Package: threeBrain (via r-universe)

October 23, 2026

Type Package

Title Your Advanced 3D Brain Visualization

Version 1.3.0.77

Description A fast, interactive cross-platform, and easy to share 'WebGL'-based 3D brain viewer that visualizes 'FreeSurfer' and/or 'AFNI/SUMA' surfaces. The viewer widget can be either standalone or embedded into 'R-shiny' applications. The standalone version only require a web browser with 'WebGL2' support (for example, 'Chrome', 'Firefox', 'Safari'), and can be inserted into any websites. The 'R-shiny' support allows the 3D viewer to be dynamically generated from reactive user inputs. Please check the publication by Wang, Magnotti, Zhang, and Beauchamp (2023, <doi:10.1523/ENEURO.0328-23.2023>) for electrode localization. This viewer has been fully adopted by 'RAVE' <https://rave.wiki>, an interactive toolbox to analyze 'iEEG' data by Magnotti, Wang, and Beauchamp (2020, <doi:10.1016/j.neuroimage.2020.117341>). Please check citation(``threeBrain'') for details.

License MPL-2.0

Encoding UTF-8

Language en-US

URL https://dipterix.org/threeBrain/,

https://github.com/dipterix/threeBrain

BugReports https://github.com/dipterix/threeBrain/issues

Imports utils, grDevices, graphics, dipsaus, xml2, servr, png, knitr, shiny (>= 1.2.0), digest (>= 0.6.22), freesurferformats (>= 0.1.7), jsonlite (>= 1.5), stringr (>= 1.3.1), htmlwidgets (>= 1.3), R6 (>= 2.3.0), gifti (>= 0.7.5), oro.nifti (>= 0.9.1)

Suggests rmarkdown, DT, ravetools, htmltools

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev zlib1g-dev

Repository https://rave-ieeg.r-universe.dev

Date/Publication 2026-10-23 04:09:55 UTC

RemoteUrl https://github.com/dipterix/threeBrain

RemoteRef HEAD

RemoteSha 15249ad055da8975259fe86d8351e8219c8b2707

Contents

AbstractGeom . . . 3
add_streamline . . . 6
BlankGeom . . . 9
brain_proxy . . . 10
brain_setup . . . 11
calculate_rotation . . . 11
check_freesurfer_path . . . 12
conform_volume . . . 12
create_group . . . 13
cross_prod . . . 15
DataCubeGeom . . . 15
DataCubeGeom2 . . . 17
default_template_directory . . . 19
FreeGeom . . . 19
freesurfer_lut . . . 20
generate_cortical_parcellation . . . 20
generate_smooth_envelope . . . 21
generate_subcortical_surface . . . 22
geom_freemesh . . . 23
geom_sphere . . . 25
GeomGroup . . . 26
get_digest_header . . . 28
get_ijk2ras . . . 29
import-fs-suma . . . 29
import_from_freesurfer . . . 30
LineSegmentsGeom . . . 31
list_electrode_prototypes . . . 33
localization_module . . . 33
merge_brain . . . 35
new_electrode_prototype . . . 36
plot_slices . . . 37
read_fs_asc . . . 38
read_fs_labels . . . 39
read_fs_m3z . . . 39
read_fs_mgh_mgz . . . 40
read_gii2 . . . 40
read_mgz . . . 41

read_volume 41
renderBrain 42
reorient_volume 42
save_brain 43
seeg_prototype 43
SphereGeom 45
SpriteGeom 46
StreamlineGeom 48
template_subject 49
TextDecorGeom 51
threeBrain 52
threejs_brain 53
threejsBrainOutput 57
TubeGeom 57
video_content 58
viewer_hub 59
ViewerProxy 61
volume_to_surf 68
voxel_colormap 70
voxel_cube 72

Index **75**

`AbstractGeom` *R6 Class - Abstract Class of Geometries*

Description

Base class inherited by all geometry types in the three-brain viewer. Provides common fields for position, layer visibility, transformation, click response, and animation clips.

Public fields

`name` Unique character name of the geometry object.

`type` Geometry type string used by the JavaScript engine.

`render_order` Rendering priority; higher values render on top.

`time_stamp` Deprecated; use the `keyframes` field instead. Numeric vector of animation time points.

`value` Deprecated; use the `keyframes` field instead. Numeric or character animation values.

`keyframes` Named list of `KeyFrame` objects that store animation clip data for this geometry.

`position` Numeric vector of length 3: object origin in world space.

`trans_mat` Optional 4-by-4 transformation matrix (`NULL` = identity).

`disable_trans_mat` Logical; when `TRUE` the transformation matrix is ignored.

`group` `GeomGroup` that owns this geometry, or `NULL`.

`clickable` Logical; whether the geometry responds to mouse clicks in the viewer.

`layer` Camera layer(s); 0 = main camera only, 1 = all cameras, 13 = invisible.

`use_cache` Logical; whether to read/write data from a file cache.

`custom_info` Optional character string for additional annotation.

`subject_code` Subject identifier string, or `NULL`.

Active bindings

`animation_types` Character vector of animation clip names attached to this geometry.

Methods

Public methods:

- `AbstractGeom$new()`
- `AbstractGeom$set_position()`
- `AbstractGeom$set_value()`
- `AbstractGeom$to_list()`
- `AbstractGeom$get_data()`
- `AbstractGeom$animation_time_range()`
- `AbstractGeom$animation_value_range()`
- `AbstractGeom$animation_value_names()`
- `AbstractGeom$clone()`

`AbstractGeom$new()`: Create a new abstract geometry. Subclasses call this via `super$initialize()`.

Usage:

```
AbstractGeom$new(
  name,
  position = c(0, 0, 0),
  group = NULL,
  layer = 0,
  trans_mat = NULL,
  ...
)
```

Arguments:

`name` Unique character name.

`position` Numeric vector of length 3: object origin. Default `c(0, 0, 0)`.

`group` `GeomGroup` to attach this geometry to, or `NULL`.

`layer` Camera layer(s), 0-13. Default `0`.

`trans_mat` Optional 4-by-4 numeric transformation matrix.

`...` Reserved for subclass use.

`AbstractGeom$set_position()`: Set the world-space position of the geometry.

Usage:

```
AbstractGeom$set_position(...)
```

Arguments:

`...` Numeric values that together form a length-3 vector `c(x, y, z)`.

`AbstractGeom$set_value()`: Attach animation data to this geometry as an animation clip.

Usage:

```
AbstractGeom$set_value(
  value = NULL,
  time_stamp = NULL,
  name = "Value",
  target = ".material.color",
  ...
)
```

Arguments:

`value` Numeric or character vector of animation values.

`time_stamp` Numeric vector of time points matching `value`.

`name` Character clip name. Defaults to `"Value"`.

`target` JavaScript property path to animate.

`...` Additional arguments passed to `KeyFrame`.

`AbstractGeom$to_list()`: Serialize the geometry to a named list for JSON export.

Usage:

```
AbstractGeom$to_list()
```

`AbstractGeom$get_data()`: Retrieve a data value from this geometry or its owning group.

Usage:

```
AbstractGeom$get_data(key = "value", force_reload = FALSE, ifnotfound = NULL)
```

Arguments:

`key` Field name or group data key to retrieve. Default `"value"`.

`force_reload` Logical; reload from the file cache even when an in-memory copy exists. Default `FALSE`.

`ifnotfound` Value returned when `key` is not found. Default `NULL`.

`AbstractGeom$animation_time_range()`: Return the time range of a named animation clip.

Usage:

```
AbstractGeom$animation_time_range(ani_name)
```

Arguments:

`ani_name` Name of the animation clip.

`AbstractGeom$animation_value_range()`: Return the value range of a named continuous animation clip.

Usage:

```
AbstractGeom$animation_value_range(ani_name)
```

Arguments:

`ani_name` Name of the animation clip.

`AbstractGeom$animation_value_names()`: Return the category level names of a named discrete animation clip.

Usage:

```
AbstractGeom$animation_value_names(ani_name)
```

Arguments:

`ani_name` Name of the animation clip.

`AbstractGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
AbstractGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`add_streamline` *Add a streamline (fiber tract) bundle to a brain object*

Description

Registers a streamline file so it renders in the 3D viewer. The tract data is not read in R; only the file path is recorded, and the viewer loads and parses the file itself. Bundles are organized into circuit groups so that several bundles contributing to the same brain circuit can be toggled together.

Usage

```
add_streamline(brain, name, color = NA)
```

Arguments

`brain`	a `'threeBrain'` brain object generated from `threeBrain` or `merge_brain`
`name`	one or more streamline keys, of the form `'circuit/bundle'`. The circuit is the top-level sub-folder under `'fs/streamline'` and the bundle is the file name without extension, prefixed by any further sub-folders it sits in; both are matched case-insensitively, and the spelling on disk is the one kept. A key may also glob the bundle with `'*'` to select a whole circuit at once. See 'Key syntax' below
`color`	line color, recycled over the bundles that `name` expands to; when `NA` (default) each bundle is colored from `'fs/streamline/colormap.csv'`, falling back to a color derived from the bundle name. See 'Colors'

Details

Files are searched with the following extension priority: `tck`, `trk`, `trk.gz`, `tt`, `tt.gz`. `VTK` formats (`vtk`, `vtp`) are not supported by the viewer. Naming such a file directly raises an error; when expanding a wild card they are skipped, so one stray file cannot abort a whole circuit. Streamline coordinates are assumed to be in scanner `RAS` space, matching the convention used when tract files are dropped onto the viewer.

The viewer simplifies the streamlines before drawing them, keeping the drawn line within `'Line Simplify Factor'` millimeters of the original. Tract files are often sampled much finer than the screen can show, and every segment costs time in each of the four views, so this is what keeps a large set of bundles interactive. The default is 0.05, and 0 draws every point; set it from R with `brain$plot(controllers = list("Line Simplify Factor" = 0))`. Streamline lengths and the `'Line Selector'` distances are measured on all of the points either way.

Value

`add_streamline` returns the `brain` object, invisibly. The underlying `brain$add_streamline` returns, invisibly, a named list of the bundles it added, keyed by `'circuit/bundle'`. That same key names the bundle's visibility controller in the viewer, as `'Show: circuit/bundle'`

Key syntax

A key names exactly one circuit, and the circuit is always the **first** component: everything after it is the bundle, so `'motor/left/AF'` is the bundle `'left/AF'` of circuit `'motor'`, not a circuit called `'motor/left'`. Sub-folders are therefore a way to load part of a circuit, not a way to create more of them, and all of `'motor'` shares one folder in the viewer's control panel.

`'motor/AF_left'` one bundle, `'AF_left'`, in circuit `'motor'`

`'motor/*'` every bundle in circuit `'motor'`, including those in its sub-folders

`'motor/'` shorthand for `'motor/*'`

`'motor/left/AF'` one bundle stored as `'fs/streamline/motor/left/AF.tck'`, known as `'left/AF'` within circuit `'motor'`

`'motor/left/*'` only the `'left'` sub-folder of circuit `'motor'`, rather than the whole circuit

`'motor/CST_*'` bundles in `'motor'` whose name starts with `'CST_'`

`'motor'` no circuit prefix, hence `'default/motor'`: the file `'motor'` in the `'default'` circuit

`'default/'` the `'default'` circuit, which covers `'fs/streamline/default'` plus the files sitting directly in `'fs/streamline'` – but not the other circuits' folders

`'*'` matches `'/'` as well, which is why `'motor/*'` reaches into sub-folders. It is only allowed in the last component, though: a wild card in a folder name, such as `'*/AF_left'` or `'motor/*/AF_left'`, raises an error so that a misspelled circuit or sub-folder fails loudly instead of quietly matching another one.

Colors

A bundle takes the first color available from three sources:

1. the `color` argument, when not `NA`;

2. the optional table `'fs/streamline/colormap.csv'`, which uses the same format as the drag-and-drop color table, with a `'Filename'` and a `'Color'` column. A `'Filename'` entry may be `'circuit/bundle'`, `'bundle'`, the bundle's bare file name (for a bundle stored in a sub-folder), or `'circuit/'` (the trailing slash marks a circuit-wide entry, which is how a whole circuit is painted one color); more specific entries win, and all comparisons ignore case and surrounding white spaces;

3. otherwise a color derived from the bundle name itself, stable across sessions.

The derived color ignores case, punctuation and the file extension, and folds left/right markers together, so `'CST_left.tck'`, `'CST-Right'` and `'cst right.trk'` all render in the same color. A name whose last six characters are hexadecimal digits spells out its own color, so a bundle called `'CST_ff8800'` renders as `'#FF8800'`.

Every bundle declared this way is downloaded and parsed by the browser when the viewer starts, so avoid declaring more bundles than needed.

