

Package: neurotransform (via r-universe)

October 3, 2026

Title Geometric Transforms for Neuroimaging Data

Version 0.2.0

Author Brad Buchsbaum [aut, cre]

Maintainer Brad Buchsbaum <brad.buchsbaum@gmail.com>

Description A self-contained, dependency-light library for geometric transforms in neuroimaging. Provides affine and nonlinear (warp) mappings between coordinate systems, plus volume-to-surface mapping primitives. Emphasizes correctness, elegance, and composability with a category-theoretic flavor.

License MIT + file LICENSE

URL <https://github.com/bbuchsbaum/neurotransform>

BugReports <https://github.com/bbuchsbaum/neurotransform/issues>

Depends R (>= 4.0.0)

Imports methods, Rcpp (>= 1.0.0), digest, neuroim2

Suggests RNifti, testthat (>= 3.0.0), Matrix, hdf5r, jsonlite, knitr, neurosurf, rmarkdown

Remotes bbuchsbaum/neurosurf

Additional_repositories <https://bbuchsbaum.r-universe.dev>

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

Collate 'RcppExports.R' 'zzz.R' 'aaa_utils.R' 'all_class.R' 'all_generic.R' 'coordinates.R' 'transform_io.R' 'lta_io.R' 'x5_io.R' 'io_helpers.R' 'warp_loader.R' 'morphism.R' 'jacobian.R' 'sampler.R' 'overlap_metrics.R' 'warp_transform.R' 'warp_chain.R' 'transform_path.R' 'afni_io.R' 'afni_warp.R' 'fsl_ingest.R' 'affine_utils.R' 'deformation_field.R' 'resampling_plan.R' 'surface_resampling.R' 'surface_area.R'

Config/pak/sysreqs cmake libicu-dev zlib1g-dev
Repository <https://bbuchsbaum.r-universe.dev>
Date/Publication 2026-09-26 20:15:03 UTC
RemoteUrl <https://github.com/bbuchsbaum/neurotransform>
RemoteRef HEAD
RemoteSha 933edddda462593941e167726e8aaa7168ff103a

Contents

adjoint	5
affine_from_components	6
affine_utils	6
Affine3DMorphism	6
Affine3DMorphism-class	7
afni_aff12_to_ras	7
afni_is_oblique	8
afni_read_aff12	9
afni_read_aff12_array	9
afni_warp	10
afni_write_aff12	10
all_class	11
all_generic	12
ants_h5_morphism	12
apply_affine	13
apply_resampling_plan	13
apply_surface_adjoint	14
apply_surface_resampling	15
as_morphism_path	16
backproject_surface_to_volume	16
build_affine_matrix	17
compose	18
compose_affines	19
compute_hash	19
convert_affine_convention	20
coordinates	21
cpp_path_apply_affine_warp_affine	21
cpp_path_apply_steps	22
decompose_affine_matrix	22
deformation_field	23
detect_fnirt_def_type	23
detect_transform_type	24
dice_coefficient	25
extract_affine	26
fsl_flirt_to_internal_affine	27
fsl_ingest	28
get_linear_factory	28

get_loader	28
Grid-class	29
grid_coords	29
grid_from_data	30
grid_of	30
grid_spec	31
has_adjoint	31
IdentityMorphism	32
IdentityMorphism-class	32
invert	33
invert_affine	33
is_affine_matrix	34
is_affine_morphism	34
is_invertible	35
is_linear_morphism	35
is_valid_path	36
is_warp_morphism	36
jaccard_index	37
jacobian	38
jacobian_det	39
jacobian_det_field	40
JacobianField-class	41
list_loaders	42
lps_to_ras	42
lta_io	43
make_resampling_plan	43
mesh_is_sphere	44
mesh_set_radius	44
morphism	45
Morphism-class	45
morphism_hash	46
morphism_kind	46
MorphismPath-class	47
overlap_metrics	47
ras_to_lps	48
ras_to_tkras	49
read_affine_matrix_txt	49
read_image	50
read_linear_transform	50
read_linear_transform_array	51
read_transform	52
read_x5	53
register_loader	53
resample	54
resample_to	55
resample_volume	55
resampling_plan	56
reverse_surface_resampling_plan	56

sample_volume_on_surface	57
sampled_points	57
SampledPoints-class	58
sampler	58
Sampler-class	59
shape_zoom_affine	59
source_of	60
surface_mesh	60
surface_resampling	61
surface_resampling_plan	61
surface_sampler	62
surface_vertex_areas	63
SurfaceMesh-class	64
SurfToSurfMorphism	64
SurfToSurfMorphism-class	65
target_of	66
tkras_to_ras	66
transform	67
transform_coords	68
transform_io	68
transform_path	69
TransformFileError	70
TransformIOError	70
validate_path	71
validate_surface_mesh	71
VolToSurfMorphism	72
VolToSurfMorphism-class	73
volume_sampler	73
warp_from_field	74
warp_loader	75
Warp3DMorphism	75
Warp3DMorphism-class	77
write_affine_matrix_txt	77
write_image	78
write_linear_transform	78
write_linear_transform_array	79
write_transform	80
write_warp_field	80
write_x5	81
x5_domain	81
x5_io	82
x5_transform	82

adjoint	<i>Get adjoint (generalized inverse)</i>
---------	--

Description

Returns an adjoint morphism when defined.

Usage

```
adjoint(object, ...)

## S4 method for signature 'Morphism'
adjoint(object, ...)

## S4 method for signature 'IdentityMorphism'
adjoint(object, ...)

## S4 method for signature 'Affine3DMorphism'
adjoint(object, ...)

## S4 method for signature 'Warp3DMorphism'
adjoint(object, ...)

## S4 method for signature 'VolToSurfMorphism'
adjoint(object, ...)

## S4 method for signature 'SurfToSurfMorphism'
adjoint(object, ...)
```

Arguments

object	A Morphism object
...	Additional arguments

Details

For invertible morphisms, `adjoint()` is defined and equals `invert()`. For other morphisms, `adjoint()` may not be implemented and will error.

Value

Adjoint morphism or error if not available

`affine_from_components`

Wrap an affine matrix built from components into an Affine3DMorphism

Description

Wrap an affine matrix built from components into an Affine3DMorphism

Usage

```
affine_from_components(source, target, ...)
```

Arguments

<code>source</code>	Source domain identifier
<code>target</code>	Target domain identifier
<code>...</code>	Arguments passed to <code>build_affine_matrix</code>

Value

Affine3DMorphism object

`affine_utils`

Affine utilities (matrix-level and morphism helpers)

Description

Lightweight helpers for building, decomposing, and testing affine transforms without introducing external dependencies or tool-specific semantics.

`Affine3DMorphism`

Create a 3D affine morphism

Description

Represents an affine transformation between volume spaces. The matrix maps target coords -> source coords (pullback semantics).

Usage

```
Affine3DMorphism(source, target, matrix, cost = 1, method_tag = "anatomical")
```

Arguments

source	Source domain hash
target	Target domain hash
matrix	4x4 affine matrix (target -> source)
cost	Path cost (default 1.0)
method_tag	Method tag (default "anatomical")

Value

Affine3DMorphism object

Examples

```
# Translation by (10, 20, 30)
mat <- diag(4)
mat[1:3, 4] <- c(10, 20, 30)
aff <- Affine3DMorphism("native", "mni", mat)
```

Affine3DMorphism-class

3D affine morphism

Description

Represents an affine transformation (rotation, translation, scaling, shear) between volume spaces.

Slots

matrix 4x4 affine transformation matrix (maps target -> source)

afni_aff12_to_ras

Convert AFNI RAI/DICOM affine to RAS

Description

Converts an AFNI RAI/DICOM affine matrix to our internal RAS convention by flipping the X and Y axes. Optionally applies AFNI-style deobliquing compensation when source/target image affines are provided. Standard 3dAllineate -1Dmatrix_save matrices map base/target coordinates to source/input coordinates, which is already the package's pullback direction.

Usage

```
afni_aff12_to_ras(
  mat_rai,
  source_affine = NULL,
  target_affine = NULL,
  oblique_correction = TRUE
)
```

Arguments

mat_rai 4x4 affine in RAI coordinates

source_affine Optional 4x4 source (moving) voxel-to-world affine

target_affine Optional 4x4 target (reference) voxel-to-world affine

oblique_correction Logical; apply AFNI cardinal rotation correction

Value

4x4 affine in RAS coordinates

Examples

```
mat_rai <- diag(4)
mat_ras <- afni_aff12_to_ras(mat_rai)
```

afni_is_oblique	<i>Detect whether an affine is oblique (not cardinal axis-aligned)</i>
-----------------	--

Description

Detect whether an affine is oblique (not cardinal axis-aligned)

Usage

```
afni_is_oblique(affine, threshold_deg = 0.01)
```

Arguments

affine 4x4 voxel-to-world affine

threshold_deg Angular threshold in degrees

Value

Logical

afni_read_aff12	<i>Read AFNI .aff12.1D affine matrix</i>
-----------------	--

Description

Reads a single affine from a .aff12.1D file and returns it as a 4x4 matrix in AFNI RAI/DICOM coordinates. AFNI axis labels describe the negative end of each axis, so numeric RAI coordinates increase toward Left, Posterior, and Superior (LPS).

Usage

```
afni_read_aff12(path, row = NULL)
```

Arguments

path	Path to .aff12.1D file
row	Optional 1-based row to select from a multi-row file. Required when the file holds more than one matrix.

Value

4x4 affine matrix (in RAI)

See Also

[afni_read_aff12_array](#) for per-volume matrix series

Examples

```
## Not run:
mat_rai <- afni_read_aff12("transform.aff12.1D")

## End(Not run)
```

afni_read_aff12_array	<i>Read every affine row of an AFNI .aff12.1D file</i>
-----------------------	--

Description

Reads all matrix rows, returning one 4x4 matrix per row in AFNI RAI/DICOM coordinates. Comment lines are returned in the "comments" attribute.

Usage

```
afni_read_aff12_array(path)
```

Arguments

path Path to a .aff12.1D (or .1D) matrix file

Value

A list of 4x4 matrices in RAI, with a "comments" attribute holding the file's comment lines

See Also

[afni_read_aff12](#) for a single-matrix file

Examples

```
path <- tempfile(fileext = ".aff12.1D")
afni_write_aff12(list(diag(4), diag(4)), path)
length(afni_read_aff12_array(path))
```

afni_warp	<i>AFNI Warp and Affine Handlers</i>
-----------	--------------------------------------

Description

AFNI-specific coordinate convention handling. AFNI 3dQwarp stores displacement fields in DICOM/LPS convention. These functions handle the LPS <-> RAS conversion for warps and RAI <-> RAS for affines.

Coordinate Systems

- RAI/DICOM (used by AFNI affines and warp displacements): +X=Left, +Y=Posterior, +Z=Superior (numerically LPS)
- RAS (NIfTI/neuroim2): +X=Right, +Y=Anterior, +Z=Superior

LPS to RAS: negate X and Y (Z is the same).

afni_write_aff12	<i>Write AFNI .aff12.1D affine matrices</i>
------------------	---

Description

Writes matrices in AFNI's canonical layout: one row of twelve row-major numbers per matrix, optionally preceded by a banner comment. This is the layout 3dAllineate -1Dmatrix_apply expects.

