface-class *NeuroSurface*

Description

a three-dimensional surface consisting of a set of triangle vertices with one value per vertex.

Details

The NeuroSurface class is a fundamental representation of surface-based neuroimaging data. It combines geometric information about a brain surface with data values mapped to each vertex of that surface.

The class consists of three core components:

- `geometry`: The underlying 3D surface structure, containing vertex coordinates, face definitions, and topological information
- `indices`: Identifiers for the subset of vertices in the geometry that have associated data values
- `data`: A numeric vector containing one value per vertex, representing measurements such as cortical thickness, functional activation, or any other surface-mapped metric

This class serves as the foundation for more specialized surface representations like `ColorMappedNeuroSurface`, `VertexColoredNeuroSurface`, and `LabeledNeuroSurface`. It facilitates common operations such as visualization, statistical analysis, and spatial processing of surface-based neuroimaging data.

Value

An object of class `NeuroSurface`.

Slots

`geometry` the surface geometry, an instance of `SurfaceGeometry` or `SurfaceSet`

`indices` an `integer` vector specifying the subset of valid surface nodes encoded in the `geometry` object.

`data` the 1-D vector of data value at each vertex of the mesh

Examples

```
# Create a simple tetrahedron mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define indices for all vertices
indices <- 1:4
```

```
# Create data values for each vertex
vertex_data <- c(0.5, 1.2, 0.8, 1.5)  # example values for the vertices

# Create the NeuroSurface object
neuro_surface <- new("NeuroSurface",
                     geometry = geometry,
                     indices = indices,
                     data = vertex_data)

# The data values are now mapped to the surface vertices
# and can be visualized or analyzed
```

`NeuroSurfaceSource-class`

NeuroSurfaceSource Class

Description

The 'NeuroSurfaceSource' class serves as a factory for creating `NeuroSurface` instances. It encapsulates all necessary information to construct a neuroimaging surface, including geometry, metadata, and indexing information.

Usage

```
NeuroSurfaceSource(
  surface_geom,
  surface_data_name,
  colind = NULL,
  nodeind = NULL
)
```

Arguments

`surface_geom`	the name of the file containing the surface geometry or a `SurfaceGeometry` instance
`surface_data_name`	the name of the file containing the data values to be mapped to the surface.
`colind`	the subset of column indices to load from surface data matrix (if provided)
`nodeind`	the subset of node indices to load from surface data matrix (if provided)

Details

This class is designed to facilitate the creation of `NeuroSurface` objects by providing a standardized way to store and access all required components. It combines geometric information, metadata, and indexing details necessary for constructing a complete neuroimaging surface representation.

Value

An object of class NeuroSurfaceSource or NeuroSurfaceVectorSource that contains information about the surface geometry and associated data. If the data has multiple columns (colind > 1), a NeuroSurfaceVectorSource is returned; otherwise, a NeuroSurfaceSource is returned. These objects can be used to load and map neuroimaging data onto brain surfaces.

Slots

`geometry` An object of class SurfaceGeometry representing the underlying surface structure.

`data_meta_info` An object of class SurfaceDataMetaInfo containing metadata about the surface data.

`colind` An integer specifying the column index of the surface map to be loaded.

`nodeind` An integer vector specifying the node indices of the surface map to be loaded.

See Also

NeuroSurface, SurfaceGeometry, SurfaceDataMetaInfo

Examples

```
# Create a simple mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")
```

```
# Create a SurfaceDataMetaInfo object
data_meta_info <- new("SurfaceDataMetaInfo",
                      header_file = "data_meta.txt",
                      data_file = "surface_data.1D",
                      file_descriptor = new("FileFormat"),
                      node_count = 4L,
                      nels = 1L,
                      label = "thickness")

# Create a NeuroSurfaceSource object
neuro_source <- new("NeuroSurfaceSource",
                    geometry = geometry,
                    data_meta_info = data_meta_info,
                    colind = 1L,
                    nodeind = 1:4)
```

NeuroSurfaceVector *NeuroSurfaceVector*

Description

construct a new NeuroSurfaceVector

Usage

```
NeuroSurfaceVector(geometry, indices, mat)
```

Arguments

geometry	a SurfaceGeometry or SurfaceSet instance
indices	an integer vector specifying the valid surface nodes.
mat	a matrix of data values (rows=nodes, columns=variables)

Value

A NeuroSurfaceVector object containing the geometry, node indices, and data matrix.

`NeuroSurfaceVector-class`
NeuroSurfaceVector Class

Description

Represents a 3D surface with multiple values per vertex.

Details

The NeuroSurfaceVector class extends the concept of NeuroSurface to handle multiple measurements for each vertex across the entire surface. Unlike NeuroSurface which stores a single value per vertex, NeuroSurfaceVector stores a matrix of values where columns represent different measures and rows correspond to vertices.

This structure is particularly useful for:

- Time series data where each column represents a different timepoint
- Multi-modal data where each column represents a different imaging modality
- Statistical results where columns represent different statistical parameters
- Feature vectors for machine learning applications

The data matrix organization (vertices as rows, measures as columns) facilitates efficient vertex-wise operations and analyses. This is in contrast to ROISurfaceVector where the matrix is transposed (measures as rows, vertices as columns).

Value

An object of class `NeuroSurfaceVector`

Slots

`geometry` SurfaceGeometry instance representing the surface structure

`indices` Integer vector of valid surface node indices

`data` Matrix of values, where columns represent different measures and rows correspond to surface nodes

Note

The number of rows in 'data' must match the number of nodes in 'geometry'.

See Also

`SurfaceGeometry`, `NeuroSurface`

Examples

```
# Create a simple tetrahedron mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define indices for all vertices
indices <- 1:4

# Create a Matrix with multiple measures for each vertex
# Each row corresponds to a vertex, each column to a different measure
require(Matrix)
vertex_data <- Matrix(
  c(0.5, 1.2, 0.8,    # Measure 1 values for vertices 1-4
    0.7, 0.3, 1.5, 0.9,  # Measure 2 values for vertices 1-4
    1.1, 0.6, 0.4, 1.3), # Measure 3 values for vertices 1-4
  nrow = 4, ncol = 3,
  byrow = FALSE
)

# Create the NeuroSurfaceVector object
neuro_surface_vector <- new("NeuroSurfaceVector",
                            geometry = geometry,
                            indices = indices,
                            data = vertex_data)
```

```
# The data matrix now maps multiple values to each surface vertex
# Vertex 1 has values: 0.5, 0.7, 1.1
# Vertex 2 has values: 1.2, 0.3, 0.6
# etc.
```

NeuroSurfaceVectorSource-class
NeuroSurfaceVectorSource

Description

A class that is used to produce a NeuroSurfaceVector instance

Value

An object of class NeuroSurfaceVectorSource.

Slots

geometry a SurfaceGeometry instance

data_meta_info a SurfaceDataMetaInfo instance

colind the column indices vector of the surface maps to be loaded

---

NIMLSurfaceDataMetaInfo-class
NIMLSurfaceDataMetaInfo

Description

This class contains meta information for surface-based data for the NIML data format

Value

An object of class NIMLSurfaceDataMetaInfo.

Slots

data the numeric data matrix of surface values (rows = nodes, columns=surface vectors)

node_indices the indices of the nodes for mapping to associated surface geometry.

Examples

```
meta_info <- new("SurfaceDataMetaInfo",
                 header_file = "data_header.txt",
                 data_file = "surface_data.niml.dset",
                 file_descriptor = new("FileFormat"),
                 node_count = length(nodes(geometry)),
                 nels = 2L,
                 label = "thickness")

surface_data <- matrix(rnorm(meta_info@node_count * meta_info@nels),
                       nrow = meta_info@node_count, ncol = meta_info@nels)

niml_meta <- new("NIMLSurfaceDataMetaInfo",
                header_file = meta_info@header_file,
                data_file = meta_info@data_file,
                file_descriptor = meta_info@file_descriptor,
                node_count = meta_info@node_count,
                nels = meta_info@nels,
                label = meta_info@label,
                data = surface_data,
                node_indices = seq_len(meta_info@node_count))
```

NIMLSurfaceFileDescriptor-class *NIMLSurfaceFileDescriptor*

Description

This class supports the NIML file format for surface-based data

Value

An object of class `NIMLSurfaceFileDescriptor`.

---

nodes *Extract Surface Node Numbers*

Description

Retrieves the node numbers from a surface object.

Usage

```
nodes(x)

## S4 method for signature 'SurfaceGeometry'
nodes(x)

## S4 method for signature 'NeuroSurface'
nodes(x)

## S4 method for signature 'NeuroSurfaceVector'
nodes(x)
```

Arguments

x	An object representing a surface.

Value

A vector of node numbers.

See Also

vertices

Examples

```
geom <- example_surface_geometry()
nodes(geom)
```

---

parcel_boundary_contact

Parcel boundary contact matrix

Description

Parcel boundary contact matrix

Usage

```
parcel_boundary_contact(
  labeled_surface,
  component_policy = c("error", "largest", "each", "merge"),
  counts = FALSE
)
```

Arguments

labeled_surface	A LabeledNeuroSurface.
component_policy	Fragment handling policy.
counts	If TRUE, return edge counts; otherwise logical contact matrix.

Value

Matrix (logical or integer) indicating parcel-unit boundary contacts.

Examples

```
# Requires a LabeledNeuroSurface
# lsurf <- read_freesurfer_annot("lh.aparc.annot", geom)
# contact <- parcel_boundary_contact(lsurf)
```

---

parcel_geodesic_centroid

Parcel centroids using geodesic medoids

Description

Parcel centroids using geodesic medoids

Usage

```
parcel_geodesic_centroid(
  labeled_surface,
  method = c("medoid", "euclidean"),
  component_policy = c("error", "largest", "each", "merge"),
  weights = NULL,
  chunk_size = 2000,
  cache = TRUE
)
```

Arguments

labeled_surface	A LabeledNeuroSurface.
method	"medoid" (geodesic mean distance) or "euclidean" (closest to Euclidean centroid).
component_policy	How to handle fragmented parcels: "error", "largest", "each", "merge".

weights	Optional numeric edge weights (defaults to E(g)$dist).
chunk_size	Number of source vertices per Dijkstra batch.
cache	Logical; cache symmetric results when targets is NULL or identical to vertices.

Value

Data frame with one row per parcel unit: unit, parcel_id, component, size, centroid_vertex, and Cartesian coordinates.

Examples

```
# Requires a LabeledNeuroSurface
# lsurf <- read_freesurfer_annot("lh.aparc.annot", geom)
# centroids <- parcel_geodesic_centroid(lsurf)
```

---

parcel_geodesic_distance_matrix

Parcel-to-parcel geodesic distances

Description

Parcel-to-parcel geodesic distances

Usage

```
parcel_geodesic_distance_matrix(
  labeled_surface,
  metric = c("centroid", "min"),
  component_policy = c("error", "largest", "each", "merge"),
  weights = NULL,
  chunk_size = 2000,
  cache = TRUE
)
```

Arguments

labeled_surface	A LabeledNeuroSurface.
metric	"centroid" (distance between parcel medoids) or "min" (minimum distance between any vertices of two parcels).
component_policy	Fragment handling policy.
weights	Optional numeric edge weights (defaults to E(g)$dist).
chunk_size	Number of source vertices per Dijkstra batch.
cache	Logical; cache symmetric results when targets is NULL or identical to vertices.

Value

A numeric matrix with parcel-unit names as dimnames.

Examples

```
# Requires a LabeledNeuroSurface
# lsurf <- read_freesurfer_annot("lh.aparc.annot", geom)
# dmat <- parcel_geodesic_distance_matrix(lsurf)
```

---

plot,SurfaceGeometry,missing-method
Plot a Surface

Description

Renders a surface object using the interactive viewer.

