Package: neurotransform (via r-universe)

May 27, 2026

Title Geometric Transforms for Neuroimaging Data

Version 0.1.0

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, knitr, rmarkdown

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_warp.R' 'fsl_ingest.R' 'affine_utils.R' 'deformation_field.R' 'resampling_plan.R' 'surface_resampling.R'

Config/pak/sysreqs make libicu-dev

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

Date/Publication 2026-04-27 16:44:07 UTC

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

RemoteRef HEAD

RemoteSha 450fc1b9bd332919514fc243dfc6e88efce1ed27

Contents

`adjoint` *Get adjoint (generalized inverse)*

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

source	Source domain identifier
target	Target domain identifier
...	Arguments passed to build_affine_matrix

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 RAI affine to RAS*

Description

Converts an AFNI RAI affine matrix to our internal RAS convention by flipping the Z axis. Optionally applies AFNI-style deobliquing compensation when source/target image affines are provided.

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

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 *Read AFNI .aff12.1D affine matrix*

Description

Reads a 3x4 affine matrix in RAI coordinates and returns a 4x4 matrix.

Usage

```
afni_read_aff12(path)
```

Arguments

path	Path to .aff12.1D file

Value

4x4 affine matrix (in RAI)

Examples

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

## End(Not run)
```

`afni_warp` *AFNI Warp and Affine Handlers*

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

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

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

`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 -> 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) or MorphismPath

Pullback Ordering

When `apply_affine=TRUE`, returns a MorphismPath ordered as `[warp, affine]`. This ordering is critical for correct pullback (resampling) semantics:

- MorphismPath applies transforms right-to-left for pullback
- `[warp, affine]` means: `affine_pullback(warp_pullback(coords))`

- Target coords -> warp lookup -> affine transform -> source coords

The ANTs forward transform is "affine then warp", so pullback (inverse) is "inverse_warp then inverse_affine" - hence the `[warp, affine]` order.

`apply_affine` *Apply affine transformation to coordinates*

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` *Apply a resampling plan to a volume*

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_resampling

Apply a surface resampling plan to vertex data

Description

Apply a surface resampling plan to vertex data

Usage

```
apply_surface_resampling(
  plan,
  x,
  inverse = FALSE,
  normalize = c("element", "sum", "none")
)
```

Arguments

plan	A SurfaceResamplingPlan
x	Vertex data (length == n_moving) or matrix (n_moving x k)
inverse	Logical; apply inverse direction (reference -> moving) by transposing weights
normalize	One of "element", "sum", or "none"

Value

Resampled vector or matrix

as_morphism_path *Coerce list/paths to MorphismPath*

Description

Coerce list/paths to MorphismPath

Usage

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

Arguments

...	Morphism objects or file paths to transforms

Value

MorphismPath object

backproject_surface_to_volume *Backproject surface values into a volume (adjoint of VolToSurf sampling)*

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 *Build a 4x4 affine matrix from components*

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 *Compose two morphisms*

Description

Returns the composition g . 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 *Compose two affine transformations*

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 *Compute a stable hash for objects*

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 *Coordinate Convention Utilities*

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 *Deformation fields and Jacobian maps*

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 *Detect FNIRT deformation type (relative vs absolute)*

Description

Heuristically determines whether an FNIRT warp stores relative displacements or absolute coordinates.

Usage

```
detect_fnirt_def_type(warp_path, sample_n = 200, threshold_mm = 5)
```

Arguments

warp_path	Path to FNIRT warp file
sample_n	Number of voxels to sample for heuristic
threshold_mm	Threshold for displacement magnitude heuristic

Value

"relative" or "absolute"

Examples

```
## Not run:
def_type <- detect_fnirt_def_type("warp.nii.gz")

## End(Not run)
```

`detect_transform_type` *Detect transform type from a file path*

Description

Detect transform type from a file path

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` *Dice coefficient for two images*

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 labels is NULL.
label	Optional single label value. If supplied, Dice is computed for that label instead of by thresholding.
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.

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 *Extract affine matrix from data*

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

flirt_mat	4x4 FLIRT matrix
source_affine	4x4 voxel-to-world for source image
ref_affine	4x4 voxel-to-world for reference image
source_dim	Optional source image dimensions (needed for handedness swap)
ref_dim	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` *FSL Coordinate Convention Handlers*

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` *Get linear transform IO backend factory*

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 *Retrieve a warp loader by name*

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 *Grid: regular coordinate grid specification*

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 *Get world coordinates for a grid*

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 *Get grid specification from volume data*

Description

Get grid specification from volume data

Usage

```
grid_from_data(data)
```

Arguments

data	Volume with geometry

Value

Grid object

grid_of *Extract Grid from volume or grid-like input*

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 *Invert a morphism*

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 *Invert an affine transformation*

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 *Check if an object is a numeric 4x4 affine matrix*

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 *Check if an object is an Affine3DMorphism*

Description

Check if an object is an Affine3DMorphism

Usage

```
is_affine_morphism(x)
```

Arguments

x	Object to test

Value

Logical

is_invertible *Check if morphism is exactly invertible*

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 *Check if morphism is linear (affine or identity)*

Description

Check if morphism is linear (affine or identity)

Usage

```
is_linear_morphism(object)
```

Arguments

object	A Morphism object

Value

Logical

is_valid_path *Check if a morphism path is valid*

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 *Check if morphism is a warp field*

Description

Check if morphism is a warp field

Usage