Usage

```
afni_write_aff12(x, path, comment = TRUE)
```

Arguments

x	A 4x4 (or 3x4) matrix in RAI, or a list of them
path	Output path
comment	Logical, or a character vector of comment lines to write as a header. TRUE writes the package's banner; FALSE writes none.

Value

Invisibly, path

Examples

```
path <- tempfile(fileext = ".aff12.1D")
afni_write_aff12(diag(4), path)
```

all_class

S4 Class Definitions for neurotransform

Description

S4 class definitions for the neurotransform package. This file defines the morphism class hierarchy. neurotransform is a pure transform kernel; domain and geometry abstractions live in higher-level packages.

Class Hierarchy

```
Morphism (virtual)
|-- IdentityMorphism
|-- Affine3DMorphism
|-- Warp3DMorphism
|-- VolToSurfMorphism
|-- SurfToSurfMorphism
```

```
MorphismPath (lightweight list wrapper for composed paths)
```

Direction Convention

A morphism $A \rightarrow B$ maps coordinates in B to coordinates in A (pullback semantics). This is the natural convention for resampling: to get a value at target location, you need to know where to sample from in the source.

all_generic

Generic Function Definitions

Description

Generic function definitions for the neurotransform package. These define the core API for coordinate transforms.

ants_h5_morphism

Load ANTs composite H5 as morphism or path

Description

Creates a morphism (or MorphismPath) from an ANTs composite H5 transform file. These files typically contain both a displacement field and an affine transform.

Usage

```
ants_h5_morphism(
  path,
  source = "source",
  target = "target",
  apply_affine = TRUE
)
```

Arguments

path	Path to ANTs H5 file
source	Source coordinate space identifier
target	Target coordinate space identifier
apply_affine	If TRUE and an embedded affine is present, return a MorphismPath combining warp and affine. If FALSE, return only the warp.

Value

Warp3DMorphism (if no affine or apply_affine=FALSE), MorphismPath, or Affine3DMorphism for an affine-only H5 file. For an affine-only file, apply_affine=FALSE is an error because there is no displacement field.

Pullback Ordering

When apply_affine=TRUE, the H5 component order is preserved in the returned MorphismPath. Both ITK composite transforms and MorphismPath evaluate the stored components from last to first, so preserving the order reproduces ITK point-transform semantics.

apply_affine	<i>Apply affine transformation to coordinates</i>
--------------	---

Description

Generic function to apply a 4x4 affine matrix to 3D coordinates.

Usage

```
apply_affine(coords, affine)
```

Arguments

coords	Numeric matrix (N x 3) or vector of length 3
affine	4x4 affine transformation matrix

Value

Transformed coordinates

Examples

```
coords <- matrix(c(10, 20, 30), ncol = 3)
affine <- diag(4)
affine[1:3, 4] <- c(5, 10, 15) # translation
apply_affine(coords, affine)
```

apply_resampling_plan	<i>Apply a resampling plan to a volume</i>
-----------------------	--

Description

Apply a resampling plan to a volume

Usage

```
apply_resampling_plan(
  plan,
  source_data,
  outside = NA_real_,
  modulate = c("none", "jacobian", "sqrt_jacobian")
)
```

Arguments

plan	ResamplingPlan
source_data	3D or 4D array in source_grid geometry
outside	Value for out-of-bounds sampling
modulate	Jacobian modulation: "none", "jacobian", "sqrt_jacobian"

Value

Array with target_grid dimensions (and trailing data dim if 4D input)

apply_surface_adjoint *Apply the Euclidean adjoint of a surface plan*

Description

Applies the transpose of the exact forward operator, including fixed masks and the selected normalization, without renormalizing its transpose. This is an algebraic adjoint, not reverse geometric interpolation or reconstruction. Unsupported source columns receive algebraic zero; details=TRUE identifies them. Missing-data omission is not a fixed linear operator and is unsupported.

Usage

```
apply_surface_adjoint(
  plan,
  x,
  normalize = c("element", "sum", "none"),
  details = FALSE
)
```

Arguments

plan	A SurfaceResamplingPlan.
x	Numeric vector or matrix with one row per reference vertex.
normalize	Forward-operator normalization: "element", "sum", or "none".
details	Return values and supported-source-column diagnostics.

Value

Vector/matrix or a list with values and available.

 apply_surface_resampling

Apply a surface resampling plan to vertex data

Description

Geometry, source ROI mass and finite-data mass are distinct. Unsupported or masked outputs are unavailable (NA), while supported numerical zero remains zero. Zero-weight missing inputs never contaminate an output. Omission returns the retained raw weight mass as an attribute, or in the detailed result.

Usage

```
apply_surface_resampling(
  plan,
  x,
  inverse = FALSE,
  normalize = c("element", "sum", "none"),
  na_policy = c("propagate", "omit", "error"),
  data_type = c("continuous", "label"),
  label_method = c("aggregate", "largest"),
  label_table = NULL,
  unassigned = NA_integer_,
  details = FALSE
)
```

Arguments

plan	A SurfaceResamplingPlan.
x	Numeric vertex vector or matrix with <code>n_moving</code> rows.
inverse	Legacy transpose followed by normalization. Deprecated; use <code>apply_surface_adjoint()</code> or construct a reverse plan explicitly.
normalize	"element" normalizes each output row; "sum" normalizes represented input columns; "none" retains stored weights. Column normalization preserves represented discrete mass before target masking; it is not anatomical area correction.
na_policy	"propagate", "omit" with finite-weight renormalization, or "error". Omission requires <code>normalize="element"</code> .
data_type	"continuous" or "label" (integer keys).
label_method	"aggregate" or "largest".
label_table	Optional data frame with a unique integer key column. Other columns, such as names and colors, are preserved unchanged.
unassigned	Integer output key for unavailable labels, or NA.
details	Return values and per-column coverage diagnostics.

Details

Label keys are never averaged. Aggregate voting sums weights for each key; largest voting selects one source vertex. Ties select the smallest label key for aggregate voting and the smallest source vertex index for largest voting.

Value

Vector/matrix, or a list when details=TRUE. Diagnostics include raw source and finite-data weight mass before normalization and target masking. Adaptive plans also report source_roi_target_area before target masking. effective_target_area is available only for row normalization, and is column specific under omission. With finite continuous data and no target exclusion it gives the area identity described in the surface vignette.

as_morphism_path	<i>Coerce list/paths to MorphismPath</i>
------------------	--

Description

File paths are read with read_transform() and no further arguments. FSL warps need image geometry that a path alone does not carry, so read them with read_transform(..., source_affine, source_dim) and pass the resulting morphisms instead.

Usage

```
as_morphism_path(...)
```

Arguments

... Morphism objects or file paths to transforms

Value

MorphismPath object

backproject_surface_to_volume	<i>Backproject surface values into a volume (adjoint of VolToSurf sampling)</i>
-------------------------------	---

Description

Implements a simple backprojection operator that distributes each surface value into the source volume grid using the same interpolation weights that forward sampling would use.

Usage

```
backproject_surface_to_volume(
    surface_values,
    morphism,
    grid,
    surface_coords = NULL,
    method = c("linear", "nearest", "ribbon"),
    outside = 0
)
```

Arguments

surface_values	Numeric vector (length N) or matrix (N x k) of values at surface points
morphism	VolToSurfMorphism
grid	Source Grid (volume geometry)
surface_coords	Optional N x 3 world coordinates; defaults to morphism@params\$mid_coords
method	"linear", "nearest", or "ribbon"
outside	Value used for out-of-bounds (default 0)

Details

This is the transpose/adjoint of the linear sampling operator for `sample_volume_on_surface(..., method="linear")` when the same `surface_coords` are used.

Value

3D array (or 4D if multivariate `surface_values`) in grid geometry

build_affine_matrix	<i>Build a 4x4 affine matrix from components</i>
---------------------	--

Description

Build a 4x4 affine matrix from components

Usage

```
build_affine_matrix(
    translation = c(0, 0, 0),
    scales = c(1, 1, 1),
    skews = c(0, 0, 0),
    angles = c(0, 0, 0),
    anchor = c("none", "origin", "centre", "center")
)
```

Arguments

translation	length-3 translation vector (applied last)
scales	length-3 scaling
skews	length-3 shear terms (xy, xz, yz)
angles	length-3 rotation angles (radians) about x, y, z (applied in Z-Y-X order)
anchor	Either "none"/"origin" or a numeric length-3 point. The "centre"/"center" keywords are not supported because this helper has no reference extent; pass the explicit anchor point instead.

Value

4x4 numeric matrix

compose	<i>Compose two morphisms</i>
---------	------------------------------

Description

Returns the composition $g \circ f$ (f applied first, then g). The result is a `MorphismPath` that can be passed to `transform()`.

Usage

```
compose(f, g)

## S4 method for signature 'Morphism,Morphism'
compose(f, g)

## S4 method for signature 'MorphismPath,Morphism'
compose(f, g)

## S4 method for signature 'Morphism,MorphismPath'
compose(f, g)

## S4 method for signature 'MorphismPath,MorphismPath'
compose(f, g)
```

Arguments

f	First morphism (applied first)
g	Second morphism (applied second)

Value

`MorphismPath` representing the composition

Examples

```
aff1 <- Affine3DMorphism("a", "b", diag(4))
aff2 <- Affine3DMorphism("b", "c", diag(4))
path <- compose(aff1, aff2) # a -> b -> c
```

compose_affines	<i>Compose two affine transformations</i>
-----------------	---

Description

Returns the composition $B(A(x)) = BA * x$.

Usage

```
compose_affines(A, B)
```

Arguments

A	First affine (applied first)
B	Second affine (applied second)

Value

Composed 4x4 affine matrix

Examples

```
A <- diag(4); A[1:3, 4] <- c(1, 2, 3)
B <- diag(4); B[1:3, 4] <- c(4, 5, 6)
compose_affines(A, B)
```

compute_hash	<i>Compute a stable hash for objects</i>
--------------	--

Description

Creates a reproducible hash using digest with `serialize=TRUE`.

Usage

```
compute_hash(..., algo = "xxhash64")
```

Arguments

...	Objects to hash
algo	Hash algorithm (default "xxhash64" for speed)

Value

Character string hash

Examples

```
compute_hash("hello", "world")
compute_hash(list(a = 1, b = 2))
```

convert_affine_convention

*Convert affine conventions between "generic" (target→source world)
and "fsl" (source→target vox)*

Description

Convert affine conventions between "generic" (target→source world) and "fsl" (source→target vox)

Usage

```
convert_affine_convention(
  matrix,
  source_affine,
  target_affine,
  source_dim = NULL,
  target_dim = NULL,
  from = c("generic", "fsl"),
  to = c("generic", "fsl")
)
```

Arguments

matrix	4x4 matrix in the from convention
source_affine	4x4 voxel-to-world for source image
target_affine	4x4 voxel-to-world for target image
source_dim	Optional source image dimensions (for FSL handedness swap)
target_dim	Optional target image dimensions (for FSL handedness swap)
from	Input convention ("generic", "fsl")
to	Output convention ("generic", "fsl")

coordinates	<i>Coordinate Convention Utilities</i>
-------------	--

Description

Functions for converting between coordinate conventions used in neuroimaging.