Usage

```
## S4 method for signature 'SurfaceGeometry,missing'
plot(
  x,
  vals = NA,
  cmap = grDevices::gray(seq(0, 1, length.out = 255)),
  vert_clrs = NULL,
  irange = range(vals),
  thresh = c(0, 0),
  alpha = 1,
  specular = "black",
  bgcol = "lightgray",
  ...
)

## S4 method for signature 'NeuroSurface,missing'
plot(
  x,
  cmap = grDevices::gray(seq(0, 1, length.out = 255)),
  vert_clrs = NULL,
  irange = range(x@data, na.rm = TRUE),
  thresh = c(0, 0),
  alpha = 1,
  specular = "black",
  bgcol = "lightgray",
  ...
)
```

```
## S4 method for signature 'LabeledNeuroSurface,missing'
plot(
  x,
  cmap = x@cols,
  vert_clrs = NULL,
  irange = range(x@data, na.rm = TRUE),
  thresh = c(0, 0),
  alpha = 1,
  specular = "black",
  bgcol = "lightgray",
  ...
)

## S4 method for signature 'ColorMappedNeuroSurface,missing'
plot(
  x,
  vert_clrs = NULL,
  alpha = 1,
  specular = "black",
  bgcol = "lightgray",
  ...
)

## S4 method for signature 'VertexColoredNeuroSurface,missing'
plot(x, alpha = 1, specular = "black", bgcol = "lightgray", ...)
```

Arguments

x	the surface to plot
vals	An optional numeric vector containing data values for each vertex on the surface. If provided and ‘vert_clrs‘ is NULL, these values are mapped to colors using ‘cmap‘ and ‘irange‘.
cmap	A vector of colors (e.g., hex codes) defining the color map used when ‘vals‘ is provided and ‘vert_clrs‘ is NULL. Defaults to ‘rainbow(256)‘.
vert_clrs	An optional character vector of hex color codes for each vertex. If provided, these colors directly override any coloring derived from ‘vals‘ and ‘cmap‘. The length should match the number of vertices in ‘surfgeom‘.
irange	An optional numeric vector of length 2, ‘c(min, max)‘. Specifies the range of ‘vals‘ to map onto the ‘cmap‘. Values outside this range will be clamped to the min/max colors. Defaults to the full range of ‘vals‘.
thresh	An optional numeric vector of length 2, ‘c(lower, upper)‘. Vertices with ‘vals‘ *outside* this range (i.e., ‘< lower‘ or ‘> upper‘) are made fully transparent. This is applied *after* the general ‘alpha‘. Defaults to NULL (no thresholding).
alpha	A numeric value between 0 (fully transparent) and 1 (fully opaque) controlling the overall transparency of the surface. Defaults to 1.

specular	The color of specular highlights on the surface, affecting its shininess. Can be a color name (e.g., "white") or hex code. Defaults to "black" for a matte look. Set to a brighter colour for a glossier appearance.
bgcol	A single hex color code or a vector of hex color codes used as the base color for the surface. If ‘vals‘ or ‘vert_clrs‘ are provided, this color is blended with the data/vertex colors. Defaults to "lightgray".
...	extra args to send to `view_surface`

Value

An htmlwidget object for interactive visualization

---

plot.neurosurf_plot *Plot method for neurosurf_plot objects*

Description

This is a convenience wrapper that renders a multi-panel surface layout and draws it to a new grid device.

Usage

```
## S3 method for class 'neurosurf_plot'
plot(x, ...)
```

Arguments

x	A "neurosurf_plot" object.
...	Additional arguments passed to `draw_surface_plot`.

Value

Invisibly returns the input `neurosurf_plot` object.

---

plot.SurfaceGeometry *Plot method for SurfaceGeometry objects*

Description

Plot method for SurfaceGeometry objects

Usage

```
## S3 method for class 'SurfaceGeometry'
plot(x, y, ...)
```

Arguments

x	A SurfaceGeometry object.
y	Ignored (for S3 method compatibility).
...	Additional arguments passed to view_surface.

Value

Invisibly returns the object ID(s) from the RGL scene.

Examples

```
geom <- example_surface_geometry()
if (interactive()) {
  plot(geom)
}
```

---

plot.SurfaceSet *Plot method for SurfaceSet objects*

Description

Plot method for SurfaceSet objects

Usage

```
## S3 method for class 'SurfaceSet'
plot(x, y, label = NULL, ...)
```

Arguments

x	A SurfaceSet.
y	Ignored (for S3 compatibility).
label	Optional surface label to display; defaults to the set's default.
...	Additional arguments passed to view_surface.

Value

Invisibly returns the object ID(s) from the RGL scene.

Examples

```
geom <- example_surface_geometry()
ss <- surface_set(inflated = geom)
if (interactive()) {
  plot(ss)
}
```

---

plot_js *Plot Surface as an HTMLWidget*

Description

Plot Surface as an HTMLWidget

Usage

```
plot_js(x, width = NULL, height = NULL, ...)

## S4 method for signature 'SurfaceGeometry'
plot_js(x, width = NULL, height = NULL, ...)
```

Arguments

x	Surface object to plot
width	The width of the widget (optional)
height	The height of the widget (optional)
...	Additional arguments passed to the plotting function

Value

An HTMLWidget object

Examples

```
geom <- example_surface_geometry()
widget <- plot_js(geom)
```

print.Searchlight *Print Method for Searchlight Iterator*

Description

Print Method for Searchlight Iterator

Usage

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

Arguments

x	An object of class "Searchlight"
...	Additional arguments (not used)

Value

Invisibly returns the Searchlight object (called for side effect).

projectCoordinates *Project 3D Coordinates onto a Surface and Smooth the Values*

Description

This function projects a set of 3D coordinates onto a given surface and creates a NeuroSurface object with the smoothed values. The projection is performed by finding the closest points on the surface, and then a kernel density smoother is applied locally to produce the final values.

Usage

```
projectCoordinates(surfgeom, points, sigma = 5, ...)
```

Arguments

surfgeom	A SurfaceGeometry object representing the surface onto which the coordinates will be projected.
points	A numeric matrix with three columns (x, y, z) representing the 3D coordinates to be projected onto the surface.
sigma	A numeric value specifying the smoothing radius for the kernel density smoother. Default is 5.
...	Additional arguments passed to the smoothing function.

Details

The function first projects each 3D coordinate onto the closest point on the surface defined by `surfgeom`. The values at these projected points are then smoothed using a kernel density smoother, where the `sigma` parameter controls the extent of the smoothing. The result is a `NeuroSurface` object containing the smoothed values, suitable for further analysis or visualization.

Value

A `NeuroSurface` object with the smoothed values mapped onto the surface.

Examples

```
# Load a sample surface from the package
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surfgeom <- read_surf_geometry(surf_file)

# Get the surface coordinates
surf_coords <- coords(surfgeom)

# Create some sample 3D coordinates to project
# We'll use a subset of the surface vertices with small random offsets
set.seed(123)
sample_indices <- sample(1:nrow(surf_coords), 50)
sample_coords <- surf_coords[sample_indices, ] + matrix(rnorm(150, 0, 0.5), ncol = 3)

# Project these coordinates onto the surface
projected_surface <- projectCoordinates(surfgeom, sample_coords, sigma = 3)

# Check the result
vals <- series(projected_surface, indices(projected_surface))
max(vals)        # Maximum density value
sum(vals > 0)    # Number of vertices with non-zero values
```

RandomSurfaceSearchlight

Create a Random Searchlight iterator for surface mesh

Description

Creates an iterator that randomly samples searchlight regions on a surface mesh using geodesic distance to define regions.

Usage

```
RandomSurfaceSearchlight(
  surfgeom,
  radius = 8,
```

```
  nodeset = NULL,
  as_deflist = FALSE
)
```

Arguments

surfgeom	A SurfaceGeometry object representing the surface mesh.
radius	Numeric, radius of the searchlight as a geodesic distance in mm. Must be a positive numeric scalar.
nodeset	Integer vector, optional subset of surface node indices to use. If supplied, must contain more than one index.
as_deflist	Logical, whether to return a deflist object.

Details

On each call to nextElem, a set of surface nodes is returned. These nodes index into the vertices of the igraph instance. When as_deflist=TRUE, the random ordering of centers is fixed when the object is created. Use set.seed before construction for reproducible sequences.

Value

An iterator object of class "RandomSurfaceSearchlight".

Examples

```
file <- system.file("extdata", "std.8_lh.smoothwm.asc", package = "neurosurf")
geom <- read_surf(file)
searchlight <- RandomSurfaceSearchlight(geom, 12)
set.seed(42)
dl <- RandomSurfaceSearchlight(geom, 12, as_deflist=TRUE)
attr(dl[[1]], "center")
nodes <- searchlight$nextElem()
length(nodes) > 1
```

---

read_freesurfer_annot *Read Freesurfer Annotation File*

Description

Reads a Freesurfer annotation file and creates a LabeledNeuroSurface object.

Usage

```
read_freesurfer_annot(file_name, geometry)
```

Arguments

file_name	Character string; path to the '.annot' file
geometry	A SurfaceGeometry object representing the surface structure

Details

This function reads binary data from a FreeSurfer annotation file, which includes vertex labels, color information, and label names. It then constructs a LabeledNeuroSurface object using this information along with the provided surface geometry.

Value

A LabeledNeuroSurface object containing:

indices	Integer vector of vertex indices
data	Numeric vector of label codes
labels	Character vector of label names
cols	Character vector of label colors in hex format

Examples

```
# Read a FreeSurfer annotation file (requires actual annotation file)
geom <- example_surface_geometry()
# labeled_surface <- read_freesurfer_annot("lh.aparc.annot", geom)
```

---

read_meta_info *Read Meta Information*

Description

Generic function to read image meta info given a file and a FileFormat instance from **neuroim2**.

Usage

```
## S4 method for signature 'AFNISurfaceFileDescriptor'
read_meta_info(x, file_name)

## S4 method for signature 'NIMLSurfaceFileDescriptor'
read_meta_info(x, file_name)

## S4 method for signature 'FreesurferAsciiSurfaceFileDescriptor'
read_meta_info(x, file_name)

## S4 method for signature 'FreesurferBinarySurfaceFileDescriptor'
read_meta_info(x, file_name)
```

```
## S4 method for signature 'GIFTISurfaceFileDescriptor'
read_meta_info(x, file_name)
```

Arguments

x	the file descriptor object
file_name	the name of the file containing meta information.

Value

A meta info object containing header information from the surface file

A metadata information object describing the surface file structure

---

read_surf *Read Surface Data from a File*

Description

This function reads surface data from a file in one of the supported formats.

Usage

```
read_surf(
  surface_name,
  surface_data_name = NULL,
  colind = NULL,
  nodeind = NULL
)
```

Arguments

surface_name	the name of the file containing the surface geometry.
surface_data_name	the name of the file containing the values to be mapped to the surface (optional).
colind	the columns/samples to load (optional), only if surface_data_name is not NULL
nodeind	the subset of node indices to load

Details

The function supports reading surface data from various formats including:

- Freesurfer ASCII (.asc)
- Freesurfer binary
- GIFTI (.gii)
- NIML Surface Dataset (.niml.dset)

The format is determined automatically from the file extension.