Examples

```
# Requires a FreeSurfer directory containing `streamline/motor/AF_left.trk`
## Not run:

brain <- threeBrain(path = "/path/to/fs", subject_code = "subject")

# circuit is "motor", bundle name is "AF_left"
add_streamline(brain, "motor/AF_left", color = "#ff8800")

# the whole `motor` circuit, alternating two colors
add_streamline(brain, "motor/*", color = c("#ff8800", "#00ccff"))

# only the `left` sub-folder of the same circuit
add_streamline(brain, "motor/left/*")

# several keys at once; colors are recycled over the resulting bundles
add_streamline(brain, c("language/", "motor/CST_*"))

# draw every point of every streamline, instead of simplifying them
brain$plot(controllers = list("Line Simplify Factor" = 0))

# bundle keys double as controller names
brain$plot(controllers = list("Show: motor/left/AF" = FALSE))

brain$streamline_types
brain$plot()


## End(Not run)
```

BlankGeom *A geometry that renders nothing*

Description

No-op geometry used when only shared group data needs to be uploaded to the viewer. The geometry renders nothing and is assigned to an invisible camera layer (31).

Super class

AbstractGeom -> BlankGeom

Public fields

`value` Reserved; always `NULL` for blank geometry.

`type` Geometry type string used by the JavaScript engine.

`clickable` Logical; whether the geometry responds to mouse clicks in the viewer.

Methods

Public methods:

- `BlankGeom$set_value()`
- `BlankGeom$new()`
- `BlankGeom$to_list()`
- `BlankGeom$clone()`

`BlankGeom$set_value():` No-op value setter; blank geometry accepts no data.

Usage:

```
BlankGeom$set_value(...)
```

Arguments:

`...` Ignored.

`BlankGeom$new():` Create a new blank geometry.

Usage:

```
BlankGeom$new(
  group,
  name = paste(sample(c(LETTERS, letters, 0:9), 16), collapse = ""),
  ...
)
```

Arguments:

`group` `GeomGroup` to attach this geometry to.

`name` Unique character name. Defaults to a random 16-character alphanumeric string.

`...` Additional arguments forwarded to `AbstractGeom`.

`BlankGeom$to_list()`: Serialize the blank geometry to a named list for JSON export.

Usage:

```
BlankGeom$to_list()
```

`BlankGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
BlankGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`brain_proxy` *Shiny Proxy for Viewer*

Description

Shiny Proxy for Viewer

Usage

```
brain_proxy(outputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`outputId`	shiny output ID
`session`	shiny session, default is current session (see `domains`)

Value

`R6` class `ViewerProxy`

brain_setup — *Setup Package, Install Environment*

Description

Setup Package, Install Environment

Usage

```
brain_setup(continued = FALSE, show_example = TRUE, ...)
```

Arguments

`continued`	logical, there are two phases of setting up environment. You probably need to restart R session after the first phase and continue setting up.
`show_example`	whether to show example of ‘N27‘ subject at the end.
`...`	ignored

Author(s)

Zhengjia Wang

calculate_rotation — *Calculate rotation matrix from non-zero vectors*

Description

Calculate rotation matrix from non-zero vectors

Usage

```
calculate_rotation(vec_from, vec_to)
```

Arguments

`vec_from`	original vector, length of 3
`vec_to`	vector after rotation, length of 3

Value

A four-by-four transform matrix

check_freesurfer_path *Function to check whether 'FreeSurfer' folder has everything we need*

Description

Function to check whether 'FreeSurfer' folder has everything we need

Usage

```
check_freesurfer_path(
  fs_subject_folder,
  autoinstall_template = FALSE,
  return_path = FALSE,
  check_volume = FALSE,
  check_surface = FALSE
)
```

Arguments

`fs_subject_folder`	character, path to 'fs' project directory or 'RAVE' subject directory
`autoinstall_template`	logical, whether 'N27' brain should be installed if missing
`return_path`	logical, whether to return 'FreeSurfer' path
`check_volume`	logical, whether to check volume data
`check_surface`	logical, whether to check surface data (not implemented yet)

Value

logical whether the directory is valid or, if `return_path` is true, return 'FreeSurfer' path

conform_volume *Conform imaging data in 'FreeSurfer' way*

Description

Reproduces conform algorithm used by `'FreeSurfer'` to conform `'NIfTI'` and `'MGH'` images.

Usage

```
conform_volume(x, save_to, dim = c(256, 256, 256))
```

Arguments

`x`	path to the image file
`save_to`	path where the conformed image will be saved, must ends with `'.mgz'`
`dim`	positive integers of length three, the conformed dimension; by default 'FreeSurfer' conform images to `1mm` volume cube with `256x256x256` dimension

Value

Nothing; the result will be save to `save_to`

`create_group` *Create a geometry group containing multiple geometries*

Description

Create a geometry group containing multiple geometries

Usage

```
create_group(name, position = c(0, 0, 0), layer = 1)
```

Arguments

`name`	string, name of the geometry
`position`	x,y,z location of the group
`layer`	layer of the group. reserved

Details

A geometry group is a container of multiple geometries. The geometries within the same group share the same shift and rotations (see example 1). In ECoG/iEEG world, you might have 'MRI', 'CT', 'FreeSurfer' that have different orientations. For example, if you want to align MRI to FreeSurfer, Instead of calculating the position of each geometries, you can just put all MRI components into a group, and then set transform of this group, making the group aligned to FreeSurfer.

GeomGroup also can be used to store large data. To generate 3D viewer, 'threeBrain' needs to dynamically serialize data into JSON format, which can be read by browsers. However, a FreeSurfer brain might be ~30 MB. This is a very large size and might take ~5 seconds to serialize. To solve this problem, GeomGroup supports cache in its 'set_group_data' method. This method supports caching static serialized data into a JSON file, and allows the files to be loaded as static data objects. By "static", I mean the data is not supposed to be dynamic, and it should be "read-only". In JavaScript code, I also optimized such that you don't need to load these large datasets repeatedly. And this allows you to load multiple subjects' brain in a short time.

Value

a GeomGroup instance

Author(s)

Zhengjia Wang

Examples

```
# Example 1: relative position

# create group
g = create_group('Group A')

# create two spheres at 10,0,0, but s2 is relative to group A
s1 = geom_sphere('Sphere 1', radius = 2, position = c(10,0,0))
s2 = geom_sphere('Sphere 2', radius = 2, position = c(10,0,0), group = g)

# set transform (rotation)
g$set_transform(matrix(c(
  0,1,0,0,
  1,0,0,0,
  0,0,1,0,
  0,0,0,1
), byrow = TRUE, ncol = 4))

# global position for s2 is 0,10,0
if( interactive() ) { threejs_brain(s1, s2) }

# Example 2: cache

## Not run:

# download N27 brain
# Make sure you have N27 brain downloaded to `default_template_directory()`
# download_N27()

template_dir <- default_template_directory()

dat = freesurferformats::read.fs.surface(
  file.path(template_dir, 'N27/surf/lh.pial')
)
vertex = dat$vertices[,1:3]
face = dat$faces[,1:3]

# 1. dynamically serialize
mesh = geom_freemesh('lh', vertex = vertex, face = face, layer = 1)

# 2. cache
# Create group, all geometries in this group are relatively positioned
tmp_file = tempfile()
mesh = geom_freemesh('Left Hemisphere cached', vertex = vertex,
                     face = face, cache_file = tmp_file)


## End(Not run)
```

cross_prod — *Calculate cross-product of two vectors in '3D'*

Description

Calculate cross-product of two vectors in '3D'

Usage

```
cross_prod(x, y)
```

Arguments

x, y	3-dimensional vectors

Value

A '3D' vector that is the cross-product of x and y

DataCubeGeom — *R6 Class - Generate Data Cube Geometry*

Description

Volumetric data cube geometry for rendering 3D scalar volumes as voxel data. Stores a flat value array and its 3D dimensions in the owning GeomGroup for JSON export to the three-brain viewer.

Super class

AbstractGeom -> DataCubeGeom

Public fields

type Geometry type string ("datacube").

clickable Logical; always FALSE for volume geometry.

Methods

Public methods:

- DataCubeGeom$new()
- DataCubeGeom$set_value()
- DataCubeGeom$to_list()
- DataCubeGeom$get_data()
- DataCubeGeom$clone()

`DataCubeGeom$new()`: Create a new data cube geometry.

Usage:

```
DataCubeGeom$new(
  name,
  value,
  dim = dim(value),
  group = GeomGroup$new(name = "default"),
  position = c(0, 0, 0),
  cache_file = NULL,
  layer = 13,
  digest = TRUE,
  ...
)
```

Arguments:

`name` Unique character name.

`value` Numeric vector or array of voxel values.

`dim` Integer vector of length 3: dimensions of the volume (`c(nx, ny, nz)`).

`group` `GeomGroup` used to store the voxel data.

`position` Numeric vector of length 3: geometry origin. Default `c(0, 0, 0)`.

`cache_file` Path to a JSON cache file, `TRUE` for a temporary file, or `NULL` to keep data in memory.

`layer` Camera layer. Default `13` (invisible).

`digest` Logical; compute a content digest for cache validation.

`...` Additional arguments forwarded to `AbstractGeom`.

`DataCubeGeom$set_value()`: Update the voxel values of the geometry (not yet implemented; currently calls `.NotYetImplemented()`).

Usage:

```
DataCubeGeom$set_value(value = NULL, dim = dim(value))
```

Arguments:

`value` Numeric vector of replacement voxel values.

`dim` Integer vector of length 3: dimensions matching `value`.

`DataCubeGeom$to_list()`: Serialize the data cube geometry to a named list for JSON export, adding the `isDataCube` flag.

Usage:

```
DataCubeGeom$to_list()
```

`DataCubeGeom$get_data()`: Retrieve a data value from this geometry or its owning group.

Usage:

```
DataCubeGeom$get_data(key, force_reload = FALSE, ifnotfound = NULL)
```

Arguments:

`key` Group data key to retrieve.

force_reload Logical; reload from the file cache even when an in-memory copy exists. Default FALSE.

ifnotfound Value returned when key is not found. Default NULL.

DataCubeGeom$clone(): The objects of this class are cloneable with this method.

Usage:

```
DataCubeGeom$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Zhengjia Wang

DataCubeGeom2 *R6 Class - Generate Data Cube Geometry via 3D Volume Texture*

Description

Volumetric data cube geometry rendered via a 3D texture lookup. Extends DataCubeGeom with four-channel color map support and opacity thresholding.

Super classes

AbstractGeom -> DataCubeGeom -> DataCubeGeom2

Public fields

type Geometry type string ("datacube2").

clickable Logical; always FALSE for volume geometry.

threshold Opacity threshold: voxel values below this level are rendered as transparent. Default 0.6.

color_format three.js texture format string; either "RGBAFormat" or "RedFormat".

color_map Named list describing the four-channel color map applied to the volume, or NULL.

trans_space_from Coordinate space of the input data before applying the transformation matrix; either "model" (default) or "scannerRAS".

Active bindings

is_datacube2 Logical flag; always TRUE for DataCubeGeom2 instances.

Methods

Public methods:

- DataCubeGeom2$new()
- DataCubeGeom2$to_list()
- DataCubeGeom2$clone()

DataCubeGeom2$new(): Create a new data cube geometry using a 3D texture.

Usage:

```
DataCubeGeom2$new(
  name,
  value,
  dim = dim(value),
  half_size = c(128, 128, 128),
  group = GeomGroup$new(name = "default"),
  position = c(0, 0, 0),
  color_format = c("RGBAFormat", "RedFormat"),
  cache_file = NULL,
  layer = 8,
  digest = TRUE,
  ...
)
```

Arguments:

`name` Unique character name.

`value` Integer vector of voxel values (0-255).

`dim` Integer vector of length 3: dimensions of the volume.

`half_size` Numeric vector of length 3: half-extents of the bounding box in world-space units. Default `c(128, 128, 128)`.

`group` `GeomGroup` used to store the voxel data.

`position` Numeric vector of length 3: geometry origin.

`color_format` `three.js` texture format: `"RGBAFormat"` (default) or `"RedFormat"`.

`cache_file` Path to a JSON cache file, `TRUE` for a temporary file, or `NULL` to keep data in memory.

`layer` Camera layer. Default `8` (main camera only).

`digest` Logical; compute a content digest for cache validation.

`...` Additional arguments forwarded to `DataCubeGeom`.

DataCubeGeom2$to_list(): Serialize the texture data cube geometry to a named list for JSON export, adding `threshold`, `color_format`, `color_map`, `isDataCube2`, and `trans_space_from`.

Usage:

```
DataCubeGeom2$to_list()
```

DataCubeGeom2$clone(): The objects of this class are cloneable with this method.