Internal convention: RAS (Right-Anterior-Superior) in millimeters.

Supported conventions

- **RAS**: Right-Anterior-Superior (internal standard)
- **LPS**: Left-Posterior-Superior (ANTs/ITK convention)
- **tkRAS**: FreeSurfer tkregister RAS
- **FSL**: FSL world coordinates (mm, may need scaling)

cpp_path_apply_affine_warp_affine

Apply affines -> warp -> affines to coordinates (target -> source)

Description

Apply affines -> warp -> affines to coordinates (target -> source)

Usage

```
cpp_path_apply_affine_warp_affine(
    coords,
    affines_pre,
    field,
    dim,
    world_to_vox,
    cubic,
    affines_post
)
```

Arguments

coords	n x 3 target world coords
affines_pre	list of 4x4 matrices applied before warp
field	displacement field
dim	field dims
world_to_vox	warp world_to_vox
cubic	use cubic sampling
affines_post	list of 4x4 matrices applied after warp

Value

n x 3 source world coords

cpp_path_apply_steps *Apply arbitrary steps (affine/warp) to coordinates (target -> source)*

Description

Apply arbitrary steps (affine/warp) to coordinates (target -> source)

Usage

cpp_path_apply_steps(coords, steps)

Arguments

coords n x 3 target world coords
 steps list of steps with kind \"affine\" or \"warp\" and associated data

Value

n x 3 transformed coords

decompose_affine_matrix
Decompose a 4x4 affine matrix into components

Description

Decompose a 4x4 affine matrix into components

Usage

decompose_affine_matrix(matrix)

Arguments

matrix 4x4 numeric matrix

Value

list with translation, rotation_matrix, scales, skews, angles

deformation_field	<i>Deformation fields and Jacobian maps</i>
-------------------	---

Description

Helpers to materialize morphisms on voxel grids for visualization or export, keeping the core morphism API untouched.

Usage

```
deformation_field(morphism, grid, with_jacobian = TRUE, log = FALSE)
```

Arguments

morphism	A Morphism (affine or warp)
grid	A Grid object (see grid_spec/grid_from_data)
with_jacobian	Logical; attach Jacobian determinant volume
log	If TRUE, store log-Jacobian

Value

An array with `dim = c(grid@dims, 3)` and class "DeformationField"

detect_fnirt_def_type	<i>Detect FNIRT deformation type (relative vs absolute)</i>
-----------------------	---

Description

Determines whether a dense FSL warp stores relative displacements or absolute source coordinates, using two kinds of evidence on a regular subsample of non-zero lattice vectors:

1. Field of view (when `source_affine` and `source_dim` are given): the reading whose implied source FSL coordinates fall inside the source image (within one voxel) for a clearly larger fraction of samples (by at least 0.25) wins.
2. Jacobian: field values are fitted as $v \approx Mx + t$ in reference FSL coordinates, so the implied reference-to-source Jacobian is M (absolute) or $M + I$ (relative). A reading wins when the other implies a volume change at least twice as extreme ($|\log |\det J||$ larger by $\log 2$).

The field-of-view test decides first. If neither is decisive the function stops rather than guess; pass `def_type` explicitly in that case. Scaled or axis-permuted mappings (e.g. a small or sagittally stored source) can make the Jacobian test ambiguous, which the field-of-view test resolves.

Usage

```
detect_fnirt_def_type(
  warp_path,
  sample_n = 1000,
  threshold_mm = NULL,
  source_affine = NULL,
  source_dim = NULL
)
```

Arguments

warp_path	Path to FNIRT warp file
sample_n	Approximate number of lattice points sampled
threshold_mm	Deprecated and ignored; retained for compatibility.
source_affine, source_dim	Optional source image voxel-to-RAS affine and dimensions, enabling the field-of-view test.

Value

"relative" or "absolute"

Examples

```
## Not run:
def_type <- detect_fnirt_def_type("warp.nii.gz", source_affine = src_affine,
                                source_dim = src_dim)

## End(Not run)
```

detect_transform_type *Detect transform type from a file path*

Description

NIfTI vector fields are classified from the header before the filename: FSL intent codes 2006 (dense field) and 2007–2009 (coefficients) and the ITK vector layout (5D with intent 1007) are definitive. Otherwise filename hints apply, then the layout: a 4D (X, Y, Z, 3) field is FSL's layout (ITK/ANTs cannot read it), and remaining fields default to ANTs. Dense FSL fields need source image geometry, so an FSL result is only usable with source_affine and source_dim.

Usage

```
detect_transform_type(path, source_affine = NULL, target_affine = NULL)
```

Arguments

path	Transform file path
source_affine	Optional source affine used to disambiguate linear files
target_affine	Optional target affine used to disambiguate linear files

Value

Character scalar naming the detected transform family

dice_coefficient	<i>Dice coefficient for two images</i>
------------------	--

Description

Convenience wrapper returning the Dice coefficient from `overlap_metrics()`.

Usage

```
dice_coefficient(
  x,
  y,
  threshold = 0,
  label = NULL,
  na.rm = TRUE,
  empty = c("one", "zero", "na"),
  check_geometry = TRUE,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

x, y	Arrays or image objects with matching dimensions.
threshold	Numeric scalar used to form foreground masks when <code>label</code> is <code>NULL</code> .
label	Optional single label value. If supplied, Dice is computed for that label instead of by thresholding.
na.rm	Logical; if <code>TRUE</code> , paired missing or non-finite numeric voxels are removed before computing masks.
empty	How to score Dice, Jaccard, overlap coefficient, and volume similarity when both masks are empty. "one" treats two empty masks as perfect agreement, "zero" returns 0, and "na" returns <code>NA_real_</code> .
check_geometry	Logical; if <code>TRUE</code> , compare available voxel-to-world affines after confirming that dimensions match.
tolerance	Numeric tolerance used for affine comparisons.

Value

Numeric scalar Dice coefficient.

Examples

```
x <- array(c(TRUE, TRUE, FALSE, FALSE), dim = c(2, 2, 1))
y <- array(c(TRUE, FALSE, TRUE, FALSE), dim = c(2, 2, 1))
dice_coefficient(x, y)
```

extract_affine	<i>Extract affine matrix from data</i>
----------------	--

Description

Extract affine matrix from data

Usage

```
extract_affine(x)

## S4 method for signature 'array'
extract_affine(x)

## S4 method for signature 'matrix'
extract_affine(x)

## S4 method for signature 'ANY'
extract_affine(x)

## S4 method for signature 'DenseNeuroVol'
extract_affine(x)

## S4 method for signature 'DenseNeuroVec'
extract_affine(x)
```

Arguments

x	Data object
---	-------------

Value

4x4 affine matrix

`fsl_flirt_to_internal_affine`*Convert FLIRT matrix to internal affine*

Description

Given a FLIRT matrix (source_FSL -> ref_FSL), returns the internal ref_world -> src_world affine for pullback semantics.

Usage

```
fsl_flirt_to_internal_affine(  
  flirt_mat,  
  source_affine,  
  ref_affine,  
  source_dim = NULL,  
  ref_dim = NULL  
)
```

Arguments

<code>flirt_mat</code>	4x4 FLIRT matrix
<code>source_affine</code>	4x4 voxel-to-world for source image
<code>ref_affine</code>	4x4 voxel-to-world for reference image
<code>source_dim</code>	Optional source image dimensions (needed for handedness swap)
<code>ref_dim</code>	Optional reference image dimensions (needed for handedness swap)

Value

4x4 internal affine (ref_world -> src_world)

Examples

```
## Not run:  
flirt_mat <- as.matrix(read.table("xform.mat"))  
src_aff <- diag(4) # from source image header  
ref_aff <- diag(4) # from reference image header  
internal <- fsl_flirt_to_internal_affine(flirt_mat, src_aff, ref_aff)  
  
## End(Not run)
```

fsl_ingest	<i>FSL Coordinate Convention Handlers</i>
------------	---

Description

FSL-specific coordinate convention handling. FSL uses scaled voxel coordinates for FLIRT matrices and has specific conventions for FNIRT warp fields (relative vs absolute).

get_linear_factory	<i>Get linear transform IO backend factory</i>
--------------------	--

Description

Returns a small factory descriptor for linear transform IO, including canonical format and reader/writer function handles.

Usage

```
get_linear_factory(fmt, is_array = TRUE)
```

Arguments

fmt	Format name (e.g. "itk", "ants", "fsl", "afni", "generic")
is_array	Logical; if TRUE, returns array-IO handlers

Value

List with format, reader, and writer

get_loader	<i>Retrieve a warp loader by name</i>
------------	---------------------------------------

Description

Gets a loader function from the registry.

Usage

```
get_loader(name = NULL)

get_warp_loader(name = NULL)
```

Arguments

name Loader name (default "neuroim2")

Value

Loader function

Examples

```
loader <- get_loader("neuroim2")
```

Grid-class	<i>Grid: regular coordinate grid specification</i>
------------	--

Description

Grid: regular coordinate grid specification

Usage

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

Arguments

object A Grid object (for show method)

Slots

dims Integer: grid dimensions
affine 4x4 voxel-to-world matrix

grid_coords	<i>Get world coordinates for a grid</i>
-------------	---

Description

Get world coordinates for a grid

Usage

```
grid_coords(grid)
```

Arguments

grid Grid object

Value

Numeric matrix (prod(dims) x 3)

grid_from_data	<i>Get grid specification from volume data</i>
----------------	--

Description

Get grid specification from volume data

Usage

```
grid_from_data(data)
```

Arguments

data Volume with geometry

Value

Grid object

grid_of	<i>Extract Grid from volume or grid-like input</i>
---------	--

Description

Extract Grid from volume or grid-like input

Usage

```
grid_of(x)
```

Arguments

x neuroim2 volume, array, or Grid

Value

Grid

grid_spec	Create a grid specification
-----------	-----------------------------

Description

Create a grid specification

Usage

```
grid_spec(dims, affine)
```

Arguments

dims	Integer vector of dimensions
affine	4x4 voxel-to-world matrix

Value

Grid object

has_adjoint	Check if morphism has an adjoint
-------------	----------------------------------

Description

Returns TRUE if `adjoint()` is available for this morphism. For linear/invertible morphisms, the adjoint is the inverse.

Usage

```
has_adjoint(object)
```

Arguments

object	A Morphism object
--------	-------------------

Value

Logical

IdentityMorphism	Create an identity morphism
------------------	-----------------------------

Description

The identity morphism maps a domain to itself. Cost is 0.

Usage

IdentityMorphism(domain)

Arguments

domain Domain hash string

Value

IdentityMorphism object

Examples

```
id <- IdentityMorphism("my_domain_hash")
```

IdentityMorphism-class	Identity morphism
------------------------	-------------------

Description

The identity morphism maps a domain to itself. Required for categorical structure and useful for composition.

invert	<i>Invert a morphism</i>
--------	--------------------------

Description

Returns the inverse morphism if available (`inverse_type == "exact"` or `"approximate"`). For morphisms with approximate or adjoint inverses, use `adjoint()` instead.

Usage

```
invert(object)