Value

an instance of the class: SurfaceGeometry or NeuroSurface or NeuroSurfaceVector

Examples

```
# Find the path to the example surface file in the package
surf_file <- system.file("extdata", "std.8_lh.smoothwm.asc", package = "neurosurf")

# Check if the file exists
if (file.exists(surf_file)) {
  # Read the surface geometry (returns SurfaceGeometry for geometry-only files)
  surf <- read_surf(surf_file)

  # Display basic information about the surface
  print(surf)

  # Get vertex coordinates
  head(coords(surf))
}
```

---

read_surf_data *load surface data and link to* SurfaceGeometry

---

Description

load surface data and link to SurfaceGeometry

Usage

```
read_surf_data(geometry, surface_data_name, colind = NULL, nodeind = NULL)
```

Arguments

geometry	a SurfaceGeometry instance
surface_data_name	the name of the file containing the values to be mapped to the surface.
colind	the subset column indices of surface dataset to load (optional)
nodeind	the subset node indices of surface dataset to include (optional)

Value

an instance of the class NeuroSurface or NeuroSurfaceVector

Examples

```
# Load geometry and surface data file
# geom <- read_surf_geometry("lh.white")
# surf_data <- read_surf_data(geom, "lh.thickness")
```

`read_surf_data_seq` *Read Surface Data Sequence*

Description

Load one or more surface datasets for both left and right hemispheres.

Usage

```
read_surf_data_seq(leftGeometry, rightGeometry, leftDataNames, rightDataNames)
```

Arguments

`leftGeometry`	a `SurfaceGeometry` instance for the left hemisphere
`rightGeometry`	a `SurfaceGeometry` instance for the right hemisphere
`leftDataNames`	a `character` vector indicating names of left-hemisphere surface data files to be mapped to geometry.
`rightDataNames`	a `character` vector indicating names of right-hemisphere surface data files to be mapped to geometry.

Value

A list of `BilatNeuroSurfaceVector` objects, one per pair of data files

`read_surf_geometry` *Read Surface Geometry from File*

Description

This function loads a supported surface geometry from a file and returns a SurfaceGeometry object.

Usage

```
read_surf_geometry(surface_name)
```

Arguments

`surface_name`	A character string specifying the name of the file containing the surface geometry.

Details

This function supports loading surface geometries from the following file formats:

- Freesurfer ASCII (.asc)
- Freesurfer binary
- GIFTI (.gii)

The appropriate loader is automatically selected based on the file extension or content.

Value

A `SurfaceGeometry` object representing the loaded surface.

Examples

```
asc_path <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
lh_surface <- read_surf_geometry(asc_path)
```

---

remeshSurface *Remesh a SurfaceGeometry object*

Description

This function applies uniform remeshing to a SurfaceGeometry object, creating a new mesh with more regular face sizes and improved quality.

Usage

```
remeshSurface(surfgeom, voxel_size = 2, ...)
```

Arguments

`surfgeom`	A `SurfaceGeometry` object to be remeshed.
`voxel_size`	Numeric value specifying the target edge length in the remeshed output. Smaller values create finer meshes with more faces.
`...`	Additional arguments to pass to `Rvcg::vcgUniformRemesh`.

Details

Remeshing is a process that reconstructs the mesh to improve its quality and/or adjust its resolution. Uniform remeshing creates a new mesh where the edge lengths are approximately equal throughout the surface, which is often desirable for analysis and visualization.

The `voxel_size` parameter controls the resolution of the output mesh:

- Smaller values create denser meshes with more vertices and faces
- Larger values create coarser meshes with fewer vertices and faces

Common reasons to remesh a surface include:

- Simplifying high-resolution meshes for faster processing
- Creating more uniform triangle sizes for better numerical stability
- Preparing meshes for specific analyses that require regular structures
- Fixing mesh defects and improving the overall quality

This function uses the VCG library (via the Rvcg package) to perform the remeshing operation, which is a robust and widely-used algorithm for mesh processing.

Value

A new `SurfaceGeometry` object with the remeshed surface.

See Also

`SurfaceGeometry`, `vcgUniformRemesh`

Examples

```
# Create a simple cube mesh
vertices <- matrix(c(
  0, 0, 0,  # vertex 1
  1, 0, 0,  # vertex 2
  1, 1, 0,  # vertex 3
  0, 1, 0,  # vertex 4
  0, 0, 1,  # vertex 5
  1, 0, 1,  # vertex 6
  1, 1, 1,  # vertex 7
  0, 1, 1   # vertex 8
), ncol = 3, byrow = TRUE)

# Define faces (12 triangular faces making a cube)
# Note indices are 0-based
faces <- matrix(c(
  # bottom face (z=0)
  0, 1, 2,
  0, 2, 3,
  # top face (z=1)
  4, 5, 6,
  4, 6, 7,
```

```
  # front face (y=0)
  0, 1, 5,
  0, 5, 4,
  # back face (y=1)
  2, 3, 7,
  2, 7, 6,
  # left face (x=0)
  0, 3, 7,
  0, 7, 4,
  # right face (x=1)
  1, 2, 6,
  1, 6, 5
), ncol = 3, byrow = TRUE)

# Create the SurfaceGeometry object
surf_geom <- SurfaceGeometry(vertices, faces, "lh")

# Remesh with a coarse voxel size - fewer triangles
coarse_remesh <- remeshSurface(surf_geom, voxel_size = 0.5)

# Remesh with a fine voxel size - more triangles
fine_remesh <- remeshSurface(surf_geom, voxel_size = 0.2)

# Visualize the meshes if rgl is available
if(interactive() && requireNamespace("rgl", quietly = TRUE)) {
  # Original mesh
  rgl::open3d()
  rgl::shade3d(surf_geom@mesh, col = "red")
  rgl::title3d(main = "Original Mesh")

  # Coarse remesh
  rgl::open3d()
  rgl::shade3d(coarse_remesh@mesh, col = "green")
  rgl::title3d(main = "Coarse Remesh (voxel_size = 0.5)")

  # Fine remesh
  rgl::open3d()
  rgl::shade3d(fine_remesh@mesh, col = "blue")
  rgl::title3d(main = "Fine Remesh (voxel_size = 0.2)")

  # Compare the number of faces in each mesh
  cat("Original mesh faces:", ncol(surf_geom@mesh$it), "\n")
  cat("Coarse remesh faces:", ncol(coarse_remesh@mesh$it), "\n")
  cat("Fine remesh faces:", ncol(fine_remesh@mesh$it), "\n")
}
```

---

render_surface_plot *Render a neurosurf plot using rgl*

Description

Render a neurosurf plot using rgl

Usage

```
render_surface_plot(x, offscreen = TRUE, scale = c(2, 2), crop = TRUE)
```

Arguments

x	A "neurosurf_plot" object.
offscreen	Logical; if TRUE, rendering is performed with rgl.useNULL = TRUE so that plots can be captured as images. A real GL context is attempted first for better antialiasing.
scale	Numeric vector of length 2 giving a supersampling factor for the offscreen snapshot. Values above 1 render at higher resolution before downscaling for smoother edges. Defaults to c(2, 2).
crop	Logical; if TRUE, automatically crops away white/empty margins from each snapshot to avoid the "tiny brain" effect in grids.

Value

A list containing rendered panel images (with aspect ratios) and layout information. This is a low-level helper intended to be wrapped by higher-level figure drawing utilities.

See Also

surface_plot, add_surface_layer, view_surface

Examples

```
geom <- example_surface_geometry()
p <- surface_plot(geom)
if (interactive()) {
  rendered <- render_surface_plot(p)
}
```

---

right *Get Right Hemisphere*

Description

Get Right Hemisphere

Usage

```
right(x)

## S4 method for signature 'BilatNeuroSurfaceVector'
right(x)
```

Arguments

x	Surface object

Value

Right hemisphere of the surface

Examples

```
# Requires bilateral surface data
# rh <- right(bilat_surface)
```

ROISurface *Create an instance of class* ROISurface

Description

Create an instance of class ROISurface

Usage

```
ROISurface(geometry, indices, data)
```

Arguments

geometry	the parent geometry: an instance of class SurfaceGeometry
indices	the parent surface indices
data	the data values, numeric vector

Value

an instance of class ROISurface

Examples

```
verts <- matrix(c(0,0,0,
                  1,0,0,
                  0,1,0), ncol=3, byrow=TRUE)
faces <- matrix(c(0L,1L,2L), ncol=3, byrow=TRUE)
geom <- SurfaceGeometry(verts, faces, "lh")

ROISurface(geom, 1L, 1)

try(ROISurface(geom, 4L, 1))      # out of range
try(ROISurface(geom, 1.5, 1))     # non-integer
```

ROISurface-class *ROISurface*

Description

A class that represents a surface-based region of interest

Details

The ROISurface class provides a way to represent a specific subset of vertices on a brain surface along with their associated data values. This is particularly useful for analyzing or visualizing specific anatomical or functional regions on the cortical surface.

The class maintains a reference to the complete parent surface geometry while storing only the relevant subset of vertices, their coordinates, and their data values. The `indices` slot allows mapping back to the original vertex indices in the parent surface.

Typical use cases include:

- Extracting and analyzing data from anatomical regions of interest
- Working with functional clusters identified in analyses
- Isolating specific surface features for detailed investigation
- Statistical analysis of data within defined surface regions

Value

An object of class `ROISurface`.

Slots

`geometry` the geometry of the parent surface: a `SurfaceGeometry` instance

`data` the vector-valued `numeric` data stored in ROI

`coords` the surface-based coordinates of the data

`indices` the node indices of the parent surface stored in the `geometry` field.

Examples

```
# Create a simple tetrahedron mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define the ROI - just using vertices 1 and 2 as an example
roi_indices <- c(1L, 2L)

# Extract coordinates for these vertices
roi_coords <- matrix(
  c(0, 0, 0,  # coordinates for vertex 1
    1, 0, 0), # coordinates for vertex 2
  ncol = 3, byrow = TRUE
)

# Create data values for the ROI vertices
roi_data <- c(0.75, 1.25)  # example values for the two vertices

# Create the ROISurface object
roi <- new("ROISurface",
           geometry = geometry,
           data = roi_data,
           coords = roi_coords,
           indices = roi_indices)
```

ROISurfaceVector *Create an instance of class* ROISurfaceVector

Description

Create an instance of class ROISurfaceVector

Usage

```
ROISurfaceVector(geometry, indices, data)
```

Arguments

geometry	the parent geometry: an instance of class SurfaceGeometry
indices	the parent surface indices
data	the data values, a matrix

Value

an instance of class ROISurfaceVector

Examples

```
verts <- matrix(c(0,0,0,
                  1,0,0,
                  0,1,0), ncol=3, byrow=TRUE)
faces <- matrix(c(0L,1L,2L), ncol=3, byrow=TRUE)
geom <- SurfaceGeometry(verts, faces, "lh")

vec <- matrix(c(0.5, 1.5), nrow=1)
ROISurfaceVector(geom, c(1L,2L), vec)

try(ROISurfaceVector(geom, c(1L,4L), vec))   # out of range
try(ROISurfaceVector(geom, c(1,2.5), vec))   # non-integer
```

`ROISurfaceVector-class` *ROISurfaceVector*

Description

A class that represents a surface-based region of interest

Details

The ROISurfaceVector class extends the concept of ROISurface to handle multiple measurements for each vertex in the region of interest. While ROISurface stores a single value per vertex, ROISurfaceVector stores a matrix of values where each row represents a different measure and each column corresponds to a vertex.

This structure is particularly useful for multivariate analyses of specific brain regions, allowing researchers to:

- Store time series data for each vertex in an ROI
- Maintain multiple modalities of data for the same surface region
- Perform multivariate statistical analyses on regional surface data
- Compare different metrics within the same anatomical region

The data matrix is organized with rows representing different measures and columns representing different vertices, which facilitates efficient access to all measurements for a particular vertex or all vertices for a particular measurement.

Value

An object of class `ROISurfaceVector`.

Slots

`geometry` the geometry of the parent surface: a `SurfaceGeometry` instance

`data matrix` data stored in ROI with number of columns equal to number of coordinates in ROI.

`coords` the surface-based coordinates of the data

`indices` the nodes of the parent surface stored in the `geometry` field.