Usage:

```
DataCubeGeom2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

---

default_template_directory

Default Directory to Store Template Brain

Description

Default Directory to Store Template Brain

Usage

```
default_template_directory(check = FALSE)
```

Arguments

check	logical, check if the folder is missing, is so, create one. This option ensures the folder is always created.

Details

When threeBrain.template_dir is not set or invalid, the function checks 'RAVE' (R Analysis and Visualization for 'iEEG', https://rave.wiki) folder at home directory. If this folder is missing, then returns results from R_user_dir('threeBrain', 'data'). To override the default behavior, use options(threeBrain.template_dir=...).

Value

A directory path where template brain is stored at; see also download_N27

Examples

```
default_template_directory()
```

FreeGeom

R6 Class - Generate Geometry from Vertices and Face Indices

Description

Geometry defined by explicit vertex coordinates and face indices, used primarily for brain surface meshes. Supports per-vertex color animation and FreeSurfer annotation overlays.

Author(s)

Zhengjia Wang

freesurfer_lut *Query the 'FreeSurfer' labels*

Description

Query the 'FreeSurfer' labels

Usage

```
freesurfer_lut
```

Details

The 'FreeSurfer' atlases use https://surfer.nmr.mgh.harvard.edu/fswiki/FsTutorial/AnatomicalROI/FreeSurferColorLUT look-up table to query indexes. The 'threeBrain' electrode localization also uses this table to export the `'FSLabel'` from electrode. If volume type is set to `'aparc_aseg'`, then please also use this table to filter.

Examples

```
freesurfer_lut$from_key(0:10)

freesurfer_lut$get_key("ctx-lh-supramarginal")
```

generate_cortical_parcellation *Generate cortical annotations from template using surface mapping*

Description

This is a low-level function. Use `brain$add_annotation` instead.

Usage

```
generate_cortical_parcellation(
  brain,
  template_subject = "fsaverage",
  annotation = "Yeo2011_7Networks_N1000",
  add_annotation = TRUE
)
```

Arguments

brain	Brain object
template_subject	template subject where the annotation is stored
annotation	annotation name in the label folder; default is 'Yeo2011_7Networks_N1000', standing for 'lh.Yeo2011_7Networks_N1000.annot' and 'rh.Yeo2011_7Networks_N1000.annot'.
add_annotation	whether to add annotation to brain

Value

brain with the annotation added if add_annotation is true

---

generate_smooth_envelope *Generate smooth envelope around surface*

Description

Alternative to 'Matlab' version of 'pial-outer-smoothed', use this function along with fill_surface.

Usage

```
generate_smooth_envelope(
  surface_path,
  save_as = NULL,
  inflate = 3,
  verbose = TRUE,
 save_format = c("auto", "bin", "asc", "vtk", "ply", "off", "obj", "gii", "mz3", "byu")
)
```

Arguments

surface_path	path to '*h.pial' surface in the 'FreeSurfer' folder, or a 3-dimensional mesh, see read.fs.surface
save_as	save final envelope to path, or NULL for dry-run
inflate	number of 'voxels' to inflate before fitting envelope; must be a non-negative integer
verbose	whether to verbose the progress; default is true
save_format	format of saved file when save_as is not NULL; see format argument in function write.fs.surface

Value

A 3-dimensional mesh that contains vertices and face indices, the result is also saved to save_as is specified.

Examples

```
if(interactive() &&
   file.exists(file.path(default_template_directory(), "N27"))) {

library(threeBrain)

fs_path <- file.path(default_template_directory(), "N27")

# lh.pial-outer-smoothed
lh_pial <- file.path(fs_path, "surf", "lh.pial")
save_as <- file.path(fs_path, "surf", "lh.pial-outer-smoothed")
generate_smooth_envelope(lh_pial, save_as)

# rh.pial-outer-smoothed
rh_pial <- file.path(fs_path, "surf", "rh.pial")
save_as <- file.path(fs_path, "surf", "rh.pial-outer-smoothed")
generate_smooth_envelope(rh_pial, save_as)

brain <- threeBrain(
  path = fs_path, subject_code = "N27",
  surface_types = 'pial-outer-smoothed'
)
brain$plot(controllers = list(
  "Surface Type" = 'pial-outer-smoothed'
))

}
```

---

generate_subcortical_surface

Approximate 'sub-cortical' surfaces from 'parcellation'

Description

Superseded by `volume_to_surf`. Please do not use this function.

Usage

```
generate_subcortical_surface(
  atlas,
  index,
  save_prefix = NULL,
  label = NULL,
  IJK2RAS = NULL,
  grow = 1,
  remesh = TRUE,
  smooth = TRUE,
  smooth_delta = 3,
```

```
  ...
)
```

Arguments

atlas	path to imaging 'parcellation', can be 'nii' or 'mgz' formats
index	'parcellation' index, see 'FreeSurfer' look-up table
save_prefix	parent folder to save the resulting surface
label	character label or name of the 'sub-cortical' structure, usually automatically derived from index
IJK2RAS	an 'Affine' matrix from 'voxel' index to 'tkrRAS', usually automatically derived from atlas
grow	amount to grow (dilate) before generating mesh
remesh, smooth, smooth_delta, ...	passed to mesh_from_volume

Value

A surface mesh, containing 'atlas' index, label, surface nodes and face indices.

---

geom_freemesh *Creates any mesh geometry given vertices and face indices*

Description

Creates any mesh geometry given vertices and face indices

Usage

```
geom_freemesh(
  name,
  vertex = NULL,
  face = NULL,
  position = c(0, 0, 0),
  layer = 1,
  cache_file = NULL,
  group = NULL
)
```

Arguments

name	unique string in a scene to tell apart from different objects
vertex	position of each vertices (3 columns)
face	face indices indicating which 3 vertices to be linked (3 columns)
position	x,y,z location of the geometry

layer	visibility of the geometry, used when there are multiple cameras 1 is visible for all cameras
cache_file	cache vertex and face data into group
group	a GeomGroup object, if null, then the group will be generated automatically

Details

When generating a free mesh internally, a group must be specified, therefore if group is NULL here, then a group will be generated. However, it's always recommended to pass a group to the free mesh.

Author(s)

Zhengjia Wang

Examples

```
## Not run:
# Make sure you have N27 brain downloaded to `default_template_directory()`
# threeBrain::download_N27()

n27_dir = file.path(default_template_directory(), "N27")
surf_type = 'pial'

# Locate mesh files
lh = read_fs_asc(file.path(n27_dir, sprintf('surf/lh.%s.asc', surf_type)))
rh = read_fs_asc(file.path(n27_dir, sprintf('surf/rh.%s.asc', surf_type)))

# Create groups
group = create_group(name = sprintf('Surface - %s (N27)', surf_type))

# create mesh
lh_mesh = geom_freemesh(
  name = sprintf('FreeSurfer Left Hemisphere - %s (N27)', surf_type),
  vertex = lh$vertices[,1:3],
  face = lh$faces[,1:3],
  group = group
)
rh_mesh = geom_freemesh(
  name = sprintf('FreeSurfer Right Hemisphere - %s (N27)', surf_type),
  vertex = rh$vertices[,1:3],
  face = rh$faces[,1:3],
  group = group
)


# Render
if( interactive() ) { threejs_brain(lh_mesh, rh_mesh) }




## End(Not run)
```

geom_sphere *Create sphere geometry*

Description

Create sphere geometry

Usage

```
geom_sphere(
  name,
  radius,
  position = c(0, 0, 0),
  layer = 1,
  group = NULL,
  value = NULL,
  time_stamp = NULL
)
```

Arguments

name	unique string in a scene to tell apart from different objects
radius	size of sphere
position	x,y,z location of the sphere
layer	visibility of the geometry, used when there are multiple cameras 1 is visible for all cameras
group	a GeomGroup object
value, time_stamp	color of the sphere, used for animation/color rendering

Author(s)

Zhengjia Wang

Examples

```
# Create a sphere with animation
g = lapply(1:10, function(ii) {
  v = rep(ii, 10)
  v[1:ii] = 1:ii
 geom_sphere(paste0('s', ii), ii, value = v, position = c(11 * ii, 0,0), time_stamp = (1:10)/10)
})
if( interactive() ) { threejs_brain(.list = g) }
```

GeomGroup *R6 Class - Generate Group of Geometries*

Description

Container that collects geometry objects belonging to the same logical group in the three-brain viewer. Manages shared data, an optional 4-by-4 transformation matrix, spatial position, and a local file cache.

Public fields

`name` Unique character name of the group.

`layer` Camera layer(s); 0 = main camera only, 1 = all cameras, 13 = invisible.

`position` Numeric vector of length 3: group origin in world space.

`group_data` Named list of shared data attached to the group.

`trans_mat` Optional 4-by-4 transformation matrix (`NULL` for identity).

`cached_items` Character vector of group data keys stored in the file cache.

`cache_env` Local environment used for in-memory caching of loaded cache files.

`cache_path` Directory path for the file cache.

`disable_trans_mat` Logical; when `TRUE` the transformation matrix is ignored during rendering.

`parent_group` Name of the parent group, or `NULL`.

`subject_code` Subject identifier string, or `NULL`.

`.cache_name` Override for the sanitized cache directory name. When `NULL` the name is derived from `$name` by replacing non-alphanumeric characters with underscores.

Methods

Public methods:

- `GeomGroup$cache_name()`
- `GeomGroup$set_transform()`
- `GeomGroup$new()`
- `GeomGroup$set_group_data()`
- `GeomGroup$get_data()`
- `GeomGroup$to_list()`
- `GeomGroup$clone()`

`GeomGroup$cache_name()`: Return the sanitized cache directory name for this group. Uses `.cache_name` when set, otherwise derives the name from `$name` by replacing non-alphanumeric characters with underscores.

Usage:

```
GeomGroup$cache_name()
```

`GeomGroup$set_transform()`: Set or clear the 4-by-4 transformation matrix for the group.

Usage:

```
GeomGroup$set_transform(mat = NULL)
```

Arguments:

`mat` A 4-by-4 numeric matrix, or `NULL` to use the identity transform.

`GeomGroup$new()`: Create a new geometry group.

Usage:

```
GeomGroup$new(
  name,
  layer = 0,
  position = c(0, 0, 0),
  cache_path = tempfile(),
  parent = NULL
)
```

Arguments:

`name` Unique character name for the group.

`layer` Camera layer(s), 0-13. Default `0`.

`position` Numeric vector of length 3: group origin. Default `c(0, 0, 0)`.

`cache_path` Directory path for caching serialized data.

`parent` Name or `GeomGroup` object of the parent group, or `NULL`.

`GeomGroup$set_group_data()`: Attach a named data object to the group, optionally storing it in the file cache.

Usage:

```
GeomGroup$set_group_data(
  name,
  value,
  is_cached = FALSE,
  cache_if_not_exists = FALSE
)
```

Arguments:

`name` Key name for the data object.

`value` Data to store.

`is_cached` Logical; whether `value` is already a cache descriptor list.

`cache_if_not_exists` Logical; write `value` to the file cache when no cache file exists yet.

`GeomGroup$get_data()`: Retrieve a data object from the group by key, loading from the file cache when necessary.

Usage:

```
GeomGroup$get_data(key, force_reload = FALSE, ifnotfound = NULL)
```

Arguments:

`key` Name of the data object to retrieve.

`force_reload` Logical; reload from the file cache even when an in-memory copy exists.

`ifnotfound` Value returned when `key` is not found.

`GeomGroup$to_list()`: Serialize the group to a named list for JSON export.

Usage:

```
GeomGroup$to_list()
```

`GeomGroup$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GeomGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`get_digest_header` *Function to read digest header*

Description

Function to read digest header

Usage

```
get_digest_header(file, key, if_error = NULL, .list = NULL)
```

Arguments

`file`	file path to a ‘JSON‘ file
`key`	character, key to extract
`if_error`	value to return if key not found or read error occurs
`.list`	alternative list to supply if file is missing

get_ijk2ras — *Get 'voxel' to world matrix*

Description

Get 'voxel' to world matrix

Usage

```
get_ijk2ras(x, type = c("scanner", "tkr"))
```

Arguments

x	path to imaging files
type	world space type; choices are `'scanner'` (same as `'sform'` or `'qform'` in) or `'NIfTI'` file headers; or `'tkr'` (used to shared surface nodes)

Value

A four by four matrix

import-fs-suma — *Import 'FreeSurfer' or 'SUMA' files into the viewer structure*

Description

Import 'T1-MRI', surface files, curvature/'sulcus', atlas, and 'Talairach' transform matrix into 'json' format. These functions are not intended to be called directly, use `import_from_freesurfer` instead.

Usage