## S4 method for signature 'Morphism'
invert(object)

## S4 method for signature 'IdentityMorphism'
invert(object)

## S4 method for signature 'Affine3DMorphism'
invert(object)

## S4 method for signature 'Warp3DMorphism'
invert(object)
```

Arguments

`object` A Morphism object

Value

Inverted morphism

Examples

```
aff <- Affine3DMorphism("a", "b", diag(4))
inv <- invert(aff) # b -> a
```

invert_affine	<i>Invert an affine transformation</i>
---------------	--

Description

Invert an affine transformation

Usage

```
invert_affine(affine)
```

Arguments

affine 4x4 affine matrix

Value

Inverted 4x4 affine matrix

Examples

```
affine <- diag(4)
affine[1:3, 4] <- c(5, 10, 15)
inv <- invert_affine(affine)
```

is_affine_matrix	<i>Check if an object is a numeric 4x4 affine matrix</i>
------------------	--

Description

Check if an object is a numeric 4x4 affine matrix

Usage

```
is_affine_matrix(x)
```

Arguments

x Object to test

Value

Logical

is_affine_morphism	<i>Check if an object is an Affine3DMorphism</i>
--------------------	--

Description

Check if an object is an Affine3DMorphism

Usage

```
is_affine_morphism(x)
```

Arguments

x Object to test

Value

Logical

is_invertible	<i>Check if morphism is exactly invertible</i>
---------------	--

Description

Returns TRUE only if a geometric inverse is available via `invert()`.

Usage

```
is_invertible(object)
```

Arguments

object	A Morphism object
--------	-------------------

Value

Logical

Examples

```
aff <- Affine3DMorphism("a", "b", diag(4))
is_invertible(aff) # TRUE
```

is_linear_morphism	<i>Check if morphism is linear (affine or identity)</i>
--------------------	---

Description

Check if morphism is linear (affine or identity)

Usage

```
is_linear_morphism(object)
```

Arguments

object	A Morphism object
--------	-------------------

Value

Logical

is_valid_path	<i>Check if a morphism path is valid</i>
---------------	--

Description

Returns TRUE if the path is composable, FALSE otherwise.

Usage

```
is_valid_path(path)
```

Arguments

path List of Morphism objects

Value

Logical

Examples

```
aff1 <- Affine3DMorphism("a", "b", diag(4))
aff2 <- Affine3DMorphism("b", "c", diag(4))
is_valid_path(list(aff1, aff2)) # TRUE
```

is_warp_morphism	<i>Check if morphism is a warp field</i>
------------------	--

Description

Check if morphism is a warp field

Usage

```
is_warp_morphism(object)
```

Arguments

object A Morphism object

Value

Logical

jaccard_index	<i>Jaccard index for two images</i>
---------------	-------------------------------------

Description

Convenience wrapper returning the Jaccard index from `overlap_metrics()`.

Usage

```
jaccard_index(
  x,
  y,
  threshold = 0,
  label = NULL,
  na.rm = TRUE,
  empty = c("one", "zero", "na"),
  check_geometry = TRUE,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

<code>x, y</code>	Arrays or image objects with matching dimensions.
<code>threshold</code>	Numeric scalar used to form foreground masks when <code>label</code> is <code>NULL</code> .
<code>label</code>	Optional single label value. If supplied, Dice is computed for that label instead of by thresholding.
<code>na.rm</code>	Logical; if <code>TRUE</code> , paired missing or non-finite numeric voxels are removed before computing masks.
<code>empty</code>	How to score Dice, Jaccard, overlap coefficient, and volume similarity when both masks are empty. "one" treats two empty masks as perfect agreement, "zero" returns 0, and "na" returns <code>NA_real_</code> .
<code>check_geometry</code>	Logical; if <code>TRUE</code> , compare available voxel-to-world affines after confirming that dimensions match.
<code>tolerance</code>	Numeric tolerance used for affine comparisons.

Value

Numeric scalar Jaccard index.

Examples

```
x <- array(c(1, 1, 0, 0), dim = c(2, 2, 1))
y <- array(c(1, 0, 1, 0), dim = c(2, 2, 1))
jaccard_index(x, y)
```

jacobian

*Jacobian Computations***Description**

Compute Jacobian matrices and determinants for morphisms. The Jacobian is the derivative of the coordinate transformation.

Usage

```
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'IdentityMorphism,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'Affine3DMorphism,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'Warp3DMorphism,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'MorphismPath,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'VolToSurfMorphism,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'SurfToSurfMorphism,ANY'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))

## S4 method for signature 'DeformationField,missing'
jacobian(morphism, coords, mode = c("pullback", "pushforward"))
```

Arguments

morphism	A Morphism or MorphismPath
coords	Numeric matrix (n x 3) of coordinates
mode	"pullback" (default) or "pushforward"

Value

JacobianField object

Semantics

For a morphism with pullback semantics (transform maps target $\rightarrow$ source), the pullback Jacobian is $d(\text{source})/d(\text{target})$, evaluated at target coordinates.

Modes

- "pullback": $d(\text{source})/d(\text{target})$ - natural for resampling
- "pushforward": $d(\text{target})/d(\text{source})$ - requires invertible morphism

Examples

```
aff <- Affine3DMorphism("a", "b", diag(4))
coords <- matrix(c(0, 0, 0, 10, 20, 30), ncol = 3, byrow = TRUE)
J <- jacobian(aff, coords)
```

jacobian_det

*Compute Jacobian determinants***Description**

Compute Jacobian determinants

Usage

```
jacobian_det(
  morphism,
  coords,
  log = FALSE,
  mode = c("pullback", "pushforward")
)

## S4 method for signature 'IdentityMorphism'
jacobian_det(
  morphism,
  coords,
  log = FALSE,
  mode = c("pullback", "pushforward")
)

## S4 method for signature 'Affine3DMorphism'
jacobian_det(
  morphism,
  coords,
  log = FALSE,
  mode = c("pullback", "pushforward")
)

## S4 method for signature 'Warp3DMorphism'
jacobian_det(
  morphism,
  coords,
```

```

    log = FALSE,
    mode = c("pullback", "pushforward")
  )

  ## S4 method for signature 'MorphismPath'
  jacobian_det(
    morphism,
    coords,
    log = FALSE,
    mode = c("pullback", "pushforward")
  )

```

Arguments

morphism	A Morphism or MorphismPath
coords	Numeric matrix (n x 3) of coordinates
log	Return log(det(J))?
mode	"pullback" or "pushforward"

Value

Numeric vector of determinants

Examples

```

aff <- Affine3DMorphism("a", "b", diag(4) * 2)
coords <- matrix(c(0, 0, 0), ncol = 3)
jacobian_det(aff, coords) # 8 (2^3)
jacobian_det(aff, coords, log = TRUE) # log(8)

```

jacobian_det_field	<i>Extract Jacobian determinant volume from a morphism on a grid</i>
--------------------	--

Description

Extract Jacobian determinant volume from a morphism on a grid

Usage

```
jacobian_det_field(morphism, grid, log = FALSE)
```

Arguments

morphism	A Morphism (affine or warp)
grid	A Grid object
log	Logical; if TRUE, return log-Jacobian determinants

Value

Array with grid dimensions containing Jacobian determinants

JacobianField-class	<i>Jacobian field: collection of 3x3 Jacobian matrices</i>
---------------------	--

Description

Jacobian field: collection of 3x3 Jacobian matrices

Extract single Jacobian matrix or subset

Number of Jacobians

Compute determinants

Invert Jacobians (for mode switching)

Usage

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

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

## S4 method for signature 'JacobianField'
length(x)

## S4 method for signature 'JacobianField'
det(x, ...)

## S4 method for signature 'JacobianField,missing'
solve(a, b, ...)
```

Arguments

object	A JacobianField object (for show method)
x	A JacobianField object (for other methods)
i	Numeric index for subsetting
j	Second index (optional)
...	Additional arguments
drop	Logical; if TRUE drop dimensions
a	A JacobianField object (for solve method)
b	Missing (solve method signature)

Slots

values Numeric array (n x 3 x 3) of Jacobian matrices
 coords Numeric matrix (n x 3) of coordinate locations
 mode Character: "pullback" or "pushforward"
 morphism_hash Hash of source morphism

list_loaders	<i>List all registered loaders</i>
--------------	------------------------------------

Description

List all registered loaders

Usage

```
list_loaders()
```

Value

Character vector of registered loader names

Examples

```
list_loaders()
```

lps_to_ras	<i>Convert LPS coordinates to RAS</i>
------------	---------------------------------------

Description

ANTs and ITK use LPS (Left-Posterior-Superior) convention. This function converts to our internal RAS convention by flipping the first two axes.

Usage

```
lps_to_ras(coords)
```

Arguments

coords Numeric matrix (N x 3) or vector of length 3 in LPS

Value

Coordinates in RAS convention

Examples

```
lps <- c(-10, -20, 30)
ras <- lps_to_ras(lps)
```

lta_io	<i>FreeSurfer LTA IO</i>
--------	--------------------------

Description

Read/write FreeSurfer LTA linear transform files (single or array forms).

make_resampling_plan	<i>Build a resampling plan</i>
----------------------	--------------------------------

Description

Build a resampling plan

Usage

```
make_resampling_plan(
  morphism,
  source_grid,
  target_grid,
  interpolation = c("linear", "cubic", "nearest"),
  reuse_count = 1L,
  cache = TRUE
)
```

Arguments

morphism	Morphism or MorphismPath
source_grid	Grid describing the source volume geometry
target_grid	Grid describing the target geometry
interpolation	"linear", "cubic", or "nearest"
reuse_count	Hint for expected reuse; >1 enables caching
cache	Logical; cache the plan by hash

Value

A ResamplingPlan object

mesh_is_sphere	<i>Check whether a surface mesh is approximately spherical</i>
----------------	--

Description

Check whether a surface mesh is approximately spherical

Usage

```
mesh_is_sphere(mesh, tolerance = 1.001)
```

Arguments

mesh	SurfaceMesh object
tolerance	Ratio tolerance for min/max radius comparison

Value

Logical

mesh_set_radius	<i>Set mesh radius for spherical meshes</i>
-----------------	---

Description

Set mesh radius for spherical meshes

Usage

```
mesh_set_radius(mesh, radius = 100)
```

Arguments

mesh	SurfaceMesh object
radius	Target radius

Value

SurfaceMesh object with rescaled coordinates

morphism

Morphism Constructors and Methods

Description

Constructors and methods for morphism classes. Morphisms represent coordinate transformations with pullback semantics.

Morphism-class

Virtual base class for morphisms

Description

A Morphism represents a coordinate transformation between domains. Direction convention: a morphism $A \rightarrow B$ maps coordinates in B to coordinates in A (pullback semantics). Data flows opposite: from A to B .

Usage

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

Arguments

object A Morphism object (for show method)

Slots

id Character identifier
source Character: source domain hash (opaque string)
target Character: target domain hash (opaque string)
kind Character morphism kind (identity, affine3d, warp3d, vol2surf, surf2surf)
params List: forward params/metadata
inverse_params List: inverse params/metadata
method_tag Character: path selection tag (anatomical/functional/etc.)
coverage Optional mask/metadata for valid region
cost Numeric: cost/weight for graph traversal
hash Stable hash of morphism
inverse_type One of "exact", "approximate", "adjoint", "none"
inverse_quality Numeric in range 0 to 1, optional confidence for inverse
inverse_method Description/label for inverse strategy
provenance List: metadata about the transform origin
cache Environment for cached data

morphism_hash	<i>Compute hash for a morphism</i>
---------------	------------------------------------

Description

Creates a stable hash from morphism properties for identity and caching.

Usage