Examples

```
# Create a simple tetrahedron mesh for the example
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
```

```
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define the ROI - just using vertices 1 and 2 as an example
roi_indices <- c(1L, 2L)

# Extract coordinates for these vertices
roi_coords <- matrix(
  c(0, 0, 0,  # coordinates for vertex 1
    1, 0, 0), # coordinates for vertex 2
  ncol = 3, byrow = TRUE
)

# Create data matrix for the ROI vertices - 3 different measures
# Each row represents a different measure, each column a different vertex
roi_data <- matrix(
  c(0.75, 1.25,  # values for measure 1
    0.50, 0.80,  # values for measure 2
    0.30, 0.60), # values for measure 3
  nrow = 3, ncol = 2
)

# Create the ROISurfaceVector object
roi_vector <- new("ROISurfaceVector",
                  geometry = geometry,
                  data = roi_data,
                  coords = roi_coords,
                  indices = roi_indices)
```

`sampler_to_triplets` *Extract sparse matrix triplets from a surface sampler*

Description

Converts a precomputed 'surface_sampler' into sparse matrix triplet format (i, j, x) suitable for constructing a dgCMatrix or for interop with other packages (e.g., neurofunctor). The triplets represent a vertices × voxels projection matrix with normalized Gaussian weights.

Usage

```
sampler_to_triplets(
  sampler,
  sigma = NULL,
  normalize = TRUE,
  min_weight = 1e-10
)
```

Arguments

`sampler`	A sampler object returned by `surface_sampler()`.
`sigma`	Bandwidth for Gaussian kernel weights. If NULL, uses the default from the sampler's dthresh (dthresh/2).
`normalize`	Logical; if TRUE (default), weights for each vertex sum to 1.
`min_weight`	Minimum weight threshold; entries below this are dropped. Default is 1e-10 to remove numerical noise.

Details

The output triplets define a sparse matrix P where P[i,j] is the weight for vertex i from voxel j. The actual volume voxel index is `voxel_indices[j]`. To construct a dgCMatrix in R:

```
Matrix::sparseMatrix(i = triplets$i, j = triplets$j, x = triplets$x, dims = triplets$dims)
```

Value

A list with class `"vol2surf_triplets"` containing:

i Integer vector of vertex indices (1-based row indices)

j Integer vector of voxel indices (1-based column indices into the volume's linear index space, corresponding to sampler$indices)

x Numeric vector of weights

dims Integer vector c(n_vertices, n_voxels) for matrix dimensions

voxel_indices The sampler$indices mapping j values to volume positions

n_vertices Number of surface vertices

n_voxels Number of candidate voxels

nnz Number of non-zero entries

params List of parameters used (sigma, normalize, min_weight, dthresh)

See Also

surface_sampler, apply_surface_sampler

Examples

```
# Requires a surface sampler
# sampler <- surface_sampler(wm, pial, template_vol)
# triplets <- sampler_to_triplets(sampler)
```

---

series,NeuroSurfaceVector,numeric-method

Extract Time Series from Surface Vector

Description

Extracts time series data from specific vertices in a surface vector.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,numeric'
series(x, i)

## S4 method for signature 'NeuroSurfaceVector,integer'
series(x, i)

## S4 method for signature 'NeuroSurfaceVector,ROISurface'
series(x, i)

## S4 method for signature 'NeuroSurface,numeric'
series(x, i)
```

Arguments

x	the object to extract the series from
i	the indices of the series to extract

Value

A matrix where columns are time points and rows are vertex indices

series_roi,NeuroSurfaceVector,numeric-method

Extract ROI Time Series from Surface Vector

Description

Extracts time series data from a region of interest in a surface vector.

Usage

```
## S4 method for signature 'NeuroSurfaceVector,numeric'
series_roi(x, i)

## S4 method for signature 'NeuroSurfaceVector,ROISurface'
series_roi(x, i)
```

Arguments

x	the object to extract the series from
i	the indices of the series to extract

Value

An ROISurfaceVector containing the extracted time series

show,SurfaceGeometryMetaInfo-method

show

Description

Prints a human-readable summary of surface objects to the console.

Usage

```
## S4 method for signature 'SurfaceGeometryMetaInfo'
show(object)

## S4 method for signature 'SurfaceDataMetaInfo'
show(object)

## S4 method for signature 'ROISurface'
show(object)

## S4 method for signature 'SurfaceGeometry'
```

```
show(object)

## S4 method for signature 'NeuroSurfaceVector'
show(object)

## S4 method for signature 'NeuroSurface'
show(object)
```

Arguments

object	The surface object to display

Value

Invisibly returns NULL. Called for its side effect of printing.

---

show_surface_plot *Show a surface plot in one step*

Description

This is a convenience wrapper around surface_plot, add_surface_layer, and plot.neurosurf_plot. It is intended for quick inspection and simple publication-style plots.

Usage

```
show_surface_plot(
  lh,
  rh = NULL,
  data = NULL,
  views = c("lateral", "medial"),
  layout = c("grid", "row", "column"),
  cmap = "viridis",
  irange = NULL,
  color_range = NULL,
  thresh = NULL,
  show_colorbar = TRUE,
  outline = FALSE,
  file = NULL,
  width = 1200,
  height = 900,
  ...
)
```

Arguments

lh, rh	Either SurfaceGeometry objects or file paths that can be read by read_surf_geometry. At least one must be provided.
data	Optional numeric vector or list of vectors containing vertex-wise data to plot. If a single numeric vector is supplied, it is split across hemispheres in left-to-right order based on the vertex counts of the surfaces. If NULL, a plain surface is shown.
views	Character vector of named views to display for each hemisphere. See surface_plot for valid values.
layout	One of "grid", "row", or "column" controlling how views and hemispheres are arranged.
cmap	Character string naming a colour map for the data layer.
irange	Optional numeric vector of length 2 giving the minimum and maximum values for the colour scale. Alias for color_range.
color_range	Optional numeric vector of length 2 giving the minimum and maximum values for the colour scale.
thresh	Optional numeric threshold band. A length-2 value is passed to the colour mapper as c(lower, upper); a scalar is treated as a symmetric band around zero.
show_colorbar	Logical; if TRUE, draw a colour bar for the data layer.
outline	Logical; if TRUE, the supplied data are treated as ROI labels and boundaries are drawn instead of a filled map.
file	Optional PNG output path. If supplied, the plot is drawn to this file instead of the active graphics device.
width, height	Pixel dimensions used when file is supplied.
...	Additional arguments passed through to add_surface_layer (for example alpha, outline_col, outline_lwd).

Value

Invisibly returns the underlying "neurosurf_plot" object. The plot is drawn as a side-effect.

Examples

```
geom <- example_surface_geometry()
if (interactive()) {
  show_surface_plot(geom, data = rnorm(nrow(coords(geom))))
}
```

show_surface_widget *Show an interactive surface widget*

Description

This is a convenience wrapper around surfwidget for quick, interactive inspection of a single surface. It returns an HTML widget that can be used in R Markdown documents or Shiny applications.

Usage

```
show_surface_widget(x, data = NULL, ...)
```

Arguments

x	A SurfaceGeometry, NeuroSurface, or related object supported by surfwidget.
data	Optional numeric vector of data values for each vertex when x is a SurfaceGeometry. Ignored if x already carries data (e.g., a NeuroSurface).
...	Additional arguments passed on to surfwidget.

Value

An htmlwidget object.

Examples

```
fs <- load_fsaverage_std8("inflated")
show_surface_widget(fs$lh)
```

smooth *Generic Function for Smoothing a Surface or Associated Data*

Description

The smooth function is a generic function designed to apply smoothing operations to various surface objects. The specific behavior of the function depends on the class of the object passed as the x argument. It can be used to smooth the geometry of a surface, the data associated with a surface, or other related operations depending on the method implemented for the object's class.

This method applies smoothing to a brain surface geometry object of class SurfaceGeometry using various algorithms. Smoothing is useful for removing noise and creating a more continuous surface.

Usage

```
smooth(x, ...)

## S4 method for signature 'SurfaceGeometry'
smooth(
  x,
  type = c("taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight",
    "surfPreserveLaplace"),
  lambda = 0.7,
  mu = -0.53,
  delta = 0.1,
  iteration = 5
)
```

Arguments

x	A SurfaceGeometry object representing the brain surface to be smoothed.
...	Additional arguments passed to the specific method for smoothing the surface object.
type	A character string specifying the smoothing algorithm to use. Available options are: **"taubin"** Applies Taubin smoothing, which preserves the overall shape of the surface while reducing noise. **"laplace"** Performs Laplacian smoothing, which is a basic smoothing method that averages the position of each vertex with its neighbors. **"HClaplace"** Applies a Laplacian smoothing with hard constraints. It preserves the boundary vertices and is useful for surfaces with important edge features. **"fujiLaplace"** Uses a Laplacian smoothing method that preserves features more aggressively than the basic Laplacian method. **"angWeight"** Performs angle-weighted smoothing, which considers the angles between faces to preserve sharp features. **"surfPreserveLaplace"** Applies surface-preserving Laplacian smoothing, aiming to maintain the original surface's key characteristics while smoothing.
lambda	A numeric value that controls the amount of smoothing. Higher values lead to more aggressive smoothing. This parameter is particularly relevant for Taubin and Laplacian smoothing methods.
mu	A numeric value used in Taubin smoothing to control shrinkage. A value close to zero reduces shrinkage, while a negative value can help in shape preservation.
delta	A numeric value used in certain smoothing algorithms to adjust the influence of smoothing (e.g., in surface-preserving methods).
iteration	An integer specifying the number of smoothing iterations to apply. More iterations result in a smoother surface but can also lead to excessive flattening.

Details

The smooth function provides a common interface for smoothing operations on different types of surface objects. The actual smoothing process varies based on the class of the object provided:

- For SurfaceGeometry objects, the function smooths the surface geometry, modifying the shape of the mesh to reduce noise.
- For NeuroSurface objects, the function smooths the data values associated with each vertex, preserving the surface geometry but producing a smoother dataset.

Users should refer to the specific method documentation for the class of object they are working with to understand the exact behavior and parameters.

Value

The function returns the smoothed SurfaceGeometry object with the updated mesh.

See Also

smooth,SurfaceGeometry-method, smooth,NeuroSurface-method

vcgSmooth for more details on the underlying smoothing algorithms.

Examples

```
sg <- example_surface_geometry()
smoothed_geom <- smooth(sg, type="taubin", lambda=0.7, iteration=10)


# Load a surface file from the extdata directory
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surface <- read_surf_geometry(surf_file)

# Apply Taubin smoothing to the brain surface
smoothed_surface1 <- smooth(surface, type = "taubin", lambda = 0.5, mu = -0.5, iteration = 10)

# Apply surface-preserving Laplacian smoothing
smoothed_surface2 <- smooth(surface, type = "surfPreserveLaplace", iteration = 5)
```

---

smooth,NeuroSurface-method

Smooth Data on a NeuroSurface Object

Description

This method applies smoothing to the data values associated with a NeuroSurface object. Unlike the geometric smoothing applied to SurfaceGeometry, this function smooths the scalar values (e.g., intensity or activation) associated with each vertex on the surface.

Usage

```
## S4 method for signature 'NeuroSurface'
smooth(x, sigma = 5, ...)
```

Arguments

x	A NeuroSurface object containing the brain surface and associated data to be smoothed.
sigma	A numeric value specifying the smoothing radius. This defines the neighborhood around each vertex used to compute the smoothed value. Default is 5.
...	Additional arguments passed to the smoothing function.

Details

The smoothing process involves averaging the data values within a geodesic neighbourhood of each vertex. For every vertex the function uses find_all_neighbors to locate all vertices within the radius specified by sigma. The smoothed value is the mean of the vertex's own value and those of its neighbours. Increasing sigma results in broader smoothing because more neighbours are included in the average.

The smoothing is particularly useful when working with noisy data or when a smoother representation of the underlying signal is desired. It is commonly applied in neuroimaging to enhance visualization or prepare data for further analysis.

Value

A new NeuroSurface object with the smoothed data values. The geometry remains unchanged.

See Also

smooth,SurfaceGeometry-method for smoothing the geometry of a surface.

Examples