```
import_fs(
  subject_name,
  fs_path,
  quiet = FALSE,
  dtype = c("T1", "surface", "curv", "atlas_volume", "atlas_surface", "xform"),
  sub_type = NULL,
  hemisphere = c("l", "r"),
  ...
)

import_suma(
  subject_name,
  fs_path,
```

```
  quiet = FALSE,
  dtype = c("T1", "surface", "curv", "atlas_volume", "atlas_surface", "xform"),
  sub_type = NULL,
  hemisphere = c("l", "r"),
  ...
)
```

Arguments

subject_name	character, subject code
fs_path	path to 'FreeSurfer' folder
quiet, ...	passed from or to other methods.
dtype	data type to import, choices are 'T1', 'surface', 'curv', 'atlas_volume', 'atlas_surface', 'xform'
sub_type	detailed files to import. 'atlas_surface' is not supported for now
hemisphere	which hemisphere to import, ignored when dtype is in 'T1', 'atlas_volume', 'atlas_surface', 'xform'.

Value

logical, TRUE if the file is or has been cached, or FALSE if the file is missing.

---

import_from_freesurfer *Import from 'FreeSurfer' and create 'JSON' cache for 3D viewer*

Description

Import from 'FreeSurfer' and create 'JSON' cache for 3D viewer

Usage

```
import_from_freesurfer(fs_path, subject_name, quiet = FALSE)
```

Arguments

fs_path	'FreeSurfer' subject directory
subject_name	subject code
quiet	whether to suppress message or not

Value

None.

LineSegmentsGeom *R6 Class - Generate Line Segments*

Description

Line-segment geometry for drawing connections or paths in the three-brain viewer. Vertices may be static numeric coordinates or dynamic electrode references resolved at render time.

Super class

`AbstractGeom` -> `LineSegmentsGeom`

Public fields

`type` Geometry type string (`"linesegments"`).

`color` CSS hex color string(s) applied to line segments.

`size` Line width(s) in world-space units.

`vertices` Vertex positions: a numeric matrix (3 x N) for static segments, or a list of position/electrode descriptors for dynamic segments.

Active bindings

`dynamic` Logical; `TRUE` when vertices are electrode references resolved at render time rather than static coordinates.

Methods

Public methods:

- `LineSegmentsGeom$new()`
- `LineSegmentsGeom$set_vertices()`
- `LineSegmentsGeom$set_color()`
- `LineSegmentsGeom$set_size()`
- `LineSegmentsGeom$to_list()`
- `LineSegmentsGeom$clone()`

`LineSegmentsGeom$new()`**:** Create a new line-segment geometry.

Usage:

```
LineSegmentsGeom$new(name, dynamic = FALSE, ...)
```

Arguments:

`name` Unique character name.

`dynamic` Logical; when `TRUE` vertices are electrode references resolved at render time rather than fixed coordinates.

`...` Additional arguments forwarded to `AbstractGeom`.

`LineSegmentsGeom$set_vertices()`: Set vertex positions for the line segments.

Usage:

```
LineSegmentsGeom$set_vertices(..., .list = list(), append = FALSE)
```

Arguments:

`...` Positions as numeric vectors of length 3 (static mode) or named lists with `subject_code` and `electrode` elements (dynamic mode).

`.list` Additional positions supplied as a list.

`append` Logical; when `TRUE` append rather than replace.

`LineSegmentsGeom$set_color()`: Set the color(s) of the line segments.

Usage:

```
LineSegmentsGeom$set_color(...)
```

Arguments:

`...` One or more R color values (names, hex strings, or integer palette indices) that are interpolated across all segments.

`LineSegmentsGeom$set_size()`: Set the line width(s).

Usage:

```
LineSegmentsGeom$set_size(...)
```

Arguments:

`...` One or more positive numbers interpolated across all segments.

`LineSegmentsGeom$to_list()`: Serialize the line-segment geometry to a named list for JSON export.

Usage:

```
LineSegmentsGeom$to_list()
```

`LineSegmentsGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LineSegmentsGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

list_electrode_prototypes *List or load all electrode prototypes*

Description

List all built-in and user-customized electrode prototypes. User paths will be searched first, if multiple prototype configuration files are found for the same type.

Usage

```
list_electrode_prototypes()

load_prototype(type)
```

Arguments

type	electrode type, character

Value

`list_electrode_prototypes` returns a named list, names are the prototype types and values are the prototype configuration paths; `load_prototype` returns the prototype instance if `type` exists, or throw an error.

Examples

```
availables <- list_electrode_prototypes()
if( "sEEG-16" %in% names(availables) ) {
  proto <- load_prototype( "sEEG-16" )

  print(proto, details = FALSE)
}
```

localization_module *Launch a 'shiny' application to localize electrodes*

Description

If 'RAVE' has been installed, please use 'RAVE' modules. This function is purely for demonstration purposes.

Usage

```
localization_module(
  subject_code,
  fs_path,
  ct_path = NULL,
  surfaces = "pial",
  use_141 = TRUE,
  shiny_options = list(launch.browser = TRUE),
  save_path = tempfile(pattern = "electrode", fileext = ".csv"),
  ...,
  control_presets = NULL,
  side_display = FALSE,
  controllers = list()
)
```

Arguments

subject_code	subject code
fs_path	the subject's 'FreeSurfer' path
ct_path	the file path of 'CT' scans that have already been aligned to 'T1'; must be in 'NIFTI' format
surfaces	which surfaces to load
use_141	whether to try 'SUMA' standard 141 surface; default is true
shiny_options	shiny application options; see `options` in `shinyApp`
save_path	a temporary file where the electrode table should be cached; this file will be used to keep track of changes in case the application is crashed or shutdown
...	other parameters to pass into `threeBrain`
control_presets, side_display, controllers	passed to `threejs_brain`

Value

A list of `'ui'` elements, `'server'` function, and a stand-alone `'app'`

Examples

```
# This example require N27 template brain to be installed
# see `?download_N27` for details

# using N27 to localize
fs_path <- file.path(default_template_directory(), "N27")
if(interactive() && dir.exists(fs_path)) {
  module <- localization_module("N27", fs_path)

  print(module$app)
}
```

merge_brain *Create Multi-subject Template*

Description

Create Multi-subject Template

Usage

```
merge_brain(
  ...,
  .list = NULL,
  template_surface_types = NULL,
  template_subject = unname(getOption("threeBrain.template_subject", "N27")),
  template_dir = default_template_directory(),
  electrode_priority = c("asis", "sphere", "prototype", "both")
)
```

Arguments

`..., .list` `Brain2` objects

`template_surface_types` which template surface types to load, default is auto-guess

`template_subject` character, subject code to be treated as template, default is ‘N27‘

`template_dir` the parent directory where template subject is stored in

`electrode_priority` electrode shape priority, used to manage how electrodes are displayed; default is `'asis'` (no change) to be backward compatible; recommended option is `'sphere'` to display contacts as spheres (radius is based on the ’Radius’ column from electrode table); `'prototype'` should work in most cases but might be inaccurate since electrodes need to maintain the original geometry shape, but the template mapping might not be linear.

Author(s)

Zhengjia Wang

new_electrode_prototype

Create or load new electrode prototype from existing configurations

Description

Create or load new electrode prototype from existing configurations

Usage

```
new_electrode_prototype(base_prototype, modifier = NULL)
```

Arguments

base_prototype	base prototype, this can be a string of prototype type (see list_electrode_prototypes), path to the prototype configuration file, configuration in 'json' format, or an electrode prototype instance
modifier	internally used

Value

An electrode prototype instance

Examples

```
available_prototypes <- list_electrode_prototypes()
if("Precision33x31" %in% names(available_prototypes)) {

  # Load by type name
  new_electrode_prototype("Precision33x31")

  # load by path
  path <- available_prototypes[["Precision33x31"]]
  new_electrode_prototype(path)

  # load by json string
  json <- readLines(path)
  new_electrode_prototype(json)

}
```

plot_slices *Plot slices of volume*

Description

Plot slices of volume

Usage

```
plot_slices(
  volume,
  overlays = NULL,
  transform = NULL,
  positions = NULL,
  zoom = 1,
  pixel_width = 0.5,
  col = c("black", "white"),
  normalize = NULL,
  zclip = NULL,
  overlay_alpha = 0.3,
  zlim = normalize,
  main = "",
  title_position = c("left", "top"),
  fun = NULL,
  nc = NA,
  which = NULL,
  ...
)
```

Arguments

volume	path to volume (underlay)
overlays	images to overlay on top of the underlay, can be either a vector of paths to the overlay volume images, or a sequence of named lists. Each list item has `'volume'` (path to the volume) and `'color'` (color of the overlay)
transform	rotation of the volume in scanner `'RAS'` space
positions	vector of length 3 or matrix of 3 columns, the `'RAS'` position of cross-hairs
zoom	zoom-in radio, default is 1
pixel_width	output image pixel resolution; default is `0.5`, one pixel is 0.5 millimeters wide
col	color palette, can be a sequence of colors
normalize	range for volume data to be normalized; either `NULL` (no normalize) or a numeric vector of length two
zclip	clip image densities; if specified, values outside of this range will be clipped into this range

overlay_alpha	transparency of the overlay; default is 0.3
zlim	image plot value range, default is identical to normalize
main	image titles
title_position	title position; choices are "left" or "top"
fun	function with two arguments that will be executed after each image is drawn; can be used to draw cross-hairs or annotate each image
nc	number of "columns" in the plot when there are too many positions, must be positive integer; default is NA (automatically determined)
which	which plane to plot; default is NULL, which will trigger new plots and add titles; set to 1 for 'Axial' plane, 2 for 'Sagittal', and 3 for 'Coronal'.
...	additional arguments passing into image

Value

Nothing

---

read_fs_asc *Read ‘FreeSurfer‘ ascii file*

Description

Read ‘FreeSurfer‘ ascii file

Usage

```
read_fs_asc(file)
```

Arguments

file	file location

Value

a list of vertices and face indices

read_fs_labels — *Read FreeSurfer Annotations*

Description

Read FreeSurfer Annotations

Usage

```
read_fs_labels(path, vertex_number)
```

Arguments

path	label path
vertex_number	force to reset vertex number if raw file is incorrect

read_fs_m3z — *Read 'FreeSurfer' m3z file*

Description

Read 'FreeSurfer' m3z file

Usage

```
read_fs_m3z(filename)
```

Arguments

filename	file location, usually located at 'mri/transforms/talairach.m3z'

Details

An 'm3z' file is a 'gzip' binary file containing a dense vector field that describes a 3D registration between two volumes/images. This implementation follows the 'Matlab' implementation from the 'FreeSurfer'. This function is released under the 'FreeSurfer' license: https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferSoftwareLicense.

Value

registration data

read_fs_mgh_mgz — *Read ‘FreeSurfer‘ ‘mgz/mgh‘ file*

Description

Read ‘FreeSurfer‘ ‘mgz/mgh‘ file

Usage

```
read_fs_mgh_mgz(filename)
```

Arguments

filename	file location

Value

list contains coordinate transforms and volume data

read_gii2 — *Function to load surface data from ‘Gifti‘ files*

Description

The function ‘read_gii2‘ is a dynamic wrapper of Python ‘nibabel‘ loader. If no Python is detected, it will switch to ‘gifti::readgii‘.

Usage

```
read_gii2(path)
```

Arguments

path	‘Gifti‘ file path

Format

An R function acting as safe wrapper for `nibabel.load`.

read_mgz — *Function to load ‘FreeSurfer‘ ‘mgz/mgh‘ file*

Description

The function ‘read_mgz‘ is a dynamic wrapper of Python ‘nibabel‘ loader. If no Python is detected, it will switch to built-in function ‘read_fs_mgh_mgz‘, which has limited features.

Usage

```
read_mgz(path)
```

Arguments

path	‘mgz/mgh‘ file path

Format

An R function acting as safe wrapper for `nibabel.load`.

read_volume — *Read volume file in 'MGH' or 'Nifti' formats*

Description

Read volume file in `'MGH'` or `'Nifti'` formats

Usage

```
read_volume(file, format = c("auto", "mgh", "nii"), header_only = FALSE)
```

Arguments

file	file path
format	the file format
header_only	whether only read headers; default is false

Value

A list of volume data and transform matrices; if `header_only=TRUE`, then volume data will be substituted by the header.

renderBrain *Shiny Renderer for threeBrain Widgets*

Description

Shiny Renderer for threeBrain Widgets

Arguments

`expr`	R expression that calls three_brain function or Brain object
`env`	environment of expression to be evaluated
`quoted`	is expr quoted? Default is false.

Author(s)

Zhengjia Wang

reorient_volume *Function to reshape data to 'RAS' order*

Description

Function to reshape data to 'RAS' order

Usage