```
morphism_hash(object)
```

Arguments

object	A Morphism object
--------	-------------------

Value

Character hash string

morphism_kind	<i>Get the kind of a morphism</i>
---------------	-----------------------------------

Description

Returns the morphism kind (identity, affine3d, warp3d, vol2surf, surf2surf).

Usage

```
morphism_kind(object)
```

Arguments

object	A Morphism object
--------	-------------------

Value

Character string indicating kind

Examples

```
aff <- Affine3DMorphism("a", "b", diag(4))
morphism_kind(aff) # "affine3d"
```

MorphismPath-class	<i>Morphism path (composed sequence)</i>
--------------------	--

Description

A lightweight wrapper around a list of morphisms representing a composed path. When passed to transform(), applies the full sequence efficiently. Order convention: path = list(f, g) means f first, then g (like g . f).

Slots

- morphisms List of Morphism objects
- source Source domain hash (from first morphism)
- target Target domain hash (from last morphism)

overlap_metrics	<i>Image overlap metrics</i>
-----------------	------------------------------

Description

Compute voxelwise overlap summaries for two images or masks. Inputs may be plain arrays or image objects that can be coerced with as.array(), including neuroim2 volumes.

Usage

```
overlap_metrics(  
  x,  
  y,  
  threshold = 0,  
  labels = NULL,  
  na.rm = TRUE,  
  empty = c("one", "zero", "na"),  
  check_geometry = TRUE,  
  tolerance = sqrt(.Machine$double.eps)  
)
```

Arguments

- x, y Arrays or image objects with matching dimensions.
- threshold Numeric scalar used to form foreground masks when labels is NULL.
- labels Optional vector of label values to compare. When supplied, threshold is ignored and one row is returned per label.
- na.rm Logical; if TRUE, paired missing or non-finite numeric voxels are removed before computing masks.

empty	How to score Dice, Jaccard, overlap coefficient, and volume similarity when both masks are empty. "one" treats two empty masks as perfect agreement, "zero" returns 0, and "na" returns NA_real_.
check_geometry	Logical; if TRUE, compare available voxel-to-world affines after confirming that dimensions match.
tolerance	Numeric tolerance used for affine comparisons.

Details

By default, numeric images are converted to foreground masks with value > threshold, and logical images use TRUE as foreground. Supply labels to compute label-wise overlap instead.

Value

A data frame with voxel counts and overlap metrics. sensitivity is intersection / n_x; precision is intersection / n_y.

Examples

```
x <- array(c(1, 1, 0, 0), dim = c(2, 2, 1))
y <- array(c(1, 0, 1, 0), dim = c(2, 2, 1))
overlap_metrics(x, y)
```

ras_to_lps	<i>Convert RAS coordinates to LPS</i>
------------	---------------------------------------

Description

Converts from our internal RAS convention to ANTs/ITK LPS convention.

Usage

```
ras_to_lps(coords)
```

Arguments

coords Numeric matrix (N x 3) or vector of length 3 in RAS

Value

Coordinates in LPS convention

Examples

```
ras <- c(10, 20, 30)
lps <- ras_to_lps(ras)
```

ras_to_tkras	<i>Convert scanner RAS to FreeSurfer tkRAS</i>
--------------	--

Description

Converts from scanner RAS to FreeSurfer's tkRAS coordinate system.

Usage

```
ras_to_tkras(ras, c_ras)
```

Arguments

ras	Numeric matrix (N x 3) or vector of length 3 in scanner RAS
c_ras	Numeric vector of length 3: the c_ras offset from FreeSurfer

Value

Coordinates in tkRAS

Examples

```
scanner_ras <- c(0.5, -2.3, 1.1)
c_ras <- c(0.5, -2.3, 1.1)
tkras <- ras_to_tkras(scanner_ras, c_ras)
```

read_affine_matrix_txt	<i>Read a 4x4 affine matrix from a text file</i>
------------------------	--

Description

Comment lines (starting with "#") are ignored. If a line contains "affineType:" the value is returned as the detected type.

Usage

```
read_affine_matrix_txt(file, type = NULL)
```

Arguments

file	Path to text file containing affine matrix
type	Optional type hint (overrides auto-detection)

Value

```
list(matrix = 4x4, type = character|NULL)
```

read_image	<i>Simple image I/O using neuroim2</i>
------------	--

Description

Simple image I/O using neuroim2

Usage

read_image(path)

Arguments

path File path

Value

neuroim2 volume (DenseNeuroVol or DenseNeuroVec) or array fallback

read_linear_transform	<i>Read an affine transform file into an Affine3DMorphism</i>
-----------------------	---

Description

Read an affine transform file into an Affine3DMorphism

Usage

```
read_linear_transform(  
  path,  
  format = c("generic", "fsl", "afni", "itk", "lta", "x5"),  
  source = "source",  
  target = "target",  
  source_affine = NULL,  
  target_affine = NULL,  
  source_dim = NULL,  
  target_dim = NULL,  
  oblique_correction = TRUE,  
  invert = FALSE  
)
```

Arguments

path	Path to transform file
format	One of "generic", "fsl", "afni", or "itk"
source	Source domain id
target	Target domain id
source_affine	Source voxel-to-world affine (required for fsl)
target_affine	Target voxel-to-world affine (required for fsl)
source_dim	Optional source image dimensions for FSL handedness-aware conversion
target_dim	Optional target image dimensions for FSL handedness-aware conversion
oblique_correction	Logical; for AFNI format, apply cardinal/deoblique correction
invert	Logical; if TRUE, invert the read matrix before wrapping

Value

Affine3DMorphism

read_linear_transform_array
Read a series of linear transforms

Description

Supports single-file transforms and FSL/MCFLIRT-style indexed files (path.000, path.001, ...).

Usage

```
read_linear_transform_array(
  path,
  format = c("generic", "fsl", "afni", "itk", "lta", "x5"),
  source = "source",
  target = "target",
  ...
)
```

Arguments

path	Base file path
format	One of "generic", "fsl", "afni", or "itk"
source	Source domain id
target	Target domain id
...	Passed to read_linear_transform()

Value

A LinearTransformArray object

read_transform	<i>Read transform from file into a Morphism</i>
----------------	---

Description

Read transform from file into a Morphism

Usage

```
read_transform(
    path,
    type = NULL,
    source = NULL,
    target = NULL,
    apply_affine = TRUE,
    ...
)
```

Arguments

path	Path or Morphism
type	Optional type hint: ants_h5, ants, fsl, fsl_coef, afni, x5, linear, fsl_affine, afni_affine, itk_affine, lta_affine
source	Optional source id
target	Optional target id
apply_affine	Logical; for ANTs H5 files, whether to include embedded affine
...	Additional constructor arguments. FSL warps require source_affine and source_dim. For dense FSL fields, supply target_affine and target_dim if the reference image grid differs from the warp lattice; FNIIRT coefficient files (fsl_coef) always require them. def_type specifies relative FSL offsets or absolute FSL coordinates (inferred from the field when omitted); warp_method controls field interpolation. See Warp3DMorphism.

Value

A Morphism or MorphismPath object

read_x5	<i>Read X5 transform file</i>
---------	-------------------------------

Description

Read X5 transform file

Usage

```
read_x5(path)
```

Arguments

path	Input file path
------	-----------------

Value

List of X5Transform objects

register_loader	<i>Register a named warp loader</i>
-----------------	-------------------------------------

Description

Adds a loader function to the registry. Loaders are used to read warp displacement fields from various file formats.

Usage

```
register_loader(name, loader)

register_warp_loader(name, loader)
```

Arguments

name	Character identifier for the loader
loader	Function(path) -> list(array, dim, world_to_vox)

Value

Invisibly returns the name

Examples

```
# Register a custom loader
my_loader <- function(path) {
  # Load and return list(array, dim, world_to_vox)
}
register_loader("my_format", my_loader)
```

resample

*Resample data through a morphism***Description**

The fundamental resampling operation: transform coordinates via morphism, then evaluate sampler at transformed coordinates.

Usage

```
resample(
  sampler,
  morphism,
  coords,
  modulate = c("none", "jacobian", "sqrt_jacobian")
)
```

Arguments

sampler	Sampler object
morphism	Morphism or MorphismPath
coords	Target coords (n x 3 matrix) or Grid
modulate	Jacobian modulation: "none", "jacobian", "sqrt_jacobian"

Value

Sampled values

Examples

```
# Create volume and sampler
vol <- array(1:27, dim = c(3, 3, 3))
sampler <- volume_sampler(vol, affine = diag(4))

# Create identity morphism
morph <- IdentityMorphism("test")

# Resample at specific coordinates
coords <- matrix(c(0, 0, 0, 1, 1, 1), ncol = 3, byrow = TRUE)
resample(sampler, morph, coords)
```

resample_to	<i>High-level resampling helper</i>
-------------	-------------------------------------

Description

High-level resampling helper

Usage

```
resample_to(moving, target, transform, method = "linear", modulate = "none")
```

Arguments

moving	Source volume (neuroim2 or array)
target	Target volume or Grid
transform	Morphism, MorphismPath, or file path. When a path is a dense FSL field, the moving image (a neuroim2 volume) supplies the source geometry the field needs. FNIRT coefficient files also need the reference geometry, which the target grid does not reliably provide, so read them with <code>read_transform()</code> first.
method	Interpolation method
modulate	Jacobian modulation

Value

neuroim2 volume (matching target geometry)

resample_volume	<i>Resample volume to new geometry</i>
-----------------	--

Description

Convenience wrapper for common volume-to-volume resampling.

Usage

```
resample_volume(data, morphism, target, method = "linear", modulate = "none")
```

Arguments

data	Source volume
morphism	Transform
target	Target geometry (Grid or volume)
method	Interpolation method
modulate	Jacobian modulation

Value

Array with target dimensions

resampling_plan	<i>Resampling plans for repeated volume resampling</i>
-----------------	--

Description

Plan once, apply many times: precompute target voxel centres and flatten the morphism path so repeated resampling of multiple volumes is cheap while keeping the public API small.

reverse_surface_resampling_plan	<i>Construct geometric resampling in the reverse direction</i>
---------------------------------	--

Description

Rebuilds correspondence from the original reference to the original moving mesh, checking both ordered geometry identities. The new source must contain faces for barycentric interpolation. This generally differs from a transpose and does not recover information discarded by downsampling. Masks swap sides.

Usage

reverse_surface_resampling_plan(plan, reference, moving)

Arguments

- | | |
|-----------|---|
| plan | Original SurfaceResamplingPlan with recorded geometry identities. |
| reference | Original reference mesh. |
| moving | Original moving mesh. |

Value

A new SurfaceResamplingPlan with geometries swapped.

sample_volume_on_surface	<i>Sample volume on surface</i>
--------------------------	---------------------------------

Description

Project volumetric data onto surface vertices.

Usage