```
# Create a tiny example surface and data
surface <- example_surface_geometry()
n_vertices <- nrow(coords(surface))
random_data <- rnorm(n_vertices)

neuro_surf <- NeuroSurface(geometry = surface,
                           indices = seq_len(n_vertices),
                           data = random_data)

# Apply smoothing with different radii
smoothed_small <- smooth(neuro_surf, sigma = 1)
smoothed_large <- smooth(neuro_surf, sigma = 2)

# The original geometry is preserved, but the data is smoothed
head(series(smoothed_large, indices(smoothed_large)))
```

snapshot_surface *Snapshot a surface to a PNG*

Description

Convenience helper for vignettes and reports: renders a surface with `view_surface()` onto an off-screen rgl device and saves a PNG. When `rgl.useNULL()` is `TRUE` (headless builds), a proper snapshot requires the `webshot2` package; otherwise a blank image is likely and an empty path is returned.

Usage

```
snapshot_surface(surfgeom, file = NULL, width = 1200, height = 900, ...)
```

Arguments

`surfgeom`	A `SurfaceGeometry` object.
`file`	Output path for the PNG. Defaults to the current knitr figure path when knitting, otherwise a temporary file.
`width, height`	Device size in pixels (controls render resolution).
`...`	Additional arguments passed to `view_surface`.

Value

The file path (invisibly). Callers can use `knitr::include_graphics()` or read the image via `png::readPNG()`. In headless mode without `webshot2`, an empty character vector is returned.

Examples

```
fs <- load_fsaverage_std8("inflated")
if (interactive()) {
  img <- snapshot_surface(fs$lh, viewpoint = "lateral", specular = "black")
}
```

surf_to_world *Get Surface-to-World Transform*

Description

Retrieves the 4x4 affine transformation matrix that converts surface coordinates to world (scanner RAS) coordinates.

Usage

```
surf_to_world(x)

## S4 method for signature 'SurfaceGeometry'
surf_to_world(x)
```

Arguments

x	An object containing surface geometry (e.g., SurfaceGeometry).

Details

The surface-to-world transform handles coordinate system conventions (e.g., RAS vs LPI), reference space alignment (e.g., MNI305 vs MNI152), and any embedded transforms from file formats (e.g., GIFTI CoordinateSystemTransformMatrix).

To transform surface vertices to world coordinates: `world_coords = surf_to_world(geom) %*% rbind(t(vertices), 1)`

Value

A 4x4 numeric matrix representing the affine transformation.

See Also

surf_to_world<-, SurfaceGeometry-class

Examples

```
surf_file <- system.file("extdata", "std.8_lh.white.asc", package = "neurosurf")
geom <- read_surf_geometry(surf_file)
xform <- surf_to_world(geom)
print(xform)
```

---

surf_to_world<- *Set Surface-to-World Transform*

Description

Sets the 4x4 affine transformation matrix that converts surface coordinates to world (scanner RAS) coordinates.

Usage

```
surf_to_world(x) <- value

## S4 replacement method for signature 'SurfaceGeometry,matrix'
surf_to_world(x) <- value
```

Arguments

x	An object containing surface geometry (e.g., SurfaceGeometry).
value	A 4x4 numeric matrix representing the affine transformation.

Value

The modified object with the updated transform.

See Also

surf_to_world, SurfaceGeometry-class

Examples

```
surf_file <- system.file("extdata", "std.8_lh.white.asc", package = "neurosurf")
geom <- read_surf_geometry(surf_file)
new_xform <- diag(4)
surf_to_world(geom) <- new_xform
```

---

surface_labels *List available surface labels*

Description

List available surface labels

Usage

```
surface_labels(x)
```

Arguments

x	SurfaceSet

Value

Character vector of labels

Examples

```
geom <- example_surface_geometry()
ss <- surface_set(inflated = geom, pial = geom)
surface_labels(ss)
```

surface_montage *Arrange multiple surface views into a single montage figure*

Description

Renders several surface views as one figure: a row (the default) or grid of panels. It is designed for R Markdown / pkgdown documents, where it replaces the repetitive "snapshot each panel, check it, otherwise fall back to an interactive widget" boilerplate with a single call.

When the session can produce static snapshots (an interactive 'rgl' device, or a headless build with **webshot2** installed) the panels are captured and tiled into one PNG. When knitting this is returned via `include_graphics`; otherwise the composed image is written to `file`. If snapshots are unavailable, the panels are drawn into a single `mfrow3d` scene and returned as one `rglwidget` (a single combined widget, never one widget per panel).

Usage

```
surface_montage(
  panels,
  ncol = NULL,
  nrow = NULL,
  width = 600,
  height = 450,
  file = NULL,
  ...
)
```

Arguments

`panels`	A list of panels. Each element is either a surface object (`SurfaceGeometry`, `NeuroSurface` and its subclasses, or a `SurfaceSet`), a file path readable by `read_surf_geometry`, or a list whose first *unnamed* element is such an object and whose remaining *named* elements are per-panel display arguments (for example `thresh`, `vals`, `cmap`, `irange`, or `viewpoint`). Per-panel arguments override the shared arguments passed through `...`.
`ncol, nrow`	Integer layout controls. If both are `NULL` (default), all panels are placed in a single row. If only one is supplied the other is derived from the number of panels.
`width, height`	Pixel dimensions used when snapshotting each panel.
`file`	Optional output path for the composed PNG. When knitting, a figure path is generated automatically and this is ignored. When called interactively, supplying `file` forces composition to that path instead of opening an interactive window.
`...`	Shared display arguments forwarded to every panel (e.g. `cmap`, `irange`, `specular`, `viewpoint`). These are passed to `view_surface` (for `SurfaceGeometry` panels) or to the object's `plot` method (for data surfaces).

Details

Each static panel is rendered in its own fresh `rgl` scene, so lighting is computed independently and panels never share or clobber one another's lights. Panels are centred on equally sized white cells so differing silhouettes stay aligned.

Value

When knitting, a `knitr` image object (static path) or a single `rglwidget` (fallback). When called interactively, the composed file path (if `file` is given) or `invisible(NULL)` after drawing an interactive `mfrow3d` window.

See Also

`view_surface`, `snapshot_surface`, `surface_plot`

Examples

```
geom <- example_surface_geometry()
set.seed(1)
vals <- rnorm(nrow(coords(geom)))
ns <- smooth(NeuroSurface(geom, indices = seq_along(vals), data = vals))

out <- tempfile(fileext = ".png")
if (interactive()) {
  surface_montage(
    list(
      ns,
      list(ns, thresh = c(-0.5, 0.5)),
      list(ns, thresh = c(-1, 1))
    ),
    cmap = grDevices::rainbow(100), irange = c(-2, 2),
    ncol = 3, file = out
  )
}
```

---

surface_plot *Create a surface plot specification*

Description

Create a surface plot specification

Usage

```
surface_plot(
  lh,
  rh = NULL,
  views = c("lateral", "medial"),
  layout = c("grid", "row", "column"),
  mirror_views = FALSE,
  flip = FALSE,
  zoom = 2,
  background = "white",
  brightness = 0.5
)
```

Arguments

lh, rh	Either SurfaceGeometry objects or file paths that can be read by read_surf_geometry. At least one must be provided.
views	Character vector of named views to display for each hemisphere. Valid values include "lateral", "medial", "ventral", "dorsal", "anterior", and "posterior". Defaults to c("lateral","medial").
layout	One of "grid", "row", or "column" controlling how views and hemispheres are arranged.
mirror_views	Logical; if TRUE, reverse the order of the right hemisphere views for "row" and "column" layouts so that they mirror the left hemisphere.
flip	Logical; if TRUE and both hemispheres are present, flip the left/right ordering in the layout (useful for anterior views).
zoom	Numeric zoom factor passed through to the underlying view_surface calls.
background	Background colour for the rgl scene.
brightness	Baseline brightness for a plain surface when no layers are added. Value in $[0, 1]$.

Value

An object of class "neurosurf_plot" that can be further modified with add_surface_layer() and rendered with render_surface_plot() or draw_surface_plot().

Examples

```
geom <- example_surface_geometry()
p <- surface_plot(geom)
p <- add_surface_layer(p, data = rnorm(nrow(coords(geom))))
```

surface_sampler *Build a reusable surface sampler for multi-frame volumes*

Description

Precompute voxel neighbors and distances for each surface vertex so that repeated volume-to-surface projections (e.g., 4D time series) can be done quickly without rebuilding nearest-neighbor searches.

Usage

```
surface_sampler(
  surf_wm,
  surf_pial,
  vol_template,
  mask = NULL,
  sampling = c("midpoint", "normal_line", "thickness"),
  n_samples = NULL,
  depth = NULL,
  radius = 3,
  knn = 6,
  dthresh = 16
)
```

Arguments

surf_wm	White-matter (inner) surface, SurfaceGeometry.
surf_pial	Pial (outer) surface, SurfaceGeometry.
vol_template	A NeuroVol used to define voxel space and candidate voxels (via mask or non-zero entries).
mask	Optional mask limiting candidate voxels; if NULL, all non-zero voxels in vol_template are used.
sampling	How to place sample points relative to the white/pial pair. Options are: • "midpoint" (default): original behaviour, samples at the midpoint between white and pial. • "thickness": samples along the white→pial line at fractions given by depth or evenly spaced. • "normal_line": samples along the vertex normal centred on the midpoint, spanning radius in both directions (or using depth offsets).
n_samples	Number of samples per vertex for sampling != "midpoint" when depth is not supplied.
depth	Optional numeric vector controlling sampling positions; interpreted as fractions of thickness (for "thickness") or multiples of radius (for "normal_line").
radius	Radius (in voxel units) for normal-line sampling when sampling = "normal_line"; also used as the distance scale when interpreting depth offsets for that mode.

knn	The number of nearest neighbors to consider for mapping (default: 6).
dthresh	The maximum distance threshold for valid mapping. A voxel is only considered if it is less than dthresh units away from the vertex (default: 2 * sigma).

Value

A list with class "surface_sampler" containing precomputed neighbor indices and distances for each vertex.

Examples

```
# Requires white and pial surfaces plus a template volume
# wm <- read_surf_geometry("lh.white")
# pial <- read_surf_geometry("lh.pial")
# template_vol <- neuroim2::read_vol("template.nii")
# sampler <- surface_sampler(wm, pial, template_vol)
```

---

surface_set *Construct a SurfaceSet*

Description

Construct a SurfaceSet

Usage

```
surface_set(..., hemi = NULL, default_label = NULL)
```

Arguments

...	Named ‘SurfaceGeometry‘ objects, or a single named list of them.
hemi	Hemisphere code; defaults to the hemi of the first geometry.
default_label	Optional default label; defaults to the first provided label.

Value

A ‘SurfaceSet‘ instance.

Examples

```
# Create a simple SurfaceSet with a single geometry
geom <- example_surface_geometry()
ss <- surface_set(inflated = geom)
surface_labels(ss)
```

SurfaceDataMetaInfo-class *SurfaceDataMetaInfo*

Description

This class contains meta information for surface-based data (the values that map to a surface geometry)

Value

An object of class `SurfaceDataMetaInfo`.

Slots

`header_file` name of the file containing meta information

`data_file` name of the file containing data

`file_descriptor` descriptor of image file format

`node_count` the number of nodes for which surface data exists

`nels` the number of data vectors (typically the number of columns in the surface data matrix; nels = 1 for a single surface data set)

`label` a label indicating the type of surface (e.g. white, pial, inflated, flat, spherical)

Examples