```
reorient_volume(volume, Torig)
```

Arguments

`volume`	3-mode tensor (voxels), usually from 'mgz', 'nii', or 'BRIK' files
`Torig`	a `4x4` transform matrix mapping volume ('CRS') to 'RAS'

Value

Reshaped tensor with dimensions corresponding to 'R', 'A', and 'S'

save_brain — *Save threeBrain widgets to local file system*

Description

Save threeBrain widgets to local file system

Usage

```
save_brain(widget, path, title = "3D Viewer", as_zip = FALSE, ...)
```

Arguments

widget	generated from function 'threejs_brain'
path	path to save the brain widget
title	widget title.
as_zip	whether to create zip file "compressed.zip".
...	ignored, used for backward compatibility

Author(s)

Zhengjia Wang

seeg_prototype — *Create 'sEEG' shaft geometry prototype*

Description

Intended for creating/editing geometry prototype, please see load_prototype to load existing prototype

Usage

```
seeg_prototype(
  type,
  center_position,
  contact_widths,
  diameter = 1,
  channel_order = seq_along(center_position),
  fix_contact = 1,
  overall_length = 200,
  description = NULL,
  dry_run = FALSE,
  default_interpolation = NULL,
```

```
  viewer_options = NULL,
  behnke_fried = FALSE,
  overwrite = FALSE
)
```

Arguments

type	type string and unique identifier of the prototype
center_position	numerical vector, contact center positions
contact_widths	numerical vector or length of one, width or widths of the contacts
diameter	probe diameter
channel_order	the channel order of the contacts; default is a sequence along the number
fix_contact	NULL or integer in channel_order, indicating which contact is the most important and should be fixed during the localization, default is 1 (inner-most target contact)
overall_length	probe length, default is 200
description	prototype description
dry_run	whether not to save the prototype configurations
default_interpolation	default interpolation string for electrode localization
viewer_options	list of viewer options; this should be a list of key-value pairs where the keys are the controller names and values are the corresponding values when users switch to localizing the electrode group
behnke_fried	whether the electrode has micro-wires at the tip; default is false
overwrite	whether to overwrite existing configuration file; default is false, which throws a warning when duplicated

Value

A electrode shaft geometry prototype; the configuration file is saved to 'RAVE' 3rd-party repository.

Examples

```
probe_head <- 2
n_contacts <- 12
width <- 2.41
contact_spacing <- 5
overall_length <- 400
diameter <- 1.12

contacts <- probe_head + width / 2 + 0:(n_contacts-1) * contact_spacing
proto <- seeg_prototype(
  type = "AdTech-sEEG-SD12R-SP05X-000",
  description = c(
    "AdTech sEEG - 12 contacts",
    "Contact length   : 2.41 mm",
```

```
    "Central spacing  : 5    mm",
    "Tip size         : 2    mm",
    "Diameter         : 1.12 mm"
  ),
  center_position = contacts,
  contact_widths = width,
  diameter = diameter,
  overall_length = overall_length,
  dry_run = TRUE
)

print(proto, details = FALSE)
```

SphereGeom *R6 Class - Generate Sphere Geometry*

Description

Sphere geometry for the three-brain viewer. Wraps a three-brain viewer sphere and supports animation clips for electrode-style data visualization.

Super class

AbstractGeom -> SphereGeom

Public fields

`type` Geometry type string (`"sphere"`).

`radius` Sphere radius in world-space units.

`width_segments` Number of horizontal segments (longitude).

`height_segments` Number of vertical segments (latitude).

Methods

Public methods:

- SphereGeom$new()
- SphereGeom$to_list()
- SphereGeom$clone()

`SphereGeom$new()`: Create a new sphere geometry.

Usage:

```
SphereGeom$new(name, position = c(0, 0, 0), radius = 5, ...)
```

Arguments:

`name` Unique character name.

`position` Numeric vector of length 3: sphere center. Default `c(0, 0, 0)`.

`radius` Sphere radius in world-space units. Default `5`.

`...` Additional arguments forwarded to `AbstractGeom`.

`SphereGeom$to_list()`: Serialize the sphere geometry to a named list for JSON export.

Usage:

```
SphereGeom$to_list()
```

`SphereGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SphereGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`SpriteGeom` *R6 Class - Generate Image Sprite Geometry*

Description

Image sprite geometry that positions a PNG texture billboard between two 3D points. The image is encoded as a data URI and rendered facing the camera in the three-brain viewer.

Super class

`AbstractGeom` -> `SpriteGeom`

Public fields

`clickable` Logical; always `FALSE` for sprite geometry.

`type` Geometry type string (`"imagesprite"`).

`image_uri` Base64 data URI of the PNG image used as the sprite texture.

`aspect_ratio` Width-to-height ratio of the source image, used to preserve proportions when scaling.

Methods

Public methods:

- `SpriteGeom$new()`
- `SpriteGeom$to_list()`
- `SpriteGeom$clone()`

`SpriteGeom$new()`: Create a new image sprite geometry.

Usage:

```
SpriteGeom$new(
  name,
  image_path,
  entry_position = c(1, 0, 0),
  target_position = c(0, 0, 0),
  ...
)
```

Arguments:

`name` Unique character name.

`image_path` Path to a PNG image file on disk.

`entry_position` Numeric vector of length 3: one end of the sprite axis (e.g. electrode entry point). Default `c(1, 0, 0)`.

`target_position` Numeric vector of length 3: the other end of the sprite axis (e.g. electrode target point). Default `c(0, 0, 0)`.

`...` Additional arguments forwarded to `AbstractGeom`.

`SpriteGeom$to_list()`: Serialize the sprite geometry to a named list for JSON export.

Usage:

```
SpriteGeom$to_list()
```

`SpriteGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SpriteGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

StreamlineGeom *R6 Class - Streamline Geometry*

Description

Geometry that points at a streamline (`tractography`) file on disk. The tract data is never read in R: the file path is registered as cached group data and the file is loaded and parsed directly by the 3D viewer.

Super class

`AbstractGeom` -> `StreamlineGeom`

Public fields

`type` Geometry type string (`"streamline"`).

`clickable` Logical; always `FALSE` for streamline geometry.

`color` Line color as an upper-case `'#RRGGBB'` string.

`streamline_name` Bundle name, using the on-disk spelling. Bundles stored in a sub-folder of the circuit keep that sub-path, for example `'left/AF'`.

`streamline_group` Circuit group name, using the on-disk spelling.

Active bindings

`data_key` Key under which the file descriptor is stored in the owning group. One circuit group holds several bundles, so the key is suffixed with `streamline_name`.

Methods

Public methods:

- `StreamlineGeom$new()`
- `StreamlineGeom$to_list()`
- `StreamlineGeom$clone()`

`StreamlineGeom$new()`: Create a streamline geometry from a file path.

Usage:

```
StreamlineGeom$new(
  name,
  path,
  streamline_name,
  streamline_group = "default",
  color = "#FF0000",
  group = GeomGroup$new(name = "default"),
  layer = 7,
  ...
)
```

Arguments:

`name` Unique character name of the geometry.

`path` Path to the streamline file.

`streamline_name` Bundle name, relative to the circuit folder; used to derive the cached data key.

`streamline_group` Circuit group name.

`color` Line color; any R color specification.

`group` `GeomGroup` that owns this geometry.

`layer` Camera layer. Default `7` (all cameras).

`...` Additional arguments forwarded to `AbstractGeom`.

`StreamlineGeom$to_list()`: Serialize the streamline geometry to a named list for `'JSON'` export, adding `color`, `streamline_name`, `streamline_group`, and `isStreamlineGeom`.

Usage:

```
StreamlineGeom$to_list()
```

`StreamlineGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
StreamlineGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`template_subject` *Download and Manage Template Subjects*

Description

Download and Manage Template Subjects

Usage

```
download_template_subject(
  subject_code = "N27",
  url,
  template_dir = default_template_directory()
)

download_N27(make_default = FALSE, ...)

set_default_template(
  subject_code,
```

```
  view = TRUE,
  template_dir = default_template_directory()
)

threebrain_finalize_installation(
  upgrade = c("ask", "always", "never", "data-only", "config-only"),
  async = TRUE
)

available_templates()
```

Arguments

subject_code	character with only letters and numbers (Important); default is ‘N27‘
url	zip file address; must be specified if subject_code is not from the followings: 'bert', 'cvs_avg35', 'cvs_avg35_inMNI152', 'fsaverage', 'fsaverage_sym', or 'N27'
template_dir	parent directory where subject's ‘FreeSurfer‘ folder should be stored
make_default	logical, whether to make ‘N27‘ default subject
...	more to pass to download_template_subject
view	whether to view the subject
upgrade	whether to check and download 'N27' brain interactively. Choices are 'ask', 'always', and 'never'
async	whether to run the job in parallel to others; default is true

Details

To view electrodes implanted in multiple subjects, it's highly recommended to view them in a template space The detail mapping method is discussed in function threeBrain.

To map to a template space, one idea is to find someone whose brain is normal. In our case, the choice is subject ‘N27‘, also known as ‘Colin 27‘. function download_N27 provides a simple and easy way to download a partial version from the Internet.

If you have any other ideas about template brain, you can use function set_default_template(subject_code, template_dir) to redirect to your choice. If your template brain is a ‘Zip‘ file on the Internet, we provide function download_template_subject to automatically install it.

Author(s)

Zhengjia Wang

TextDecorGeom *R6 Class - Text Decoration Geometry*

Description

R6 Class - Text Decoration Geometry

Super class

AbstractGeom -> TextDecorGeom

Public fields

`text` Label string rendered in the canvas

`font_size` World-space height of the sprite in mm

`color` CSS color string for the text (e.g. `"#ffffff"`)

`font_weight` CSS font-weight integer (e.g. 400 = normal, 700 = bold)

`decor_id` Stable ID used to identify this decoration in the viewer. Defaults to the geometry `name`.

`type` Geometry type string used by the JavaScript engine.

`clickable` Logical; whether the geometry responds to mouse clicks in the viewer.

Methods

Public methods:

- TextDecorGeom$new()
- TextDecorGeom$to_list()
- TextDecorGeom$clone()

`TextDecorGeom$new()`: Create a new text decoration geometry.

Usage:

```
TextDecorGeom$new(
  text = "",
  position = c(0, 0, 0),
  name = NULL,
  decor_id = NULL,
  font_size = 5,
  color = "#ffffff",
  font_weight = 400,
  layer = 1,
  ...
)
```

Arguments:

`text` Character string to display.

`position` Numeric vector of length 3: `c(x, y, z)` in world space.

`name` Unique geometry name. If `NULL` a random ID is generated automatically.

`decor_id` Stable decoration ID visible to the Shiny proxy. Defaults to `name`.

`font_size` World-space height of the sprite in mm. Default `5`.

`color` CSS color string. Default `"#ffffff"`.

`font_weight` CSS font-weight integer. Default `400`.

`layer` Camera layer(s), 0-13 (0 = main camera only, 1 = all cameras). Default `1`.

`...` Additional arguments forwarded to `AbstractGeom`.

`TextDecorGeom$to_list()`: Serialize to a list for JSON export.

Usage:

```
TextDecorGeom$to_list()
```

`TextDecorGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TextDecorGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`threeBrain` *Create a brain object*

Description

Create a brain object

Usage

```
threeBrain(
  path,
  subject_code,
  surface_types = c("pial", "smoothwm", "inflated", "sphere.reg"),
  atlas_types,
  annotation_types = "label/aparc.a2009s",
  ...,
  streamline_types = "default/",
  template_subject = unname(getOption("threeBrain.template_subject", "N27")),
  backward_compatible = getOption("threeBrain.compatible", FALSE)
)
```

Arguments

- `path` — path to 'FreeSurfer' directory, or 'RAVE' subject directory containing 'FreeSurfer' files, or simply a 'RAVE' subject
- `subject_code` — subject code, characters
- `surface_types` — surface types to load; default is `'pial'`, other common types are `'white'`, `'smoothwm'`, `'inflated'`, `'sphere.reg'`
- `atlas_types` — brain atlas to load; default is `'wmparc'`, or if not exists, `'aparc+aseg'`, other choices are `'aparc.a2009s+aseg'`, `'aparc.DKTatlas+aseg'`, depending on the atlas files in `'fs/mri'` folder
- `annotation_types` — annotations, this can be one or more files relative to the 'FreeSurfer' subject directory. Each annotation can be discrete such as surface atlas, or continuous such as surface curvature.
- `...` — reserved for future use
- `streamline_types` — streamline (`tractography`) bundles to load from the `'fs/streamline'` folder; passed straight to `add_streamline`, so each entry may name one bundle (`'motor/AF_left'`), a whole circuit (`'motor/'` or `'motor/*'`, which reaches into the circuit's sub-folders too), or just one of those sub-folders (`'motor/left/*'`). The default `'default/'` loads the `'default'` circuit, that is every streamline file placed directly under `'fs/streamline'` as well as under `'fs/streamline/default'` – the other circuits' folders are left alone. Use `NULL` to skip. Please note that each declared bundle is downloaded and parsed by the browser when the viewer starts, hence only one group is loaded by default; use `add_streamline` to add more.
- `template_subject` — template subject to refer to; used for group template mapping
- `backward_compatible` — whether to support old format; default is false

---

threejs_brain — *Create a Threejs Brain and View it in Browsers*

Description

Create a Threejs Brain and View it in Browsers

Usage