```
sample_volume_on_surface(  
  data,  
  morphism,  
  surface_coords = NULL,  
  method = "linear"  
)
```

Arguments

data	Source volume
morphism	VolToSurfMorphism
surface_coords	Surface vertex coordinates (optional)
method	Interpolation method

Value

Values at surface vertices

sampld_points	<i>Construct sampled points</i>
---------------	---------------------------------

Description

Construct sampled points

Usage

```
sampld_points(coords)
```

Arguments

coords	Numeric matrix with 3 columns (N x 3)
--------	---------------------------------------

Value

SampldPoints object

SampledPoints-class	<i>Sampled points (irregular spatial reference)</i>
---------------------	---

Description

Lightweight container for point samples in world coordinates.

Usage

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

Arguments

object A SampledPoints object (for the show method).

Slots

coords Numeric matrix (N x 3)

sampler	<i>Sampler Classes and Resampling</i>
---------	---------------------------------------

Description

Samplers encapsulate data + interpolation strategy, enabling elegant composition with morphisms for resampling operations.

Design

A Sampler wraps volumetric or surface data with an interpolation method. The core operation is: given coordinates, return interpolated values.

Resampling through a morphism is then simply:

```
source_coords <- transform(morphism, target_coords)
values <- sampler(source_coords)
```

Sampler-class	<i>Sampler: data + interpolation encapsulation</i>
---------------	--

Description

Sampler: data + interpolation encapsulation

Usage

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

Arguments

object A Sampler object (for show method)

Slots

evaluate Function: (coords) -> values
 dims Integer: data dimensions
 vdim Integer: output dimensionality per point
 bounds List: min/max coordinate bounds
 method Character: interpolation method
 outside Value for out-of-bounds queries

shape_zoom_affine	<i>Build a centered voxel-to-world affine from shape and zooms</i>
-------------------	--

Description

Mirrors the common neuroimaging convention where voxel sizes define the linear part and the image center maps to world origin.

Usage

```
shape_zoom_affine(shape, zooms, x_flip = TRUE, y_flip = FALSE)
```

Arguments

shape Length-N image shape
 zooms Length-N voxel sizes
 x_flip Logical; if TRUE, flip X axis (radiological-on-disk style)
 y_flip Logical; if TRUE, also flip Y axis (with x_flip gives DICOM style)

Value

4x4 affine matrix

source_of	<i>Get source domain of a morphism</i>
-----------	--

Description

Get source domain of a morphism

Usage

```
source_of(object)

source_domain(object)

## S4 method for signature 'Morphism'
source_of(object)

## S4 method for signature 'Morphism'
source_domain(object)

## S4 method for signature 'MorphismPath'
source_of(object)
```

Arguments

object A Morphism object

Value

Source domain hash (character)

surface_mesh	<i>Construct a surface mesh</i>
--------------	---------------------------------

Description

Construct a surface mesh

Usage

```
surface_mesh(vertices, faces = NULL)
```

Arguments

- vertices Numeric matrix with 3 columns (V x 3)
- faces Optional matrix with 3 columns (F x 3), 0- or 1-based indices

Value

SurfaceMesh object

surface_resampling	<i>Surface Resampling Plans</i>
--------------------	---------------------------------

Description

Lightweight surface-to-surface resampling using precomputed sparse triplets. This keeps the core API small while enabling reusable vertex-data transport.

surface_resampling_plan	<i>Build a surface resampling plan</i>
-------------------------	--

Description

Computes reusable interpolation weights mapping vertex data from a moving mesh onto a reference mesh.

Usage

```
surface_resampling_plan(  
  reference,  
  moving,  
  method = c("barycentric", "nearest", "adaptive_bary_area"),  
  spherical = TRUE,  
  radius = 100,  
  source_mask = NULL,  
  target_mask = NULL,  
  outside = NULL,  
  source_areas = NULL,  
  target_areas = NULL,  
  experimental = FALSE  
)
```

Arguments

reference	Reference/output surface (SurfaceMesh or surface-like object)
moving	Moving/input surface (SurfaceMesh or surface-like object)
method	Interpolation method: "barycentric", "nearest", or experimental "adaptive_bary_area"
spherical	Logical; if TRUE, require both meshes to be approximately spherical
radius	Radius used when spherical=TRUE (both meshes are rescaled to this radius)
source_mask	Optional logical or finite numeric input ROI; positive values include vertices. Exclusion is applied before row normalization.
target_mask	Optional output ROI, applied after normalization.
outside	Uncovered-query policy: "error", "missing", or "nearest". Defaults to "error" for spherical and "nearest" for nonspherical meshes.
source_areas, target_areas	Geometry-bound SurfaceVertexAreas objects, required for adaptive area re-sampling, with matching area units.
experimental	Explicitly opt into the adaptive area implementation. Its full-template Workbench parity gate has not passed.

Details

Spherical barycentric resampling uses the closest point on the rescaled triangular mesh, including edges and vertices (ordinary barycentric, without adaptive area correction). It does not use radial ray intersections. For nonspherical meshes, only interior orthogonal projections are considered, with an explicit policy for uncovered queries. The default is nearest-vertex fallback for nonspherical meshes and an error for spherical meshes.

Value

A SurfaceResamplingPlan object

surface_sampler	Create a surface sampler
-----------------	--------------------------

Description

Create a surface sampler

Usage

```
surface_sampler(  
  vertices,  
  data,  
  faces = NULL,  
  method = c("nearest", "barycentric"),  
  projection = c("interior", "closest")  
)
```

Arguments

vertices	Vertex coordinates (V x 3), SampledPoints, or SurfaceMesh
data	Vertex data (length V or V x k)
faces	Face indices (F x 3, optional)
method	"nearest" or "barycentric"
projection	Barycentric projection policy: interior orthogonal projections (default), or the closest point including edges and vertices.

Value

Sampler object

surface_vertex_areas *Bind vertex areas to a registered surface geometry*

Description

Supply either positive area values with explicit provenance, or an anatomical mesh with the same ordered faces and vertex identities as the registration geometry. Mesh areas allocate one third of each Euclidean triangle's area to each corner. Registration spheres are never used implicitly as anatomy.

Usage

```
surface_vertex_areas(
  geometry,
  anatomical = NULL,
  values = NULL,
  units = "mm^2",
  provenance = NULL
)
```

Arguments

geometry	Registration SurfaceMesh to which the areas correspond.
anatomical	Optional anatomical SurfaceMesh with matching topology.
values	Optional finite positive numeric area vector.
units	Area units, for example "mm^2"; both sides must use the same units.
provenance	Nonempty source description, required for supplied values.

Value

A SurfaceVertexAreas object with values, geometry identity, units and provenance.

SurfaceMesh-class	<i>Surface mesh (vertices + faces)</i>
-------------------	--

Description

Lightweight mesh reference for surface operations.

Usage

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

Arguments

object A SurfaceMesh object (for the show method).

Slots

coords Numeric matrix (V x 3) of vertices
 faces Integer matrix (F x 3) of 0-based triangle indices

SurfToSurfMorphism	<i>Create a surface-to-surface morphism</i>
--------------------	---

Description

Maps between surface meshes via spherical registration.

Usage

```
SurfToSurfMorphism(
  source,
  target,
  method = c("sphere", "area", "sulc"),
  source_sphere = "",
  target_sphere = "",
  mapping = c("nearest", "barycentric"),
  source_vertices = NULL,
  target_vertices = NULL,
  faces = NULL,
  cost = 1,
  method_tag = "anatomical"
)
```

Arguments

source	Source surface domain hash
target	Target surface domain hash
method	Registration method: "sphere", "area", "sulc"
source_sphere	Path to source sphere surface
target_sphere	Path to target sphere surface
mapping	How to map target points onto the source mesh. "nearest" maps by nearest target vertex (index correspondence); "barycentric" maps using barycentric weights on target faces.
source_vertices	Optional V x 3 numeric matrix of source mesh vertices. Required for mapping="nearest" and mapping="barycentric".
target_vertices	Optional V x 3 numeric matrix of target mesh vertices. Required for mapping="nearest" and mapping="barycentric".
faces	Optional F x 3 integer matrix of triangle vertex indices (0- or 1-based). Required for mapping="barycentric".
cost	Path cost (default 1.0)
method_tag	Method tag (default "anatomical")

Value

SurfToSurfMorphism object

SurfToSurfMorphism-class

Surface-to-surface morphism

Description

Maps between surface meshes (e.g., native to fsaverage via spherical registration).

Slots

method Registration method ("sphere", "area", "sulc")
source_sphere Path to source sphere surface
target_sphere Path to target sphere surface

target_of	<i>Get target domain of a morphism</i>
-----------	--

Description

Get target domain of a morphism

Usage

```
target_of(object)

target_domain(object)

## S4 method for signature 'Morphism'
target_of(object)

## S4 method for signature 'Morphism'
target_domain(object)

## S4 method for signature 'MorphismPath'
target_of(object)
```

Arguments

object A Morphism object

Value

Target domain hash (character)

tkras_to_ras	<i>Convert FreeSurfer tkRAS to scanner RAS</i>
--------------	--

Description

FreeSurfer’s tkrregister uses a shifted coordinate system (tkRAS) that differs from scanner RAS by the c_ras offset stored in the surface header.

Usage

```
tkras_to_ras(tkras, c_ras)
```

Arguments

tkras Numeric matrix (N x 3) or vector of length 3 in tkRAS
c_ras Numeric vector of length 3: the c_ras offset from FreeSurfer

Value

Coordinates in scanner RAS

Examples

```
tkras <- c(0, 0, 0)
c_ras <- c(0.5, -2.3, 1.1)
scanner_ras <- tkras_to_ras(tkras, c_ras)
```

transform	<i>Transform coordinates through a morphism (pullback)</i>
-----------	--

Description

Applies the coordinate pullback: given coordinates in the target domain, returns corresponding coordinates in the source domain.

Usage

```
transform(morphism, coords)

## S4 method for signature 'IdentityMorphism'
transform(morphism, coords)

## S4 method for signature 'Affine3DMorphism'
transform(morphism, coords)

## S4 method for signature 'Warp3DMorphism'
transform(morphism, coords)

## S4 method for signature 'VolToSurfMorphism'
transform(morphism, coords)

## S4 method for signature 'SurfToSurfMorphism'
transform(morphism, coords)

## S4 method for signature 'MorphismPath'
transform(morphism, coords)
```

Arguments

morphism	A Morphism object or MorphismPath
coords	Matrix of coordinates (N x 3) in target domain

Value

Matrix of coordinates (N x 3) in source domain

Examples

```
# Create an affine morphism
aff <- Affine3DMorphism("source_hash", "target_hash", diag(4))
coords <- matrix(c(10, 20, 30), ncol = 3)
transform(aff, coords)
```

transform_coords	<i>Transform coordinates (compatibility alias)</i>
------------------	--

Description

Alias for transform() to maintain neurofunctor compatibility.

Usage

```
transform_coords(object, coords)