```
meta_info <- new("SurfaceDataMetaInfo",
                 header_file = "data_header.txt",
                 data_file = "surface_data.1D",
                 file_descriptor = new("FileFormat"),
                 node_count = length(nodes(geometry)),
                 nels = 1L,
                 label = "thickness")
```

SurfaceDisk *Create a Region on Surface*

Description

Creates a Region on a Surface from a radius and surface

Usage

```
SurfaceDisk(surf, index, radius, max_order = NULL)
```

Arguments

surf	a SurfaceGeometry or BrainSurface or BrainSurfaceVector
index	the index of the central surface node. Must be a numeric integer value within 1:length(V(surf@graph)).
radius	the size in mm of the geodesic radius. Must be a single positive numeric value.
max_order	maximum number of edges to traverse. default is computed based on average edge length.

Details

The igraph associated with surf must have an edge attribute named dist containing numeric weights with no NA values.

Value

An ROISurfaceVector if surf is a BrainSurface or BrainSurfaceVector, otherwise an ROISurface containing the vertices within the specified geodesic radius.

SurfaceGeometry *Create a SurfaceGeometry Object*

Description

This function creates a new SurfaceGeometry object from vertex coordinates and face indices.

Usage

```
SurfaceGeometry(
  vert,
  faces,
  hemi,
  label = NA_character_,
  surf_to_world = diag(4)
)
```

Arguments

vert	A numeric matrix with 3 columns representing the x, y, and z coordinates of vertices.
faces	An integer matrix where each row represents a face, containing indices of vertices that form the face.
hemi	A character string indicating the hemisphere ("lh" for left, "rh" for right, or other identifier).
label	Optional character string describing the surface type (e.g., "pial", "white", "inflated", "sphere").

surf_to_world Optional 4x4 affine transformation matrix from surface coordinates to world (scanner RAS) coordinates. Defaults to identity matrix. This transform handles coordinate system conventions (e.g., RAS vs LPI) and reference space alignment (e.g., MNI305 vs MNI152).

Details

This function constructs a SurfaceGeometry object by creating a mesh and a graph representation of the surface. It uses the rgl package for 3D visualization and the igraph package for graph operations.

The vertex indices in the faces matrix should be 0-based (starting from 0), as they get incremented by 1 when passed to the rgl mesh function.

Value

A new object of class "SurfaceGeometry" containing:

mesh	An rgl mesh object representing the surface.
graph	An igraph object representing the mesh connectivity.
hemi	The hemisphere identifier.
surf_to_world	The 4x4 affine transformation matrix.

Examples

```
# Create a simple icosahedron-like mesh with 12 vertices
set.seed(123)
vertices <- matrix(rnorm(36), ncol=3)

# Create faces with 0-based indices (0 to 11)
# Each face connects three vertices
faces <- matrix(sample(0:11, 60, replace=TRUE), ncol=3)

# Create the SurfaceGeometry object
surf_geom <- SurfaceGeometry(vertices, faces, "lh")

# Visualize the mesh if rgl is available
if(interactive() && requireNamespace("rgl", quietly = TRUE)) {
  rgl::open3d()
  rgl::shade3d(surf_geom@mesh, col="lightblue")
}
```

SurfaceGeometry-class *SurfaceGeometry Class*

Description

The 'SurfaceGeometry' class represents a three-dimensional surface consisting of a set of triangle vertices. It encapsulates the mesh structure, graph representation, and hemisphere information of a brain surface.

Details

This class is fundamental for representing brain surface geometries in neuroimaging analyses. The mesh slot contains the vertex and face information, while the graph slot provides a network representation of the surface topology. The hemi slot specifies which hemisphere the surface represents.

The `surf_to_world` transform enables proper projection between surface and volume spaces: `world_pos = surf_to_world %*% c(vertex_pos, 1)`, then `voxel_ijk = world_to_vox %*% world_pos`.

Value

An object of class `SurfaceGeometry`.

Slots

`mesh` An object of class `mesh3d` representing the underlying 3D mesh structure.

`graph` An object of class `igraph` representing the underlying graph structure of the surface.

`hemi` A character string indicating the hemisphere of the surface ("left", "right", or "both").

`label` A character string describing the surface type (e.g., "pial", "white", "inflated", "sphere").

`surf_to_world` A 4x4 numeric matrix representing the affine transformation from surface coordinates to world (scanner RAS) coordinates. This handles coordinate system conventions (e.g., RAS vs LPI), reference space alignment (e.g., MNI305 vs MNI152), and any embedded transforms from file formats (e.g., GIFTI CoordinateSystemTransformMatrix). Defaults to the identity matrix (surface coordinates are assumed to be in world space).

See Also

`mesh3d`, `igraph`, `surf_to_world`

Examples

```
geometry <- example_surface_geometry()
```

SurfaceGeometryMetaInfo-class
SurfaceGeometryMetaInfo Class

Description

The 'SurfaceGeometryMetaInfo' class encapsulates meta information for brain surface geometry. It stores details about the file locations, surface properties, and spatial characteristics.

Details

This class is crucial for maintaining metadata about brain surface geometries. It provides a structured way to store information about file locations, surface properties, and spatial characteristics, which is essential for proper handling and processing of brain surface data in neuroimaging analyses.

Value

An object of class `SurfaceGeometryMetaInfo`.

Slots

`header_file` A character string specifying the name of the file containing meta information.

`data_file` A character string specifying the name of the file containing the actual surface data.

`file_descriptor` An object of class 'FileFormat' describing the image file format.

`vertices` An integer indicating the number of surface vertices.

`faces` An integer indicating the number of faces in the surface mesh.

`embed_dimension` An integer specifying the dimensionality of the embedding (typically 3 for 3D surfaces).

`label` A character string indicating the type of surface (e.g., "white", "pial", "inflated", "flat", "spherical").

`hemi` A character string indicating the hemisphere ("lh" for left, "rh" for right, or "unknown").

Examples

```
meta_info <- new("SurfaceGeometryMetaInfo",
                 header_file = "surface_meta.txt",
                 data_file = "surface_data.gii",
                 file_descriptor = new("FileFormat"),
                 vertices = 40000L,
                 faces = 79998L,
                 embed_dimension = 3L,
                 label = "white",
                 hemi = "lh")
```

SurfaceGeometrySource-class
SurfaceGeometrySource Class

Description

The 'SurfaceGeometrySource' class serves as a factory for creating SurfaceGeometry instances. It encapsulates the meta information required to construct a surface geometry.

Details

This class is designed to facilitate the creation of SurfaceGeometry objects by providing a standardized way to store and access the required metadata. It acts as an intermediate step in the process of loading and constructing surface geometries from various file formats and sources.

Value

An object of class SurfaceGeometrySource.

Slots

meta_info An object of class SurfaceGeometryMetaInfo containing the metadata necessary for creating a surface geometry.

See Also

SurfaceGeometry, SurfaceGeometryMetaInfo

Examples

```
# Create a SurfaceGeometryMetaInfo object
meta_info <- new("SurfaceGeometryMetaInfo",
                 header_file = "surface_meta.txt",
                 data_file = "surface_data.gii",
                 file_descriptor = new("FileFormat"),
                 vertices = 40000L,
                 faces = 79998L,
                 embed_dimension = 3L,
                 label = "white",
                 hemi = "lh")

# Create a SurfaceGeometrySource object
geom_source <- new("SurfaceGeometrySource", meta_info = meta_info)

# Use geom_source to create a SurfaceGeometry object (hypothetical function)
# surface_geometry <- createSurfaceGeometry(geom_source)
```

SurfaceSearchlight *SurfaceSearchlight*

Description

Creates an iterator that systematically samples searchlight regions on a surface mesh using geodesic distance to define regions.

Usage

```
SurfaceSearchlight(
  surfgeom,
  radius = 8,
  nodeset = NULL,
  distance_type = c("euclidean", "geodesic", "spherical"),
  as_deflist = FALSE
)
```

Arguments

surfgeom	A SurfaceGeometry object representing the surface mesh.
radius	Numeric, radius of the searchlight as a geodesic distance in mm. Must be a positive, length-one value.
nodeset	Integer vector, optional subset of surface node indices to use. If provided, the vector must contain at least one node index.
distance_type	Character, the distance metric to use: "euclidean", "geodesic", or "spherical".
as_deflist	Logical, whether to return a deflist object.

Details

Create a Searchlight iterator for surface mesh using geodesic distance to define regions.

This function creates a systematic searchlight iterator, which visits each node of the surface mesh in order. The searchlight region for each node is defined by the specified radius and distance metric.

Value

An iterator object of class "Searchlight".

Examples

```
file <- system.file("extdata", "std.8_lh.smoothwm.asc", package = "neurosurf")
geom <- read_surf(file)
searchlight <- SurfaceSearchlight(geom, 12, distance_type = "geodesic")
nodes <- searchlight$nextElem()
```

`SurfaceSet-class` *SurfaceSet: bundle multiple surface variants for one hemisphere*

Description

A 'SurfaceSet' keeps several 'SurfaceGeometry' variants (e.g., pial, white, inflated, sphere) for the same hemisphere together with a default choice.

Usage

```
## S4 method for signature 'SurfaceSet'
geometry(x)

## S4 method for signature 'SurfaceSet'
vertices(x, ...)

## S4 method for signature 'SurfaceSet'
faces(x, ...)

## S4 method for signature 'SurfaceSet'
nodes(x)

## S4 method for signature 'SurfaceSet'
graph(x, ...)

## S4 method for signature 'SurfaceSet'
curvature(x, ...)
```

Arguments

`x`	a 'SurfaceSet'
`...`	additional arguments forwarded to the underlying geometry method

Value

An S4 object of class `SurfaceSet`. Use `surface_set` to construct instances.

Slots

`hemi` Character string for hemisphere (e.g., "lh" or "rh").

`surfaces` Named list of 'SurfaceGeometry' objects keyed by their label.

`default_label` Character string giving the label to use by default.

surfwidget *Create a Surface Widget*

Description

This generic function creates a widget for visualizing surface data, allowing for different implementations based on the type of surface object.

Create a surfwidget to display brain surface data.

Usage

```
surfwidget(x, width = NULL, height = NULL, ...)

## S4 method for signature 'SurfaceGeometry'
surfwidget(
  x,
  width = NULL,
  height = NULL,
  data = NULL,
  cmap = jet_colors(256),
  irange = NULL,
  thresh = c(0, 0),
  vertexColors = NULL,
  alpha = 1,
  curvature = NULL,
  colorbar = TRUE,
  colorbar_label = NULL,
  layers = NULL,
  config = list(),
  ...
)

## S4 method for signature 'NeuroSurface'
surfwidget(
  x,
  width = NULL,
  height = NULL,
  cmap = jet_colors(256),
  irange = range(x@data),
  thresh = c(0, 0),
  vertexColors = NULL,
  alpha = 1,
  curvature = NULL,
  colorbar = TRUE,
  colorbar_label = NULL,
  layers = NULL,
```

```
  config = list(),
  ...
)

## S4 method for signature 'ColorMappedNeuroSurface'
surfwidget(
  x,
  width = NULL,
  height = NULL,
  thresh = NULL,
  vertexColors = NULL,
  alpha = 1,
  curvature = NULL,
  colorbar = TRUE,
  colorbar_label = NULL,
  layers = NULL,
  config = list(),
  ...
)