```
is_warp_morphism(object)
```

Arguments

object	A Morphism object

Value

Logical

jaccard_index *Jaccard index for two images*

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

x, y	Arrays or image objects with matching dimensions.
threshold	Numeric scalar used to form foreground masks when labels is NULL.
label	Optional single label value. If supplied, Dice is computed for that label instead of by thresholding.
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.

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 -> source), the pullback Jacobian is d(source)/d(target), evaluated at target coordinates.

Modes

- "pullback": d(source)/d(target) - natural for resampling
- "pushforward": d(target)/d(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 *Extract Jacobian determinant volume from a morphism on a grid*

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 *Jacobian field: collection of 3x3 Jacobian matrices*

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 *List all registered loaders*

Description

List all registered loaders

Usage

```
list_loaders()
```

Value

Character vector of registered loader names

Examples

```
list_loaders()
```

---

lps_to_ras *Convert LPS coordinates to RAS*

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 *FreeSurfer LTA IO*

Description

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

make_resampling_plan *Build a resampling plan*

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 *Check whether a surface mesh is approximately spherical*

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 *Set mesh radius for spherical meshes*

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 -> 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 *Compute hash for a morphism*

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 *Get the kind of a morphism*

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 *Morphism path (composed sequence)*

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 *Image overlap metrics*

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 *Convert RAS coordinates to LPS*

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 *Convert scanner RAS to FreeSurfer tkRAS*

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 *Read a 4x4 affine matrix from a text file*

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 *Simple image I/O using neuroim2*

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 *Read an affine transform file into an Affine3DMorphism*

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 *Read transform from file into a Morphism*

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
...	Passed to morphism constructors

Value

A Morphism or MorphismPath object

read_x5 *Read X5 transform file*

Description

Read X5 transform file

Usage

```
read_x5(path)
```

Arguments

path	Input file path

Value

List of X5Transform objects

---

register_loader *Register a named warp loader*

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 *High-level resampling helper*

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
method	Interpolation method
modulate	Jacobian modulation

Value

neuroim2 volume (matching target geometry)

resample_volume *Resample volume to new geometry*

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 *Resampling plans for repeated volume resampling*

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.

sample_volume_on_surface *Sample volume on surface*

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

sampled_points *Construct sampled points*

Description

Construct sampled points

Usage

```
sampled_points(coords)
```

Arguments

coords	Numeric matrix with 3 columns (N x 3)

Value

SampledPoints object

SampledPoints-class *Sampled points (irregular spatial reference)*

Description

Lightweight container for point samples in world coordinates.

Usage

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

Slots

coords Numeric matrix (N x 3)

sampler *Sampler Classes and Resampling*

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 *Sampler: data + interpolation encapsulation*

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 *Build a centered voxel-to-world affine from shape and zooms*

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 *Get source domain of a morphism*

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 *Construct a surface mesh*

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 *Surface Resampling Plans*

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 *Build a surface resampling plan*

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"),
  spherical = TRUE,
  radius = 100
)
```

Arguments

reference	Reference/output surface (SurfaceMesh or surface-like object)
moving	Moving/input surface (SurfaceMesh or surface-like object)
method	Interpolation method: "barycentric" or "nearest"
spherical	Logical; if TRUE, require both meshes to be approximately spherical
radius	Radius used when spherical=TRUE (both meshes are rescaled to this radius)

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")
)
```

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"

Value

Sampler object

SurfaceMesh-class *Surface mesh (vertices + faces)*

Description

Lightweight mesh reference for surface operations.

Usage

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

Slots

coords Numeric matrix (V x 3) of vertices

faces Integer matrix (F x 3) of 0-based triangle indices

SurfToSurfMorphism *Create a surface-to-surface morphism*

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 *Get target domain of a morphism*

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 *Convert FreeSurfer tkRAS to scanner RAS*

Description

FreeSurfer's tkregister 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` *Transform coordinates through a morphism (pullback)*

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` *Transform coordinates (compatibility alias)*

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` *Transform IO Helpers*

Description

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

`transform_path` *Transform Path Application*

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 *Construct a transform file-format error condition*

Description

Construct a transform file-format error condition

Usage

```
TransformFileError(message)
```

Arguments

message	Error message

Value

Condition object of class `TransformFileError`

TransformIOError *Construct a transform IO error condition*

Description

Construct a transform IO error condition

Usage

```
TransformIOError(message)
```

Arguments

message	Error message

Value

Condition object of class `TransformIOError`

validate_path *Validate a morphism path*

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
```

VolToSurfMorphism *Create a volume-to-surface morphism*

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
domain	Optional domain hash

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 *Warp Field Loader Registry*

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` *Create a 3D warp field morphism*

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"
)
```

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")

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 *Write image using neuroim2*

Description

Write image using neuroim2

Usage

```
write_image(x, path)
```

Arguments

x	neuroim2 volume or array
path	Output path

write_linear_transform *Write an affine transform to disk*

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 *Write a transform to disk*

Description

Convenience wrapper around `write_linear_transform()` for affine morphisms and raw 4x4 matrices. For `Warp3DMorphism`, writes a NIfTI vector field.

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 `write_linear_transform()` or `write_warp_field()`

Value

Invisibly returns `path`

write_warp_field *Write a warp morphism as a NIfTI vector field*

Description

Write a warp morphism as a NIfTI vector field

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` *Write X5 transform file*

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` *Create an X5 domain descriptor*

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 *X5 Transform IO*

Description

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

x5_transform *Create an X5 transform descriptor*

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

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

Value

An `X5Transform` object

Index