## S4 method for signature 'Morphism'
transform_coords(object, coords)

## S4 method for signature 'MorphismPath'
transform_coords(object, coords)
```

Arguments

- object A Morphism object
- coords Matrix of coordinates (N x 3)

Value

Transformed coordinates

transform_io	<i>Transform IO Helpers</i>
--------------	-----------------------------

Description

Lightweight IO helpers for linear transform backends, including format factories, typed IO conditions, and array-style linear transform IO.

transform_path	<i>Transform Path Application</i>
----------------	-----------------------------------

Description

Apply a morphism path to coordinates using optimized warp chain.

Given a path of morphisms from source to target and coordinates in the target domain, returns the corresponding coordinates in the source domain.

Usage

```
transform_path(path, target_coords)
```

Arguments

path	List of Morphism objects ordered source -> target
target_coords	Numeric matrix (N x 3) of target world coords

Value

Numeric matrix (N x 3) of source world coords

Pullback Direction

Given a path [f: A->B, g: B->C], transform_path applies the pullback:

```
coords_in_C --(g pullback)--> coords_in_B --(f pullback)--> coords_in_A
```

The path is applied from last to first (g then f).

Examples

```
# Create affine morphisms
aff1 <- Affine3DMorphism("native", "mni", diag(4))
aff2 <- Affine3DMorphism("mni", "template", diag(4))
path <- list(aff1, aff2)

# Transform template coords to native space
template_coords <- matrix(c(0, 0, 0, 10, 20, 30), ncol = 3, byrow = TRUE)
native_coords <- transform_path(path, template_coords)
```

TransformFileError	<i>Construct a transform file-format error condition</i>
--------------------	--

Description

Construct a transform file-format error condition

Usage

TransformFileError(message)

Arguments

message	Error message
---------	---------------

Value

Condition object of class TransformFileError

TransformIOError	<i>Construct a transform IO error condition</i>
------------------	---

Description

Construct a transform IO error condition

Usage

TransformIOError(message)

Arguments

message	Error message
---------	---------------

Value

Condition object of class TransformIOError

validate_path	<i>Validate a morphism path</i>
---------------	---------------------------------

Description

Checks that a path is composable (each morphism's target matches the next's source). Raises an error if the path is invalid.

Usage

```
validate_path(path)
```

Arguments

path	List of Morphism objects
------	--------------------------

Value

Invisibly returns the path if valid

Examples

```
aff1 <- Affine3DMorphism("a", "b", diag(4))
aff2 <- Affine3DMorphism("b", "c", diag(4))
validate_path(list(aff1, aff2)) # OK
```

validate_surface_mesh	<i>Validate a surface mesh</i>
-----------------------	--------------------------------

Description

Reports invalid faces and topology without repairing them. Spherical admission additionally requires origin-based radius spread within tolerance, a closed connected orientable manifold of sphere topology, unambiguous radial face orientations and radial mapping degree one. Input face winding is free. Generic open meshes do not require closed topology or spherical geometry.

Usage

```
validate_surface_mesh(
  mesh,
  spherical = TRUE,
  error = FALSE,
  radius_tolerance = 1.001
)
```

Arguments

mesh	SurfaceMesh or surface-like object with faces.
spherical	Require an embedded spherical triangulation.
error	Stop when validation fails instead of returning diagnostics.
radius_tolerance	Maximum ratio of vertex radii about the origin.

Value

Diagnostic list with valid, issues (one-based offending indices), topology counts, radius range, radial degree and numerical margins.

VolToSurfMorphism	<i>Create a volume-to-surface morphism</i>
-------------------	--

Description

Maps from volume domain to surface domain using various sampling strategies.

Usage

```
VolToSurfMorphism(
  source,
  target,
  method = c("trilinear", "ribbon", "mid_thickness", "nearest"),
  ribbon_inner = "",
  ribbon_outer = "",
  ribbon_inner_coords = NULL,
  ribbon_outer_coords = NULL,
  n_ribbon_samples = 6L,
  mid_coords = NULL,
  cost = 2,
  method_tag = "anatomical"
)
```

Arguments

source	Volume domain hash
target	Surface domain hash
method	Sampling method: "trilinear", "ribbon", "mid_thickness", "nearest"
ribbon_inner	Path to inner (white) surface for ribbon sampling
ribbon_outer	Path to outer (pial) surface for ribbon sampling
ribbon_inner_coords	Pre-loaded inner surface coordinates (optional)

ribbon_outer_coords	Pre-loaded outer surface coordinates (optional)
n_ribbon_samples	Number of samples along ribbon normal
mid_coords	Pre-computed midpoint coordinates (optional)
cost	Path cost (default 2.0)
method_tag	Method tag (default "anatomical")

Value

VolToSurfMorphism object

VolToSurfMorphism-class
Volume-to-surface morphism

Description

Maps volume coordinates to surface vertices. Supports different sampling strategies (ribbon, trilinear point sampling).

Slots

method Sampling method ("ribbon", "trilinear", "nearest", "mid_thickness")
 ribbon_inner Path to inner (white) surface for ribbon sampling
 ribbon_outer Path to outer (pial) surface for ribbon sampling
 n_ribbon_samples Number of samples along ribbon normal

volume_sampler *Create a volume sampler*

Description

Create a volume sampler

Usage

```
volume_sampler(
  data,
  affine = NULL,
  method = c("linear", "nearest", "cubic"),
  outside = NA_real_
)
```

Arguments

data	Numeric array (3D/4D) or volume object
affine	4x4 voxel-to-world (extracted if NULL)
method	Interpolation: "linear", "nearest", "cubic"
outside	Value for out-of-bounds

Value

Sampler object

Examples

```
vol <- array(rnorm(27), dim = c(3, 3, 3))
sampler <- volume_sampler(vol, affine = diag(4))
coords <- matrix(c(1, 1, 1), ncol = 3)
sampler@evaluate(coords)
```

warp_from_field

Create an in-memory dense-field warp morphism

Description

Builds a Warp3DMorphism directly from a 4D field array and a grid, without requiring an on-disk warp file.

Usage

```
warp_from_field(
  source,
  target,
  field,
  grid = NULL,
  representation = c("displacements", "deformations"),
  warp_method = c("linear", "cubic"),
  cost = 1.5,
  method_tag = "anatomical",
  id = NULL
)
```

Arguments

source	Source domain id
target	Target domain id
field	Numeric array with dims c(X, Y, Z, 3)
grid	Grid describing voxel geometry. If NULL, uses attr(field, "grid") when available (e.g., from deformation_field()).

representation	Either "displacements" (relative offsets) or "deformations" (absolute source coordinates)
warp_method	Interpolation method for warp lookup
cost	Path cost
method_tag	Method tag
id	Optional stable id for the in-memory warp payload

Value

Warp3DMorphism

warp_loader	<i>Warp Field Loader Registry</i>
-------------	-----------------------------------

Description

Pluggable loader system for warp displacement fields. Provides a registry for different file formats (NIfTI via neuroim2, AFNI, FSL, ANTs H5) with morphism-level caching.

Loader Function Contract

A loader function must accept a file path and return a list with:

- array: Numeric vector of displacement values (flattened 3 x X x Y x Z)
- dim: Integer vector c(X, Y, Z)
- world_to_vox: 4x4 matrix mapping world coords to voxel indices

Warp3DMorphism	<i>Create a 3D warp field morphism</i>
----------------	--

Description

Represents a nonlinear warp field transformation.

Usage

```
Warp3DMorphism(  
  source,  
  target,  
  warp_path,  
  warp_type = c("ants", "ants_h5", "fsl", "fsl_coef", "afni", "freesurfer", "dense"),  
  inverse_path = "",  
  def_type = NULL,  
  warp_method = c("linear", "cubic"),
```

```

cost = 1.5,
method_tag = "anatomical",
source_affine = NULL,
source_dim = NULL,
target_affine = NULL,
target_dim = NULL
)

```

Arguments

source	Source domain hash
target	Target domain hash
warp_path	Path to displacement field file
warp_type	One of "ants", "ants_h5", "fsl", "fsl_coef", "afni", "freesurfer", "dense"
inverse_path	Path to inverse warp (optional)
def_type	Deformation type: "relative" (displacement) or "absolute" (coordinates)
warp_method	Interpolation method for warp field lookup: "linear" or "cubic"
cost	Path cost (default 1.5)
method_tag	Method tag (default "anatomical")
source_affine, source_dim	Source image voxel-to-RAS affine and three dimensions. Required for FSL warps (warp_type = "fsl" or "fsl_coef"), and construction fails without them: FSL field values are source scaled-voxel coordinates, which cannot be recovered from the warp.
target_affine, target_dim	Reference image voxel-to-RAS affine and three dimensions. For dense FSL fields they default to the warp lattice; supply both when the field lattice differs from the reference image grid. Required for FNIRT coefficient files ("fsl_coef"), which store the reference dimensions and voxel size but not its orientation or origin.

Value

Warp3DMorphism object

Examples

```
warp <- Warp3DMorphism("native", "mni", "path/to/warp.nii.gz")
```

Warp3DMorphism-class *3D warp field morphism*

Description

Represents a nonlinear warp field transformation (e.g., ANTs SyN, FSL FNIRT). Stores metadata; the actual warp field is loaded on demand.

Slots

warp_path Path to the warp field file

warp_type Type of warp ("ants", "ants_h5", "fsl", "fsl_coef", "afni", "freesurfer", "dense")

inverse_path Path to inverse warp (if available)

write_affine_matrix_txt

Write a 4x4 affine matrix to a text file

Description

Write a 4x4 affine matrix to a text file

Usage

```
write_affine_matrix_txt(matrix, file, type = NULL, comment = TRUE)
```

Arguments

matrix 4x4 numeric affine matrix

file Output file path

type Optional type label to write as comment

comment Logical; if TRUE, write type as comment header

Value

Invisibly returns the file path

write_image	<i>Write image using neuroim2</i>
-------------	-----------------------------------

Description

Write image using neuroim2

Usage

write_image(x, path)

Arguments

x	neuroim2 volume or array
path	Output path

write_linear_transform	<i>Write an affine transform to disk</i>
------------------------	--

Description

Write an affine transform to disk

Usage

```
write_linear_transform(  
  x,  
  path,  
  format = c("generic", "fsl", "afni", "itk", "lta", "x5"),  
  source_affine = NULL,  
  target_affine = NULL,  
  source_dim = NULL,  
  target_dim = NULL,  
  oblique_correction = TRUE,  
  invert = FALSE  
)
```

Arguments

x	Affine3DMorphism or 4x4 matrix (internal pullback convention)
path	Output file path
format	One of "generic", "fsl", "afni", or "itk"
source_affine	Source voxel-to-world affine (required for fsl)

target_affine	Target voxel-to-world affine (required for fsl)
source_dim	Optional source image dimensions for FSL handedness-aware conversion
target_dim	Optional target image dimensions for FSL handedness-aware conversion
oblique_correction	Logical; for AFNI format, apply cardinal/deoblique correction
invert	Logical; if TRUE, invert before writing

Value

Invisibly returns path

```
write_linear_transform_array
```

Write a series of linear transforms

Description

Write a series of linear transforms

Usage