## S4 method for signature 'VertexColoredNeuroSurface'
surfwidget(
  x,
  width = NULL,
  height = NULL,
  alpha = 1,
  curvature = NULL,
  colorbar = TRUE,
  colorbar_label = NULL,
  layers = NULL,
  config = list(),
  ...
)
```

Arguments

x	A SurfaceGeometry, NeuroSurface, ColorMappedNeuroSurface, or VertexColoredNeuroSurface object
width	The width of the widget
height	The height of the widget
...	Additional arguments for customizing the widget appearance and behavior.
data	Optional. Numeric vector of data values for each vertex.
cmap	Optional. Color map for data visualization.
irange	Optional. Intensity range for data visualization.
thresh	Optional. Threshold range for data visualization.
vertexColors	Optional. Vector of colors for each vertex.

`alpha` Opacity of the surface (0 to 1).

`curvature` Optional numeric vector of curvature values for each vertex. If not supplied for a `SurfaceGeometry` object, it is computed via `curvature(x)`.

`colorbar` Logical; if `TRUE` (default), render a colorbar when a colormap is used.

`colorbar_label` Optional character label shown alongside the colorbar.

`layers` Optional list of additional data layers to display on the surface. Each layer should be a list with elements such as `data`, `cmap`, `alpha`, and optionally outline-specific parameters.

`config` A list of configuration options for the surface rendering:

`shininess` Numeric between 0 and 100. Controls the shininess of the material. Higher values create a more polished appearance. Default is 30.

`specularColor` Character. Hex color code for the specular highlights. Default is "#111111".

`flatShading` Logical scalar. If `TRUE`, uses flat shading; if `FALSE`, uses smooth shading. Default is `FALSE`.

`ambientLightColor` Character. Hex color code for the ambient light. Default is "#404040".

`directionalLightColor` Character. Hex color code for the directional light. Default is "#ffffff".

`directionalLightIntensity` Numeric between 0 and 1. Intensity of the directional light. Default is 0.5.

Unknown elements in `config` are ignored with a warning.

Details

The surfwidget function creates an interactive widget for visualizing surface data, such as brain surfaces. The specific implementation depends on the class of the object provided, allowing for customized behavior for different types of surface representations.

Value

An HTMLWidget object representing the surface visualization.

An HTMLWidget object

See Also

`plot_js`, `SurfaceGeometry`, `NeuroSurface`

Examples

```
geom <- example_surface_geometry()
surfwidget(geom)
```

updateColorMap *Update Surface Color Map*

Description

Change the color map used by an existing surfwidget.

Usage

```
updateColorMap(widget, colorMap)
```

Arguments

widget	A surfwidget object as returned by surfwidget.
colorMap	A vector of colors defining the new color map.

Value

The modified surfwidget object (invisibly).

---

values,ROISurface-method
Extract Data Values from Surface Objects

Description

Extracts the data values associated with vertices in a surface object.

Usage

```
## S4 method for signature 'ROISurface'
values(x, ...)

## S4 method for signature 'ROISurfaceVector'
values(x, ...)

## S4 method for signature 'NeuroSurface'
values(x, ...)
```

Arguments

x	the object to extract the values from
...	additional arguments

Value

A numeric vector or matrix of data values

VertexColoredNeuroSurface *VertexColoredNeuroSurface*

Description

This function creates a VertexColoredNeuroSurface object, which represents a surface with explicit colors assigned to each vertex.

Usage

```
VertexColoredNeuroSurface(geometry, indices, colors, data = NULL)
```

Arguments

geometry	A SurfaceGeometry object representing the underlying surface structure.
indices	An integer vector specifying the indices of valid surface nodes.
colors	A character vector of hex color codes for each vertex.
data	An optional numeric vector of data values. If not provided, defaults to zeros. This parameter exists for compatibility with the parent NeuroSurface class but is not used for coloring (colors are specified directly via the colors parameter).

Details

This object represents a surface where each vertex has an explicitly assigned color, bypassing any data-to-color mapping. This is useful when you want direct control over vertex colors, such as when displaying parcellation results or pre-computed color schemes.

Value

A VertexColoredNeuroSurface object containing the surface geometry with explicit vertex colors.

See Also

SurfaceGeometry, NeuroSurface, ColorMappedNeuroSurface

Examples

```
# Load a sample surface geometry
surf_file <- system.file("extdata", "std.8_lh.inflated.asc", package = "neurosurf")
surf_geom <- read_surf_geometry(surf_file)

# Get first 100 vertices for this example
n_verts <- min(100, nrow(coords(surf_geom)))
vertex_indices <- 1:n_verts
```

```
# Create colors based on vertex coordinates (x-position)
x_coords <- coords(surf_geom)[vertex_indices, 1]
vertex_colors <- ifelse(x_coords > median(x_coords), "#FF6B6B", "#4ECDC4")

# Create the VertexColoredNeuroSurface object
colored_surf <- VertexColoredNeuroSurface(geometry = surf_geom,
                                          indices = vertex_indices,
                                          colors = vertex_colors)

# Print the object summary
print(colored_surf)

# The object can now be plotted with the specified colors
# plot(colored_surf) # Requires rgl package
```

`VertexColoredNeuroSurface-class` *VertexColoredNeuroSurface*

Description

A three-dimensional surface consisting of a set of triangle vertices with one color per vertex.

Details

This class extends `NeuroSurface` by adding per-vertex coloring functionality. The `colors` slot contains a vector of hex color codes that define the color for each vertex. Unlike `ColorMappedNeuroSurface`, this class does not use a colormap or data mapping, but instead directly specifies colors for each vertex.

Value

An object of class `VertexColoredNeuroSurface`.

Slots

`geometry` The surface geometry, an instance of `SurfaceGeometry` or `SurfaceSet`

`indices` An `integer` vector specifying the subset of valid surface nodes encoded in the `geometry` object

`colors` A character vector of hex color codes representing the color of each vertex

See Also

`view_surface`

Examples

```
# First create a simple tetrahedron mesh
vertices <- c(
  0, 0, 0,
  1, 0, 0,
  0, 1, 0,
  0, 0, 1
)
triangles <- c(
  1, 2, 3,
  1, 2, 4,
  1, 3, 4,
  2, 3, 4
)

# Create mesh3d object
mesh <- rgl::mesh3d(vertices = vertices, triangles = triangles)

# Create a graph representation
edges <- rbind(
  c(1,2), c(1,3), c(1,4),
  c(2,3), c(2,4),
  c(3,4)
)
graph <- igraph::graph_from_edgelist(edges)

# Create a SurfaceGeometry object
geometry <- new("SurfaceGeometry",
                mesh = mesh,
                graph = graph,
                hemi = "left")

# Define indices for all vertices
indices <- 1:4

# Create placeholder data (required by NeuroSurface parent class)
vertex_data <- c(0, 0, 0, 0)  # Not used for coloring

# Define explicit colors for each vertex
vertex_colors <- c(
  "#FF0000",  # Red for vertex 1
  "#00FF00",  # Green for vertex 2
  "#0000FF",  # Blue for vertex 3
  "#FFFF00"   # Yellow for vertex 4
)

# Create the VertexColoredNeuroSurface object
colored_vertices <- new("VertexColoredNeuroSurface",
                        geometry = geometry,
                        indices = indices,
                        data = vertex_data,
                        colors = vertex_colors)
```

```
# In this example, each vertex has an explicit color:
# - Vertex 1 is red
# - Vertex 2 is green
# - Vertex 3 is blue
# - Vertex 4 is yellow
```

`VertexData-class` *VertexData*

Description

A set of arbitrary vertices associated with a data table

Details

The VertexData class provides a flexible way to associate arbitrary data with specific vertices in a brain surface. Unlike ROISurface and similar classes, VertexData does not require or maintain a reference to surface geometry, making it lighter and more versatile for storing and manipulating vertex-specific information.

This class is particularly useful for:

- Storing heterogeneous data (different data types) associated with vertices
- Maintaining analysis results like statistical outcomes for specific vertices
- Tracking vertex-specific annotations or classifications
- Creating lookup tables for vertex properties that can be joined with other data

The `data` slot contains a data frame where each row corresponds to a vertex in the same order as the `indices` slot. This structure allows storing multiple attributes of different types (numeric, character, logical) for each vertex, and facilitates standard data frame operations like filtering, sorting, and joining.

Value

An object of class `VertexData`.

Slots

`indices` the node indices

`data` the associated table with `nrow(data) == length(indices)`

Examples

```
# Create a set of vertex indices
vertex_indices <- c(10L, 25L, 50L, 100L, 200L)

# Create a data frame with information for each vertex
# Each row corresponds to one vertex in the same order as indices
vertex_data <- data.frame(
  value = c(0.5, 1.2, 0.8, 1.5, 0.3),
  label = c("A", "B", "A", "C", "B"),
  significant = c(TRUE, FALSE, TRUE, TRUE, FALSE)
)

# Create the VertexData object
vd <- new("VertexData",
         indices = vertex_indices,
         data = vertex_data)

# Access the data
# vd@data$value
# vd@data$label[vd@data$significant]
# vd@indices[vd@data$label == "A"]
```

---

vertices *Extract Vertices from a Surface Object*

Description

Extracts the vertices from a surface object, providing a standardized interface across different surface representations.

Usage

```
vertices(x, ...)

## S4 method for signature 'SurfaceGeometry'
vertices(x, indices)

## S4 method for signature 'NeuroSurface'
vertices(x)

## S4 method for signature 'NeuroSurfaceVector'
vertices(x, indices)
```

Arguments

x	An object representing a surface.

...	Additional arguments passed to methods.
indices	a vector of indices specifying the valid surface nodes.

Value

A matrix or data structure containing vertex information.

See Also

nodes, faces

Examples

```
vertex_data <- vertices(example_surface_geometry())
num_vertices <- nrow(vertex_data)
```

---

view_surface *Display a 3D Brain Surface using RGL*

Description

Renders a 3D brain surface mesh using the 'rgl' package. This function provides flexible options for coloring the surface based on data values or predefined colors, adjusting transparency, controlling lighting, setting viewpoints, and overlaying spherical markers.

Usage

```
view_surface(
  surfgeom,
  vals = NA,
  cmap = grDevices::rainbow(256, alpha = 1),
  vert_clrs = NULL,
  bgcol = "lightgray",
  alpha = 1,
  add_normals = TRUE,
  thresh = NULL,
  irange = NULL,
  specular = "black",
  lit = NULL,
 viewpoint = c("lateral", "medial", "ventral", "dorsal", "anterior", "posterior"),
  new_window = TRUE,
  offset = c(0, 0, 0),
  zoom = 1,
  spheres = NULL,
  spheres_map_surface = NULL,
  spheres_map_label = NULL,
```