```
threejs_brain(
  ...,
  .list = list(),
  width = NULL,
  height = NULL,
  background = "#FFFFFF",
```

```
  cex = 1,
  timestamp = TRUE,
  title = "",
  side_canvas = FALSE,
  side_zoom = 1,
  side_width = 250,
  side_shift = c(0, 0),
  side_display = TRUE,
  control_panel = TRUE,
  control_presets = NULL,
  control_display = TRUE,
  camera_center = c(0, 0, 0),
  camera_pos = c(500, 0, 0),
  start_zoom = 1,
  symmetric = 0,
  default_colormap = "Value",
  palettes = NULL,
  value_ranges = NULL,
  value_alias = NULL,
  show_inactive_electrodes = TRUE,
 surface_colormap = system.file("palettes", "surface", "ContinuousSample.json", package
    = "threeBrain"),
 voxel_colormap = system.file("palettes", "datacube2", "FreeSurferColorLUT.json",
    package = "threeBrain"),
  videos = list(),
  widget_id = "threebrain_data",
  tmp_dirname = NULL,
  debug = FALSE,
  enable_cache = FALSE,
  token = NULL,
  controllers = list(),
  browser_external = TRUE,
  global_data = list(),
  global_files = list(),
  qrcode = NULL,
  custom_javascript = NULL,
  show_modal = "auto",
  embed = FALSE,
  remote_passkey = getOption("threeBrain.remote_passkey", NULL)
)
```

Arguments

`...`, `.list`	geometries inherit from AbstractGeom
`width`, `height`	positive integers. Width and height of the widget. By default width=‘100%‘, and height varies.
`background`	character, background color such as `"#FFFFFF"` or `"white"`
`cex`	positive number, relative text magnification level

`timestamp` logical, whether to show time-stamp at the beginning

`title` viewer title

`side_canvas` logical, enable side cameras to view objects from fixed perspective

`side_zoom` numerical, if side camera is enabled, zoom-in level, from 1 to 5

`side_width` positive integer, side panel size in pixels

`side_shift` integer of length two, side panel shift in pixels (‘CSS style‘: top, left)

`side_display` logical, show/hide side panels at beginning

`control_panel` logical, enable control panels for the widget

`control_presets` characters, presets to be shown in control panels

`control_display` logical, whether to expand/collapse control UI at the beginning

`camera_center` numerical, length of three, XYZ position where camera should focus at

`camera_pos` XYZ position of camera itself, default (0, 0, 500)

`start_zoom` numerical, positive number indicating camera zoom level

`symmetric` numerical, default 0, color center will be mapped to this value

`default_colormap` character, which color map name to display at startup

`palettes` named list, names corresponds to color-map names if you want to change color palettes

`value_ranges` named list, similar to `palettes`, value range for each values

`value_alias` named list, legend title for corresponding variable

`show_inactive_electrodes` logical, whether to show electrodes with no values

`surface_colormap` a color map or its path generated by `create_colormap(gtype="surface")` to render surfaces vertices; see `create_colormap` for details.

`voxel_colormap` a color map or its path generated by `create_colormap(gtype="volume")` to render volume such as atlases; see `create_colormap` for details.

`videos` named list, names corresponds to color-map names, and items are generated from `video_content`

`widget_id` character, internally used as unique identifiers for widgets; only use it when you have multiple widgets in one website

`tmp_dirname` character path, internally used, where to store temporary files

`debug` logical, internally used for debugging

`enable_cache` whether to enable cache, useful when rendering the viewers repeatedly in shiny applications

`token` unique character, internally used to identify widgets in ’JavaScript’ `'localStorage'`

`controllers` list to override the settings, for example `proxy$get_controllers()`

`browser_external` logical, use system default browser (default) or built-in one.

`global_data`, `global_files`
: internally use, mainly to store orientation matrices and files.

`qrcode`
: 'URL' to show in the 'QR' code; can be a character string or a named list of `'url'` and `'text'` (hyper-reference text)

`custom_javascript`
: customized temporary 'JavaScript' code that runs after ready state; available 'JavaScript' variables are:

  `'groups'` input information about each group

  `'geoms'` input information about each geometry

  `'settings'` input information about canvas settings

  `'scene'` 'threejs' scene object

  `'canvas'` canvas object

  `'gui'` controls data panel

  `'presets'` preset `'gui'` methods

`show_modal`
: logical or `"auto"`, whether to show a modal instead of direct rendering the viewers; designed for users whose browsers render the viewer with neither `WebGPU` nor `WebGL2`; only used in shiny applications

`embed`
: whether to try embedding the viewer in current run-time; default is false (will launch default web browser); set to true if running in `'rmarkdown'` or `'quarto'`, or to see the viewer in `'RStudio'` default panel.

`remote_passkey`
: the passkey shown in the viewer's `"Passkey"` field (control panel), `"17285"` or `"17285.key"`; default is the one `viewer_hub_start` stored in `options(threeBrain.remote_passkey)`. The viewer only joins the hub once the user clicks `"Connect"`; `save_brain` keeps the port but not the key. See `viewer_hub`.

Author(s)

Zhengjia Wang

Examples

```
if( interactive() ) {
library(threeBrain)

# Please use `download_N27` to download N27 Collins template brain
n27_path <- file.path(default_template_directory(), "N27")
if( dir.exists(n27_path) ) {

  brain <- threeBrain(path = n27_path, subject_code = "N27",
                              surface_types = c('pial', 'smoothwm'))
  print(brain)

  brain$plot(
    background = "#000000",
    controllers = list(
      'Voxel Type' = 'aparc_aseg',
      'Surface Type' = 'smoothwm',
      'Blend Factor' = 1,
```

```
        'Right Opacity' = 0.3,
        'Overlay Sagittal' = TRUE
      ),
      show_modal = TRUE
    )

  }
  }
```

---

threejsBrainOutput *Shiny Output for threeBrain Widgets*

Description

Shiny Output for threeBrain Widgets

Arguments

`outputId`	unique identifier for the widget
`width, height`	width and height of the widget. By default width="100 and height="500px".
`reportSize`	whether to report widget size in shiny `session$clientData`

Author(s)

Zhengjia Wang

TubeGeom *R6 Class - Generate Tube Geometry*

Description

Tube geometry that sweeps a circular cross-section along a curved path defined by control points. Used to render electrode shafts and tract trajectories in the three-brain viewer.

Super class

`AbstractGeom` -> `TubeGeom`

Public fields

`type` Geometry type string (`"tube"`).

`radial_segments` Number of segments around the tube circumference.

`tubular_segments` Number of segments along the tube path.

`control_data` Flattened numeric vector (row-major) of control points; each row encodes `x`, `y`, `z`, `t` (normalized path position 0-1), and `radius`.

`image_uri` Base64 data URI of the tube texture image, or `NULL` for a plain color.

Methods

Public methods:

- `TubeGeom$new()`
- `TubeGeom$to_list()`
- `TubeGeom$clone()`

`TubeGeom$new()`: Create a new tube geometry.

Usage:

```
TubeGeom$new(name, control_data, image_uri = NULL, ...)
```

Arguments:

`name` Unique character name.

`control_data` Numeric matrix with 5 columns: `x`, `y`, `z`, `t` (path position), `radius`. Must have at least 2 rows.

`image_uri` Optional base64 data URI string for a texture image.

`...` Additional arguments forwarded to `AbstractGeom`.

`TubeGeom$to_list()`: Serialize the tube geometry to a named list for JSON export.

Usage:

```
TubeGeom$to_list()
```

`TubeGeom$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TubeGeom$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Zhengjia Wang

`video_content` *Add video content to the viewer*

Description

Add video content to the viewer

Usage

```
video_content(
  path,
  duration = Inf,
  time_start = 0,
  asp_ratio = 16/9,
  local = TRUE
)
```

Arguments

`path`	local file path or 'URL'
`duration`	duration of the video
`time_start`	start time relative to the stimuli onset
`asp_ratio`	aspect ratio; default is `16/9`
`local`	used only when `path` is a 'URL': whether to download the video before generating the viewer; see 'Details'

Details

The video path can be either local file path or a 'URL' from websites. When path is from the internet, there are two options: download the video before generating the viewer, or directly use the 'URL'.

If download happens before a viewer is generated (`local=TRUE`), then the video content is local. The viewer will be self-contained. However, the distribution will contain the video, and the archive size might be large.

If raw 'URL' is used (`local=FALSE`), then viewer is not self-contained as the video link might break anytime. The 'screenshot' and 'record' function might be limited if the 'URL' has different domain than yours. However, the distribution will not contain the video, hence smaller. This works in the scenarios when it is preferred not to share video files or they are licensed, or simply distribution is limited. Besides, this method is slightly faster than the local alternatives.

`viewer_hub` *Let other programs drive the 3D viewers*

Description

A viewer can join a small hub that runs on this computer; R, `Python` or AI agents then send commands to the viewer through the hub and read back what it shows. These functions start, list and stop hubs. The hub is a `Node.js` program (version 18 or newer) that comes with this package; `viewer_hub_path()` gives its location.

Usage

```
viewer_hub_path()

viewer_hub_start(
  port = 17285L,
  key = NULL,
  generate_key = FALSE,
  idle_minutes = 30
)

viewer_hub_list(port = 17285L)

viewer_hub_stop(port = 17285L)
```

Arguments

port	the port of the hub; default is 17285
key	a key that the hub requires (letters, digits, "_" and "-"), or NULL for none
generate_key	whether the hub requires a random key; ignored when key is given
idle_minutes	the hub stops after this many minutes with nothing connected; 0 keeps it running

Details

viewer_hub_start() reuses the hub that already runs on port, or starts one in the background. It returns the hub's passkey: the port, or "port.key" when the hub requires a key, and stores it in options(threeBrain.remote_passkey), so viewers made afterwards show it in their "Passkey" field. To connect a viewer, click "Connect" below that field in its control panel; a viewer never connects by itself.

Without a key, any program or web page on this computer can connect to the hub; set generate_key = TRUE (or key) when that matters. The hub only listens on 127.0.0.1, so other computers cannot reach it.

Running hubs are recorded in tools::R_user_dir("threeBrain", "cache"), where other programs and AI agents find and join them. To let an agent drive the viewers, register the hub's MCP mode once, for example with Claude Code:

```
claude mcp add threebrain -- node /path/to/threebrain-hub.mjs mcp
```

where the path is viewer_hub_path(). The examples in the package repository (examples/WebSocket) show how R and Python scripts send commands.

The hub itself does not need R. Collaborators who received a viewer saved with save_brain run the same file through an agent plugin (examples/mcp in the package repository); their agent starts the hub and gives them the passkey.

Value

viewer_hub_path(): the path of the hub script. viewer_hub_start(): the passkey, invisibly. viewer_hub_list(): a data frame of the running hubs, with columns port, passkey (NA when the key is unknown), keyed, viewers, by (who started it) and started. viewer_hub_stop(): nothing.

Examples

```
## Not run:

passkey <- viewer_hub_start()

brain <- threeBrain(
  path = file.path(default_template_directory(), "N27"),
  subject_code = "N27", surface_types = "pial"
)
# In the viewer's control panel, click "Connect"
brain$plot()

viewer_hub_list()
```

```
viewer_hub_stop()