```
write_linear_transform_array(
  x,
  path,
  format = c("generic", "fsl", "afni", "itk", "lta", "x5"),
  ...
)
```

Arguments

x	A LinearTransformArray, list of affine morphisms/matrices, or single transform
path	Base output path
format	One of "generic", "fsl", "afni", or "itk"
...	Passed to write_linear_transform()

Value

Invisibly returns written path(s)

write_transform	<i>Write a transform to disk</i>
-----------------	----------------------------------

Description

Convenience wrapper around `write_linear_transform()` for affine morphisms and raw 4x4 matrices. For `Warp3DMorphism`, writes an LPS NIfTI vector field (ANTs-compatible when the output contains displacements).

Usage

```
write_transform(x, path, type = NULL, ...)
```

Arguments

x	Affine3DMorphism, Warp3DMorphism, or 4x4 numeric matrix
path	Output file path
type	Optional output type: - linear: generic, fsl, afni, itk - warp: nifti, displacements, deformations
...	Passed to <code>write_linear_transform()</code> or <code>write_warp_field()</code>

Value

Invisibly returns path

write_warp_field	<i>Write a warp morphism as a NIfTI vector field</i>
------------------	--

Description

Writes a 5D vector image (X, Y, Z, 1, 3), with intent code 1007 and LPS components. Displacement output is compatible with ANTs. Absolute deformation output contains source coordinates and must be explicitly read as such; ANTs expects displacements. Requires `RNifti`.

Usage

```
write_warp_field(
  x,
  path,
  representation = c("auto", "displacements", "deformations")
)
```

Arguments

x	Warp3DMorphism
path	Output NIfTI path
representation	Output representation: "displacements", "deformations", or "auto" (use morphism native representation)

Value

Invisibly returns path

write_x5	<i>Write X5 transform file</i>
----------	--------------------------------

Description

Write X5 transform file

Usage

write_x5(path, x5_list)

Arguments

path	Output file path
x5_list	List of X5Transform objects

Value

Invisibly returns path

x5_domain	<i>Create an X5 domain descriptor</i>
-----------	---------------------------------------

Description

Create an X5 domain descriptor

Usage

x5_domain(grid, size, mapping = NULL, coordinates = NULL)

Arguments

grid	Logical
size	Integer vector
mapping	Optional matrix
coordinates	Optional coordinate kind string

Value

An X5Domain object

x5_io	<i>X5 Transform IO</i>
-------	------------------------

Description

Read and write X5 transform files for linear and nonlinear transforms.

x5_transform	<i>Create an X5 transform descriptor</i>
--------------	--

Description

Create an X5 transform descriptor

Usage

```
x5_transform(  
  type,  
  transform,  
  subtype = NULL,  
  representation = NULL,  
  metadata = NULL,  
  dimension_kinds = NULL,  
  domain = NULL,  
  inverse = NULL,  
  jacobian = NULL,  
  additional_parameters = NULL,  
  array_length = 1L  
)
```

Arguments

<code>type</code>	"linear", "nonlinear", or "composite"
<code>transform</code>	Numeric array of transform parameters
<code>subtype</code>	Optional subtype
<code>representation</code>	Optional representation label
<code>metadata</code>	Optional named list or JSON string
<code>dimension_kinds</code>	Optional character vector
<code>domain</code>	Optional X5Domain
<code>inverse</code>	Optional inverse array
<code>jacobian</code>	Optional Jacobian array
<code>additional_parameters</code>	Optional array
<code>array_length</code>	Integer array length

Value

An X5Transform object

Index

[, JacobianField, numeric, ANY, ANY-method
(JacobianField-class), [41](#)
[, JacobianField, numeric-method
(JacobianField-class), [41](#)

adjoint, [5](#)
adjoint, Affine3DMorphism-method
(adjoint), [5](#)
adjoint, IdentityMorphism-method
(adjoint), [5](#)
adjoint, Morphism-method (adjoint), [5](#)
adjoint, SurfToSurfMorphism-method
(adjoint), [5](#)
adjoint, VolToSurfMorphism-method
(adjoint), [5](#)
adjoint, Warp3DMorphism-method
(adjoint), [5](#)
Affine3DMorphism, [6](#)
Affine3DMorphism-class, [7](#)
affine_from_components, [6](#)
affine_utils, [6](#)
afni_aff12_to_ras, [7](#)
afni_is_oblique, [8](#)
afni_read_aff12, [9](#), [10](#)
afni_read_aff12_array, [9](#), [9](#)
afni_warp, [10](#)
afni_write_aff12, [10](#)
all_class, [11](#)
all_generic, [12](#)
ants_h5_morphism, [12](#)
apply_affine, [13](#)
apply_resampling_plan, [13](#)
apply_surface_adjoint, [14](#)
apply_surface_resampling, [15](#)
as_morphism_path, [16](#)

backproject_surface_to_volume, [16](#)
build_affine_matrix, [6](#), [17](#)

compose, [18](#)
compose, Morphism, Morphism-method
(compose), [18](#)
compose, Morphism, MorphismPath-method
(compose), [18](#)
compose, MorphismPath, Morphism-method
(compose), [18](#)
compose, MorphismPath, MorphismPath-method
(compose), [18](#)
compose_affines, [19](#)
compute_hash, [19](#)
convert_affine_convention, [20](#)
coordinates, [21](#)
cpp_path_apply_affine_warp_affine, [21](#)
cpp_path_apply_steps, [22](#)

decompose_affine_matrix, [22](#)
deformation_field, [23](#)
det, JacobianField-method
(JacobianField-class), [41](#)
detect_fnirt_def_type, [23](#)
detect_transform_type, [24](#)
dice_coefficient, [25](#)

extract_affine, [26](#)
extract_affine, ANY-method
(extract_affine), [26](#)
extract_affine, array-method
(extract_affine), [26](#)
extract_affine, DenseNeuroVec-method
(extract_affine), [26](#)
extract_affine, DenseNeuroVol-method
(extract_affine), [26](#)
extract_affine, matrix-method
(extract_affine), [26](#)

fsl_flirt_to_internal_affine, [27](#)
fsl_ingest, [28](#)

get_linear_factory, [28](#)
get_loader, [28](#)

- get_warp_loader (get_loader), 28
- Grid-class, 29
- grid_coords, 29
- grid_from_data, 30
- grid_of, 30
- grid_spec, 31

- has_adjoint, 31

- IdentityMorphism, 32
- IdentityMorphism-class, 32
- invert, 33
- invert, Affine3DMorphism-method (invert), 33
- invert, IdentityMorphism-method (invert), 33
- invert, Morphism-method (invert), 33
- invert, Warp3DMorphism-method (invert), 33
- invert_affine, 33
- is_affine_matrix, 34
- is_affine_morphism, 34
- is_invertible, 35
- is_linear_morphism, 35
- is_valid_path, 36
- is_warp_morphism, 36

- jaccard_index, 37
- jacobian, 38
- jacobian, Affine3DMorphism, ANY-method (jacobian), 38
- jacobian, DeformationField, missing-method (jacobian), 38
- jacobian, IdentityMorphism, ANY-method (jacobian), 38
- jacobian, MorphismPath, ANY-method (jacobian), 38
- jacobian, SurfToSurfMorphism, ANY-method (jacobian), 38
- jacobian, VolToSurfMorphism, ANY-method (jacobian), 38
- jacobian, Warp3DMorphism, ANY-method (jacobian), 38
- jacobian_det, 39
- jacobian_det, Affine3DMorphism-method (jacobian_det), 39
- jacobian_det, IdentityMorphism-method (jacobian_det), 39
- jacobian_det, MorphismPath-method (jacobian_det), 39
- jacobian_det, Warp3DMorphism-method (jacobian_det), 39
- jacobian_det_field, 40
- JacobianField-class, 41

- length, JacobianField-method (JacobianField-class), 41
- list_loaders, 42
- lps_to_ras, 42
- lta_io, 43

- make_resampling_plan, 43
- mesh_is_sphere, 44
- mesh_set_radius, 44
- morphism, 45
- Morphism-class, 45
- morphism_hash, 46
- morphism_kind, 46
- MorphismPath-class, 47

- overlap_metrics, 47

- ras_to_lps, 48
- ras_to_tkras, 49
- read_affine_matrix_txt, 49
- read_image, 50
- read_linear_transform, 50
- read_linear_transform_array, 51
- read_transform, 52
- read_x5, 53
- register_loader, 53
- register_warp_loader (register_loader), 53
- resample, 54
- resample_to, 55
- resample_volume, 55
- resampling_plan, 56
- reverse_surface_resampling_plan, 56

- sample_volume_on_surface, 57
- sampled_points, 57
- SampledPoints-class, 58
- sampler, 58
- Sampler-class, 59
- shape_zoom_affine, 59
- show, Grid-method (Grid-class), 29
- show, JacobianField-method (JacobianField-class), 41

- show, Morphism-method (Morphism-class), [45](#)
- show, SampledPoints-method (SampledPoints-class), [58](#)
- show, Sampler-method (Sampler-class), [59](#)
- show, SurfaceMesh-method (SurfaceMesh-class), [64](#)
- solve, JacobianField, missing-method (JacobianField-class), [41](#)
- source_domain (source_of), [60](#)
- source_domain, Morphism-method (source_of), [60](#)
- source_of, [60](#)
- source_of, Morphism-method (source_of), [60](#)
- source_of, MorphismPath-method (source_of), [60](#)
- surface_mesh, [60](#)
- surface_resampling, [61](#)
- surface_resampling_plan, [61](#)
- surface_sampler, [62](#)
- surface_vertex_areas, [63](#)
- SurfaceMesh-class, [64](#)
- SurfToSurfMorphism, [64](#)
- SurfToSurfMorphism-class, [65](#)
- target_domain (target_of), [66](#)
- target_domain, Morphism-method (target_of), [66](#)
- target_of, [66](#)
- target_of, Morphism-method (target_of), [66](#)
- target_of, MorphismPath-method (target_of), [66](#)
- tkras_to_ras, [66](#)
- transform, [67](#)
- transform, Affine3DMorphism-method (transform), [67](#)
- transform, IdentityMorphism-method (transform), [67](#)
- transform, MorphismPath-method (transform), [67](#)
- transform, SurfToSurfMorphism-method (transform), [67](#)
- transform, VolToSurfMorphism-method (transform), [67](#)
- transform, Warp3DMorphism-method (transform), [67](#)
- transform_coords, [68](#)
- transform_coords, Morphism-method (transform_coords), [68](#)
- transform_coords, MorphismPath-method (transform_coords), [68](#)
- transform_io, [68](#)
- transform_path, [69](#)
- TransformFileError, [70](#)
- TransformIOError, [70](#)
- validate_path, [71](#)
- validate_surface_mesh, [71](#)
- VolToSurfMorphism, [72](#)
- VolToSurfMorphism-class, [73](#)
- volume_sampler, [73](#)
- Warp3DMorphism, [75](#)
- Warp3DMorphism-class, [77](#)
- warp_from_field, [74](#)
- warp_loader, [75](#)
- write_affine_matrix_txt, [77](#)
- write_image, [78](#)
- write_linear_transform, [78](#)
- write_linear_transform_array, [79](#)
- write_transform, [80](#)
- write_warp_field, [80](#)
- write_x5, [81](#)
- x5_domain, [81](#)
- x5_io, [82](#)
- x5_transform, [82](#)