```
  spheres_as_vertices = FALSE,
  vectors = NULL,
  vector_vertices = NULL,
  vector_scale = NULL,
  vector_color = "red",
  vector_alpha = 0.8,
  vector_lwd = 1.5,
  vals_vertices = NULL,
  vals_smoothing = c("auto", "nearest"),
  vals_smoothing_steps = 20,
  label = NULL,
  ...
)
```

Arguments

surfgeom	A SurfaceGeometry object representing the 3D brain surface mesh to be displayed, or a SurfaceSet containing multiple variants.
vals	An optional numeric vector containing data values for each vertex on the surface. If provided and ‘vert_clrs‘ is NULL, these values are mapped to colors using ‘cmap‘ and ‘irange‘.
cmap	A vector of colors (e.g., hex codes) defining the color map used when ‘vals‘ is provided and ‘vert_clrs‘ is NULL. Defaults to ‘rainbow(256)‘.
vert_clrs	An optional character vector of hex color codes for each vertex. If provided, these colors directly override any coloring derived from ‘vals‘ and ‘cmap‘. The length should match the number of vertices in ‘surfgeom‘.
bgcol	A single hex color code or a vector of hex color codes used as the base color for the surface. If ‘vals‘ or ‘vert_clrs‘ are provided, this color is blended with the data/vertex colors. Defaults to "lightgray".
alpha	A numeric value between 0 (fully transparent) and 1 (fully opaque) controlling the overall transparency of the surface. Defaults to 1.
add_normals	Logical. If TRUE (default), surface normals are calculated and added to the mesh, which improves the appearance of lighting effects.
thresh	An optional numeric vector of length 2, ‘c(lower, upper)‘. Vertices with ‘vals‘ *outside* this range (i.e., ‘< lower‘ or ‘> upper‘) are made fully transparent. This is applied *after* the general ‘alpha‘. Defaults to NULL (no thresholding).
irange	An optional numeric vector of length 2, ‘c(min, max)‘. Specifies the range of ‘vals‘ to map onto the ‘cmap‘. Values outside this range will be clamped to the min/max colors. Defaults to the full range of ‘vals‘.
specular	The color of specular highlights on the surface, affecting its shininess. Can be a color name (e.g., "white") or hex code. Defaults to "black" for a matte look. Set to a brighter colour for a glossier appearance.
lit	Logical. If TRUE, enables lighting effects on the surface. If FALSE, disables lighting for a flat appearance. If NULL (default), automatically sets to TRUE for interactive sessions and FALSE when knitting (when rgl.useNULL is TRUE).

`viewpoint` A character string specifying a predefined view (e.g., "lateral", "medial", "ventral", "dorsal", "anterior", "posterior"). The actual view depends on the hemisphere (‘surfgeom@hemi‘, e.g., "left_lateral"). Alternatively, a 4x4 transformation matrix defining a custom view. Defaults to "lateral".

`new_window` Logical. If TRUE (default), opens a new ‘rgl‘ window for the plot. If FALSE, attempts to plot in the currently active ‘rgl‘ window (useful for updates or within Shiny apps).

`offset` A numeric vector of length 3 specifying a translation offset ‘c(x, y, z)‘ applied to the surface coordinates before rendering. Defaults to ‘c(0, 0, 0)‘.

`zoom` A numeric value controlling the camera zoom level. Defaults to 1 (no zoom). Values > 1 zoom in, < 1 zoom out.

`spheres` An optional data frame to draw spheres at specific locations on or near the surface. Must contain columns ‘x‘, ‘y‘, ‘z‘ (coordinates), and ‘radius‘. Can optionally include a ‘color‘ column (hex codes or color names) for individual sphere colors (defaults to black). Alternatively, supply a ‘vertex‘ column (1-based vertex ids) and set `spheres_as_vertices = TRUE` to position foci by vertex.

`spheres_map_surface` Optional `SurfaceGeometry`, `SurfaceSet`, or file path used to map sphere coordinates to the nearest vertex on that surface before snapping to `surfgeom`. Assumes both surfaces share the same vertex ordering (e.g., white -> inflated).

`spheres_map_label` Optional surface label to use when `spheres_map_surface` is a `SurfaceSet`.

`spheres_as_vertices` Logical; if `TRUE`, interpret the ‘vertex‘ column of `spheres` as 1-based vertex ids on `surfgeom` rather than raw coordinates.

`vectors` Optional matrix (n x 3) of XYZ vectors to draw as line glyphs.

`vector_vertices` Optional vertex ids matching rows of `vectors` when they are defined on a subset of vertices.

`vector_scale` Optional numeric scale factor for vectors. If `NULL`, a heuristic scale based on mesh extent and vector magnitudes is used.

`vector_color` Colour for the vectors (single value or vector).

`vector_alpha` Opacity for the vectors (0–1).

`vector_lwd` Numeric line width for vector glyphs.

`vals_vertices` Optional integer vector of 1-based vertex ids corresponding to ‘vals‘ when ‘length(vals) < n_vertices‘. Enables sparse data inputs.

`vals_smoothing` One of ‘"auto"‘ (default) or ‘"nearest"‘. When using sparse data, ‘"auto"‘ diffuses values with neighbor averaging after nearest fill; ‘"nearest"‘ performs nearest-neighbour fill only.

`vals_smoothing_steps` Integer number of smoothing iterations applied when ‘vals_smoothing = "auto"‘. Ignored otherwise.

`label` Optional surface label to select when ‘surfgeom‘ is a `SurfaceSet`. Defaults to the set’s ‘default_label‘.

`...` Additional arguments passed directly to ‘rgl::shade3d‘ for fine-grained control over rendering (e.g., ‘lit‘, ‘smooth‘).

Details

Coloring: Surface vertex colors are determined by the following priority: 1. ‘vert_clrs‘: If provided, these specific hex colors are used. 2. ‘vals‘ & ‘cmap‘: If ‘vals‘ is provided and ‘vert_clrs‘ is NULL, ‘vals‘ are mapped to ‘cmap‘ based on ‘irange‘. 3. ‘bgcol‘: If neither ‘vert_clrs‘ nor ‘vals‘ are used for coloring, ‘bgcol‘ is applied uniformly. If ‘bgcol‘ is specified alongside ‘vert_clrs‘ or ‘vals‘, the colors are blended based on the ‘alpha‘ parameter.

Transparency: Overall transparency is set by ‘alpha‘. Additional threshold-based transparency can be applied using ‘thresh‘ when ‘vals‘ are provided. Vertices with values outside the ‘thresh‘ range become fully transparent.

Lighting: ‘add_normals=TRUE‘ is recommended for realistic lighting. The ‘specular‘ parameter controls the shininess.

Viewpoint: Predefined viewpoints (‘"lateral"‘, ‘"medial"‘, etc.) are automatically adjusted based on the hemisphere specified in ‘surfgeom@hemi‘ (e.g., "lh" results in "left_lateral"). If ‘hemi‘ is unknown, the current ‘rgl‘ view is used unless a custom 4x4 matrix is provided.

Performance: Rendering very large surfaces or surfaces with complex coloring/transparency can be computationally intensive.

Value

Invisibly returns the object ID(s) of the shape(s) added to the RGL scene by ‘rgl::shade3d‘. This can be useful for modifying the scene later.

See Also

shade3d, spheres3d, view3d, SurfaceGeometry

Examples

```
surf_geom <- example_surface_geometry()
if (interactive()) {
  view_surface(surf_geom, viewpoint = "lateral")
}
```

---

vol_to_surf *Map values from a 3D volume to a surface in the same coordinate space*

Description

This function maps values from a 3D volume to a surface representation, allowing for different mapping strategies.

Usage

```
vol_to_surf(
  surf_wm,
  surf_pial,
  vol,
  mask = NULL,
  fun = c("avg", "nn", "mode"),
  knn = 6,
  sigma = 8,
  dthresh = sigma * 2,
  fill = 0,
  sampling = c("midpoint", "normal_line", "thickness"),
  n_samples = NULL,
  depth = NULL,
  radius = 3,
  sampler = NULL
)
```

Arguments

surf_wm	The white matter (inner) surface, typically of class SurfaceGeometry.
surf_pial	The pial (outer) surface, typically of class SurfaceGeometry.
vol	An image volume of type NeuroVol that is to be mapped to the surface.
mask	A mask defining the valid voxels in the image volume. If NULL, all non-zero voxels are considered valid.
fun	The mapping function to use. Options are: • "avg": Average of nearby voxels (default) • "nn": Nearest neighbor • "mode": Most frequent value among nearby voxels
knn	The number of nearest neighbors to consider for mapping (default: 6).
sigma	The bandwidth of the smoothing kernel for the "avg" mapping function (default: 8).
dthresh	The maximum distance threshold for valid mapping. A voxel is only considered if it is less than dthresh units away from the vertex (default: 2 * sigma).
fill	Value used when no nearby voxels are found (default: 0 to preserve previous behavior).
sampling	How to place sample points relative to the white/pial pair. Options are: • "midpoint" (default): original behaviour, samples at the midpoint between white and pial. • "thickness": samples along the white→pial line at fractions given by depth or evenly spaced. • "normal_line": samples along the vertex normal centred on the midpoint, spanning radius in both directions (or using depth offsets).

n_samples	Number of samples per vertex for sampling != "midpoint" when depth is not supplied.
depth	Optional numeric vector controlling sampling positions; interpreted as fractions of thickness (for "thickness") or multiples of radius (for "normal_line").
radius	Radius (in voxel units) for normal-line sampling when sampling = "normal_line"; also used as the distance scale when interpreting depth offsets for that mode.
sampler	Optional surface sampler object created by surface_sampler(). When provided, the sampler is reused and other sampling-related arguments are ignored.

Value

A NeuroSurface object containing the mapped values.

Examples

```
# Load standard white and pial surfaces from the package
wm_surf_file <- system.file("extdata", "std.8_lh.white.asc", package = "neurosurf")
pial_surf_file <- system.file("extdata", "std.8_lh.pial.asc", package = "neurosurf")

surf_wm <- read_surf_geometry(wm_surf_file)
surf_pial <- read_surf_geometry(pial_surf_file)

# Create a dummy volume for demonstration purposes
bb <- matrix(c(-80, 80, -120, 80, -60, 90), 3, 2, byrow = TRUE)
spacing <- c(1, 1, 1)
dims <- ceiling(abs(bb[,2] - bb[,1]) / spacing)
origin <- bb[,1]
sp <- neuroim2::NeuroSpace(dims, spacing, origin)
vol <- neuroim2::NeuroVol(rnorm(prod(dims)), sp)

# Map volume to surface using average mapping
mapped_surf <- vol_to_surf(surf_wm, surf_pial, vol, fun = "avg")
print(mapped_surf)
```

---

vol_to_surf_sdf *Map a volume to surface after SDF-based rigid alignment*

Description

This wrapper estimates a rigid transform from the input volume to the template surface space using a signed distance field (SDF), then reuses vol_to_surf() for the actual sampling in volume space.

Usage

```
vol_to_surf_sdf(
  surf_wm,
  surf_pial,
  vol,
  sdf_vol,
  mask = NULL,
  n_hull = 2000L,
  thresh = NULL,
  init_par = rep(0, 6),
  outside_penalty = 20,
  method = "L-BFGS-B",
  ...
)
```

Arguments

surf_wm	White-matter (inner) surface in template space
surf_pial	Pial (outer) surface in template space
vol	NeuroVol in (approximate) template/MNI space
sdf_vol	NeuroVol SDF of the template cortical surface
mask	optional mask for hull extraction and mapping
n_hull	number of hull points to retain (approximate)
thresh	optional intensity threshold for hull computation
init_par	optional initial parameters (tx, ty, tz, rx, ry, rz)
outside_penalty	value used when SDF sampling is outside the grid
method	optimization method passed to roptim (default "L-BFGS-B")
...	passed through to vol_to_surf() (e.g., fun, knn, sigma)

Value

A NeuroSurface with values mapped from vol

Examples

```
# Requires surfaces and volumes in template space
# wm <- read_surf_geometry("lh.white")
# pial <- read_surf_geometry("lh.pial")
# vol <- neuroim2::read_vol("mydata.nii")
# sdf <- neuroim2::read_vol("template_sdf.nii")
# result <- vol_to_surf_sdf(wm, pial, vol, sdf)
```

`write_surf_data` *Write Surface Data to File*

Description

This function writes surface data from a NeuroSurface or NeuroSurfaceVector object to a .1D.dset file.

Usage

```
write_surf_data(surf, outstem, hemi = "")
```

Arguments

`surf`	An object of class `NeuroSurface` or `NeuroSurfaceVector` containing the surface data to be written.
`outstem`	A character string specifying the base name for the output file (without extension).
`hemi`	A character string specifying the hemisphere ("lh" for left, "rh" for right). Default is an empty string.

Details

The function writes the surface data to a .1D.dset file, which is a tabular data format. The output file contains node indices in the first column, followed by data values in subsequent columns. The file name is constructed by combining `outstem`, `hemi` (if provided), and the extension ".1D.dset". For NeuroSurfaceVector objects, all columns of data are written. For NeuroSurface objects, only the single data vector is written. The data is written without row names, column names, or quotes.

Value

This function does not return a value. It writes the data to a .1D.dset file as a side effect.

Examples

```
sg <- example_surface_geometry()
nv <- nrow(vertices(sg))
ns <- NeuroSurface(geometry = sg, indices = seq_len(nv), data = rnorm(nv))
write_surf_data(ns, file.path(tempdir(), "output_data"), "lh")
```

Index