## End(Not run)
```

ViewerProxy *R6 Class - Shiny Proxy for the Three-Brain Viewer*

Description

Returned by brain_proxy. Provides reactive active bindings and setter methods for controlling a running three-brain viewer widget from within a Shiny application.

Active bindings (reactive fields) reflect the viewer state and trigger Shiny reactivity when read inside a reactive context:

background Current background color as a hex string.

text_decorations List of current text decoration objects.

main_camera Named list with position, up, and zoom.

side_display Logical indicating whether the side canvas is visible.

surface_type Current surface type string (e.g. "pial").

display_variable Name of the currently displayed data clip.

plane_position Named numeric vector c(R, A, S) giving the slice cursor position in FreeSurfer surface coordinates.

controllers Full controller state list.

controller_specs Controller types, choices, and ranges, keyed by controller name.

current_subject Named list describing the currently active subject, including transform matrices.

Active bindings

background Current viewer background color as a hex string (e.g. "#FFFFFF"), from controller "Background Color". Reactive.

text_decorations List of current text decoration parameter lists pushed from JavaScript. Each element has fields id, text, position, color, font_size, and layer. Reactive.

main_camera Named list with elements position (length-3 numeric), up (length-3 numeric), and zoom (positive numeric) describing the main camera state. Reactive.

side_display Logical indicating whether the side canvas panel is currently visible (controller "Show Panels"), or NULL when the viewer has no side panels. Reactive.

surface_type Current brain surface type string (e.g. "pial", "white"), from controller "Surface Type". Reactive.

display_variable Name of the data clip currently displayed in the viewer (controller "Display Data"). "[None]" when nothing is displayed. Reactive.

plane_position Named numeric vector c(R, A, S) giving the slice cursor position in FreeSurfer surface coordinates. Reactive.

`localization_table` Data frame of localization electrode contacts parsed from the JSON pushed by the viewer, or `NULL`. Reactive.

`localization_add_quaternion` Rotation data list pushed by the viewer when a localization point is added. Reactive.

`mouse_event_double_click` Named list describing the last double-click mouse event in the viewer. Reactive.

`mouse_event_click` Named list describing the last single-click mouse event in the viewer. Reactive.

`mouse_event_focus` Named list describing the object last focused with the viewer's focus mode (hold `'F'` and click). Unlike `$mouse_event_click`, focus mode reaches surfaces, volume slices, voxel volumes and streamlines, and does not change the viewer's selection. Entries include `'name'`, `'type'`, `'geom_type'`, `'subject'`, the focus point in `'tkrRAS'` and `'ScanRAS'` coordinates, and, where the geometry defines them, one-based `'vertex_index'`, `'face_index'` or `'line_index'` plus the underlay, display and threshold values under the cursor. Reactive.

`controllers` Full named list of the viewer's current controller (GUI panel) state. Reactive.

`controller_specs` Named list, keyed by controller name, that describes each controller of the viewer's control panel: `name`, `folder` (e.g. `"Surface Settings"`), `type` (`"boolean"`, `"number"`, `"option"`, `"color"`, `"string"`, `"function"`, `"interval"`, or `"linegraph"`), `choices` and `values` (options), `min`, `max`, and `step` (numbers), `hidden`, and `disabled`. The viewer resends it when its control panel changes (e.g. after new electrode data). Values are in field `controllers`. Reactive.

`current_subject` Named list describing the currently active subject. Includes `subject_code`, `Norig`, `Torig`, and `xfm` transform matrices. Reactive.

`sync` Sync token string pushed by the viewer on each render cycle; useful for triggering reactive updates. Reactive.

`acpc_alignment` Named list describing the AC-PC alignment in scanner RAS space. Includes `ac`, `pc`, `ras2acpc`, and set-flags. Reactive.

Methods

Public methods:

- `ViewerProxy$print()`
- `ViewerProxy$new()`
- `ViewerProxy$isolate()`
- `ViewerProxy$get_controllers()`
- `ViewerProxy$get_controller_specs()`
- `ViewerProxy$set_controllers()`
- `ViewerProxy$set_background()`
- `ViewerProxy$set_title()`
- `ViewerProxy$set_zoom_level()`
- `ViewerProxy$set_camera()`
- `ViewerProxy$set_display_data()`
- `ViewerProxy$set_focused_electrode()`
- `ViewerProxy$set_electrode_data()`

- ViewerProxy$set_electrode_palette()
- ViewerProxy$set_cex()
- ViewerProxy$set_localization_electrode()
- ViewerProxy$set_matrix_world()
- ViewerProxy$add_localization_electrode()
- ViewerProxy$clear_localization()
- ViewerProxy$set_incoming_localization_hemisphere()
- ViewerProxy$set_values()
- ViewerProxy$get_text_decorations()
- ViewerProxy$set_text_decoration()
- ViewerProxy$delete_text_decoration()
- ViewerProxy$get_crosshair_position()
- ViewerProxy$set_crosshair_position()

`ViewerProxy$print()`: Print a summary of the proxy's available fields and methods.

Usage:

```
ViewerProxy$print(...)
```

Arguments:

`...` Ignored.

`ViewerProxy$new()`: Create a new viewer proxy. Normally called via `brain_proxy` rather than directly.

Usage:

```
ViewerProxy$new(outputId, session = shiny::getDefaultReactiveDomain())
```

Arguments:

`outputId` Shiny output element ID of the three-brain widget.

`session` Shiny reactive domain session. Defaults to the current session returned by `shiny::getDefaultReactiveDomai`

`ViewerProxy$isolate()`: Read an active binding without triggering Shiny reactivity.

Usage:

```
ViewerProxy$isolate(name)
```

Arguments:

`name` Name of the active binding to read (character scalar).

`ViewerProxy$get_controllers()`: Get the current controller state as an isolated (non-reactive) list.

Usage:

```
ViewerProxy$get_controllers()
```

`ViewerProxy$get_controller_specs()`: Get the controller specifications (type, choices, range) as an isolated (non-reactive) list; see field `controller_specs`.

Usage:

```
ViewerProxy$get_controller_specs()
```

`ViewerProxy$set_controllers()`: Send a named list of controller key-value pairs to the viewer.

Usage:

```
ViewerProxy$set_controllers(ctrl)
```

Arguments:

`ctrl` Named list of controller settings to apply.

`ViewerProxy$set_background()`: Set the viewer background color.

Usage:

```
ViewerProxy$set_background(col)
```

Arguments:

`col` Any R color value accepted by `col2hexStr`.

`ViewerProxy$set_title()`: Display a title overlay in the viewer.

Usage:

```
ViewerProxy$set_title(title)
```

Arguments:

`title` Character string to display as the title. Pass an empty string or omit to clear the title.

`ViewerProxy$set_zoom_level()`: Set the viewer camera zoom level.

Usage:

```
ViewerProxy$set_zoom_level(zoom)
```

Arguments:

`zoom` Positive numeric zoom factor.

`ViewerProxy$set_camera()`: Point the main camera toward a direction in world space.

Usage:

```
ViewerProxy$set_camera(position, up)
```

Arguments:

`position` Numeric vector of length 3: desired camera direction (will be normalized to a unit vector then scaled to radius 500).

`up` Numeric vector of length 3: camera up direction. Defaults to `c(0, 0, 1)`.

`ViewerProxy$set_display_data()`: Change the displayed data clip and optional value range.

Usage:

```
ViewerProxy$set_display_data(variable = "", range = NULL)
```

Arguments:

`variable` Name of the data clip to display. Pass `""` to clear.

`range` Numeric vector of length 2 giving the display range, or `NULL` to use the natural range of the data.

`ViewerProxy$set_focused_electrode()`: Move the viewer focus to a specific electrode contact.

Usage:

```
ViewerProxy$set_focused_electrode(subject_code, electrode)
```

Arguments:

`subject_code` Character scalar: subject identifier.

`electrode` Integer scalar: electrode contact number.

`ViewerProxy$set_electrode_data()`: Push a data frame of electrode values to the viewer.

Usage:

```
ViewerProxy$set_electrode_data(
  data,
  palettes = NULL,
  value_ranges = NULL,
  clear_first = FALSE,
  update_display = TRUE,
  override = TRUE
)
```

Arguments:

`data` Data frame with at least columns `Subject` and `Electrode`.

`palettes` Optional named list of color palette vectors.

`value_ranges` Optional named list of `c(min, max)` ranges.

`clear_first` Logical; clear existing electrode data before applying. Default `FALSE`.

`update_display` Logical; update the viewer display after the data is applied. Default `TRUE`.

`override` Logical; override existing data for the same clip. Default `TRUE`.

`ViewerProxy$set_electrode_palette()`: Apply a color palette to an existing electrode data clip.

Usage:

```
ViewerProxy$set_electrode_palette(colors, variable)
```

Arguments:

`colors` Character vector of CSS color strings.

`variable` Name of the data clip to apply the palette to.

`ViewerProxy$set_cex()`: Set the global text magnification factor for the viewer.

Usage:

```
ViewerProxy$set_cex(cex = 1)
```

Arguments:

`cex` Positive numeric magnification factor. Default `1`.

`ViewerProxy$set_localization_electrode()`: Update the parameters of a localization electrode contact.

Usage:

```
ViewerProxy$set_localization_electrode(which, params, update_shiny = TRUE)
```

Arguments:

`which` Integer index of the electrode contact to update.

`params` Named list of parameter values to update.

`update_shiny` Logical; push the update to Shiny inputs. Default `TRUE`.

`ViewerProxy$set_matrix_world()`: Set the world transform matrix of a named scene object.

Usage:

```
ViewerProxy$set_matrix_world(name, m44)
```

Arguments:

`name` Character: name of the three-brain scene object.

`m44` Numeric vector of length 16 or a 4-by-4 matrix (row-major).

`ViewerProxy$add_localization_electrode()`: Add a new localization electrode contact to the viewer.

Usage:

```
ViewerProxy$add_localization_electrode(params, update_shiny = TRUE)
```

Arguments:

`params` Named list with at least `Coord_x`, `Coord_y`, `Coord_z` in FreeSurfer surface coordinates (unless `is_prototype = TRUE`).

`update_shiny` Logical; push the addition to Shiny inputs. Default `TRUE`.

`ViewerProxy$clear_localization()`: Remove all localization electrode contacts from the viewer.

Usage:

```
ViewerProxy$clear_localization(update_shiny = TRUE)
```

Arguments:

`update_shiny` Logical; push the change to Shiny inputs. Default `TRUE`.

`ViewerProxy$set_incoming_localization_hemisphere()`: Set which hemisphere is targeted for the next electrode localization click.

Usage:

```
ViewerProxy$set_incoming_localization_hemisphere(hemisphere)
```

Arguments:

`hemisphere` Character scalar: `"left"` or `"right"`.

`ViewerProxy$set_values()`: Add an animation clip of scalar or categorical values to a named geometry object in the viewer.

Usage:

```
ViewerProxy$set_values(
  name,
  target_object,
  data_type,
  value,
  palette = rainbow(64),
  symmetric = FALSE,
```

```
  time = ifelse(length(value) == 1, 0, stop("time must match length with value")),
   value_range = NULL,
   time_range = NULL,
   value_names = NULL,
   switch_display = FALSE
)
```

Arguments:

`name` Character clip name used as the display variable label.

`target_object` Name of the geometry object to animate.

`data_type` Either `"continuous"` or `"discrete"`.

`value` Numeric or character vector of values.

`palette` Color palette: a vector of R colors. Default `rainbow(64)`.

`symmetric` Logical; whether the color scale is symmetric around zero. Default `FALSE`.

`time` Numeric vector of time points matching `value`. Defaults to `0` for a single value.

`value_range` Numeric vector of length 2 overriding the data range. `NULL` uses the natural range.

`time_range` Numeric vector of length 2 overriding the time range. `NULL` uses the natural range.

`value_names` Optional character vector of level labels for discrete data.

`switch_display` Logical; switch the viewer display to this clip after upload. Default `FALSE`.

`ViewerProxy$get_text_decorations()`**:** Get the current text decorations from the viewer.

Returns a list of named lists, each with fields `id`, `text`, `position`, `color`, `font_size`, and `layer`. Returns an empty list when no decorations exist.

Usage:

```
ViewerProxy$get_text_decorations()
```

`ViewerProxy$set_text_decoration()`**:** Create or update a text decoration in the viewer.

Usage:

```
ViewerProxy$set_text_decoration(
  id,
  text = NULL,
  position = NULL,
  font_size = NULL,
  color = NULL,
  layer = NULL
)
```

Arguments:

`id` Character scalar: stable decoration ID.

`text` Character scalar: label to display.

`position` Numeric vector of length 3 in world space.

`font_size` Positive number: world-space sprite height (mm).

`color` CSS color string (e.g. `"#ff0000"`).

`layer` Integer or integer vector: camera layer(s).

`ViewerProxy$delete_text_decoration()`: Delete one or more text decorations from the viewer.

Usage:

```
ViewerProxy$delete_text_decoration(id)
```

Arguments:

`id` Character scalar or vector of decoration IDs to remove.

`ViewerProxy$get_crosshair_position()`: Get the current slice cursor position in a requested coordinate space.

Usage:

```
ViewerProxy$get_crosshair_position(
  space = c("tkrRAS", "MNI305", "MNI152", "scanner", "CRS")
)
```

Arguments:

`space` Coordinate space. One of `"tkrRAS"` (default), `"MNI305"`, `"MNI152"`, `"scanner"`, or `"CRS"`.

`ViewerProxy$set_crosshair_position()`: Move the slice cursor to a given position in a requested coordinate space.

Usage:

```
ViewerProxy$set_crosshair_position(
  position,
  space = c("tkrRAS", "MNI305", "MNI152", "scanner", "CRS")
)
```

Arguments:

`position` Numeric vector of length 3.

`space` Coordinate space of `position`. One of `"tkrRAS"` (default), `"MNI305"`, `"MNI152"`, `"scanner"`, or `"CRS"`.

Author(s)

Zhengjia Wang

`volume_to_surf` *Generate surface file from* `'nii'` *or* `'mgz'` *volume files*

Description

Generate surface file from `'nii'` or `'mgz'` volume files

Usage

```
volume_to_surf(
  volume,
  save_to = NA,
  lambda = 0.2,
  degree = 2,
  threshold_lb = 0.5,
  threshold_ub = NA,
  format = "auto",
  smooth_method = c("implicit", "explicit", "none"),
  smooth_iterations = 10L,
  max_vertices = 5e+05
)
```

Arguments

`volume` path to the volume file, or object from `read_volume`.

`save_to` where to save the surface file; default is `NA` (no save).

`lambda` `'Laplacian'` smooth, the higher the smoother

`degree` `'Laplacian'` degree; default is `2`

`threshold_lb` lower threshold of the volume (to create mask); default is `0.5`

`threshold_ub` upper threshold of the volume; default is `NA` (no upper bound). Voxels strictly between the two thresholds form the mask; voxels that are `NA`, `NaN`, or infinite are invalid and never part of it

`format` The format of the file if `save_to` is a valid path, choices include

- `'auto'` Default, supports `'FreeSurfer'` binary format and `'ASCII'` text format, based on file name suffix
- `'bin'` `'FreeSurfer'` binary format
- `'asc'` `'ASCII'` text format
- `'ply'` 'Stanford' `'PLY'` format
- `'off'` Object file format
- `'obj'` `'Wavefront'` object format
- `'gii'` `'GIfTI'` format. Please avoid using `'gii.gz'` as the file suffix
- `'mz3'` `'Surf-Ice'` format
- `'byu'` `'BYU'` mesh format
- `'vtk'` Legacy `'VTK'` format

`'gii'`, otherwise `'FreeSurfer'` format. Please do not use `'gii.gz'` suffix.

`smooth_method` `"implicit"` (default) smooths with `vcg_smooth_implicit` using `lambda` and `degree`; `"explicit"` smooths with `mris_smooth` instead, repeated neighbor averaging whose memory grows only linearly with the surface (with a **ravetools** version that does not have `mris_smooth`, the `"laplace"` type of `vcg_smooth_explicit` is used); `"none"` returns the surface without smoothing

`smooth_iterations` number of averaging rounds when `smooth_method` is `"explicit"`; default is `10`

max_vertices used only when smooth_method is "implicit", whose memory grows quickly with the surface size: surfaces with more vertices than this are reduced to about this many with ravetools::vcg_decimate() before smoothing, which removes vertices from flat regions first and keeps the shape; default is 500000. Because the smoothing works in mesh steps, the same lambda and degree smooth a reduced surface more; use a larger value or Inf to smooth at full resolution. With a **ravetools** version that does not have vcg_decimate, surfaces with more than 20000 vertices are not smoothed, since the implicit smoothing of those versions can crash on large surfaces

Value

Triangle 'rgl' mesh (vertex positions in native 'RAS'). If save_to is a valid path, then the mesh will be saved to this location. When no valid voxel lies within the thresholds, a warning is issued and the mesh has a single vertex at the origin and no face; it is still saved to save_to.

See Also

read_volume, vcg_isosurface, vcg_smooth_implicit

Examples

```
library(threeBrain)
N27_path <- file.path(default_template_directory(), "N27")
if(dir.exists(N27_path)) {
  aseg <- file.path(N27_path, "mri", "aparc+aseg.mgz")

  # generate surface for left-hemisphere insula
  mesh <- volume_to_surf(aseg, threshold_lb = 1034,
                         threshold_ub = 1036)

  if (interactive()) {
    ravetools::rgl_view({
      ravetools::rgl_call("shade3d", mesh, color = "yellow")
    })
  }
}
```

---

voxel_colormap *Color maps for volume or surface data*

Description

Color maps for volume or surface data

Usage

```
create_colormap(
  gtype = c("surface", "volume"),
  dtype = c("continuous", "discrete"),
  key,
  color,
  value,
  alpha = FALSE,
  con = NULL,
  auto_rescale = FALSE,
  ...
)

save_colormap(cmap, con)

freeserfer_colormap(con)

load_colormap(con)

read_colormap(con, format = c("rave", "itksnap"))
```

Arguments

gtype	geometry type, choices are "surface", "volume"
dtype	data type, "continuous" or "discrete"
key	non-negative integer vector corresponding to color values; its length must exceed 1; see 'Details'
color	characters, corresponding to color strings for each key
value	actual value for each key
alpha	whether to respect transparency
con	a file path to write results to or to read from. The file path can be passed as voxel_colormap into threejs_brain.
auto_rescale	automatically scale the color according to image values; only valid for continuous color maps
...	used by continuous color maps, passed to colorRampPalette
cmap	color map object
format	file format to read from

Details

Internal 'JavaScript' shader implementation uses integer color keys to connect color palettes and corresponding values. The keys must be non-negative.

Zero key is a special color key reserved by system. Please avoid using it for valid values.

Value

A list of color map information

Examples

```
# Creates a symmetric continuous colormap with 3 keys
# The color range is -10 to 10
# The colors are 'blue','white','red' for these keys

pal <- create_colormap(
  gtype = "volume", dtype = "continuous",
  key = c(1,2,3), value = c(-10,0,10),
  color = c('blue','white','red'))

print( pal )

# ---------------- Get colormap key from a value ------------

# returns key index starting from
pal$get_key( -10 )

# nearest value
pal$get_key( 2 )

# set threshold, key is now 0 (no color)
pal$get_key( 2, max_delta = 1 )


# ---------------- Save and load ----------------
f <- tempfile( fileext = '.json' )
save_colormap( pal, f )
cat(readLines(f), sep = '\n')

load_colormap(f)
```

`voxel_cube` *Generate volume data from 'MNI' coordinates*

Description

Generate volume data from 'MNI' coordinates

Usage

```
add_voxel_cube(
  brain,
  name,
  cube,
```

```
  size = c(256, 256, 256),
  trans_mat = NULL,
  trans_space_from = c("model", "scannerRAS"),
  color_format = c("RGBAFormat", "RedFormat")
)

add_nifti(
  brain,
  name,
  path,
  trans_mat = NULL,
  color_format = c("RGBAFormat", "RedFormat"),
  trans_space_from = c("model", "scannerRAS")
)

create_voxel_cube(
  mni_ras,
  value,
  colormap,
  keys = colormap$get_key(value),
  dimension = c(256, 256, 256)
)
```

Arguments

brain	a 'threeBrain' brain object generated from threeBrain or merge_brain. If you have 'rave' package installed, the brain can be generated from rave::rave_brain2
name	the name of voxel cube, only letters, digits and '_' are allowed; other characters will be replaced by '_'
cube	a 3-mode array; see the following example
size	the actual size of the volume, usually dot multiplication of the dimension and voxel size
trans_mat	the transform matrix of the volume. For add_voxel_cube, this matrix should be from data cube geometry model center to world ('tkrRAS') transform. For add_nifti, this matrix is the 'Nifti' 'RAS' to world ('tkrRAS') transform.
trans_space_from	where does trans_mat transform begin; default is from object 'model' space; alternative space is 'scannerRAS', meaning the matrix only transform volume cube from its own 'scannerRAS' to the world space.
color_format	color format for the internal texture. Default is 4-channel 'RGBAFormat'; alternative choice is 'RedFormat', which saves volume data with single red-channel to save space
path	'Nifti' data path
mni_ras	'MNI' 'RAS' coordinates, should be a n-by-3 matrix
value	data values (length n); used if keys is missing
colormap	a color map generated from create_colormap; see voxel_colormap for details

keys	integer color-keys generated from a color map with length of n; alternatively, you could specify value and colormap to generate keys automatically
dimension	volume dimension; default is a 256 x 256 x 256 array cube; must be integers and have length of 3

Value

create_voxel_cube returns a list of cube data and other informations; add_voxel_cube returns the brain object

Examples

```
# requires N27 brain to be installed
# use `download_N27()` to download template Collins brain


# sample MNI coords
tbl <- read.csv(system.file(
  'sample_data/example_cube.csv', package = 'threeBrain'
))
head(tbl)

# load colormap
cmap <- load_colormap(system.file(
  'palettes/datacube2/Mixed.json', package = 'threeBrain'
))

x <- create_voxel_cube(
  mni_ras = tbl[, c('x', 'y', 'z')],
  keys = tbl$key,
  dimension = c(128, 128, 128)
)


n27_path <- file.path(default_template_directory(), "N27")
if( interactive() && dir.exists(n27_path) ) {
  brain <- merge_brain()

  # or add_voxel_cube(brain, 'example', x$cube)
  x$add_to_brain(brain, 'example')

  brain$plot(controllers = list(
    "Voxel Type" = 'example',
    'Right Opacity' = 0.3,
    'Left Opacity' = 0.3,
    'Background Color' = '#000000'
  ), voxel_colormap = cmap)
}
```

Index

AbstractGeom, 3, *9*, *15*, *17*, *31*, *45*, *46*, *48*, *51*, *57*
add_nifti *(*voxel_cube*)*, 72
add_streamline, 6, *53*
add_voxel_cube *(*voxel_cube*)*, 72
available_templates *(*template_subject*)*, 49

BlankGeom, 9
brain_proxy, 10, *61*, *63*
brain_setup, 11

calculate_rotation, 11
check_freesurfer_path, 12
colorRampPalette, *71*
conform_volume, 12
create_colormap, *55*
create_colormap *(*voxel_colormap*)*, 70
create_group, 13
create_voxel_cube *(*voxel_cube*)*, 72
cross_prod, 15

DataCubeGeom, 15, *17*
DataCubeGeom2, 17
default_template_directory, 19
domains, *10*
download_N27, *19*
download_N27 *(*template_subject*)*, 49
download_template_subject *(*template_subject*)*, 49

fill_surface, *21*
FreeGeom, 19
freeserfer_colormap *(*voxel_colormap*)*, 70
freesurfer_lut, 20

generate_cortical_parcellation, 20
generate_smooth_envelope, 21
generate_subcortical_surface, 22
geom_freemesh, 23
geom_sphere, 25
GeomGroup, 26
get_digest_header, 28
get_ijk2ras, 29

image, *38*
import-fs-suma, 29
import_from_freesurfer, *29*, 30
import_fs *(*import-fs-suma*)*, 29
import_suma *(*import-fs-suma*)*, 29

LineSegmentsGeom, 31
list_electrode_prototypes, 33, *36*
load_colormap *(*voxel_colormap*)*, 70
load_prototype, *43*
load_prototype *(*list_electrode_prototypes*)*, 33
localization_module, 33

merge_brain, *6*, 35, *73*
mesh_from_volume, *23*
mris_smooth, *69*

N27 *(*template_subject*)*, 49
new_electrode_prototype, 36

plot_slices, 37

read.fs.surface, *21*
read_colormap *(*voxel_colormap*)*, 70
read_fs_asc, 38
read_fs_labels, 39
read_fs_m3z, 39
read_fs_mgh_mgz, 40
read_gii2, 40
read_mgz, 41
read_volume, 41, *69*, *70*
renderBrain, 42
reorient_volume, 42

save_brain, 43, *56*, *60*
save_colormap *(*voxel_colormap*)*, 70

seeg_prototype, 43
set_default_template
(template_subject), 49
shinyApp, *34*
SphereGeom, 45
SpriteGeom, 46
StreamlineGeom, 48

template_subject, 49
TextDecorGeom, 51
threeBrain, *6*, *34*, 52, *73*
threebrain_finalize_installation
(template_subject), 49
threejs_brain, *34*, 53, *71*
threejsBrainOutput, 57
TubeGeom, 57

vcg_isosurface, *70*
vcg_smooth_explicit, *69*
vcg_smooth_implicit, *69, 70*
video_content, *55*, 58
viewer_hub, *56*, 59
viewer_hub_list (viewer_hub), 59
viewer_hub_path (viewer_hub), 59
viewer_hub_start, *56*
viewer_hub_start (viewer_hub), 59
viewer_hub_stop (viewer_hub), 59
ViewerProxy, 61
volume_to_surf, *22*, 68
voxel_colormap, 70, *73*
voxel_cube, 72

write.fs.surface, *21*