

---

# entente Documentation

**Metabolize**

**Sep 28, 2019**



---

## Contents

---

<b>1</b>	<b>entente package</b>	<b>1</b>
<b>2</b>	<b>Indices and tables</b>	<b>5</b>
	<b>Python Module Index</b>	<b>7</b>
	<b>Index</b>	<b>9</b>



## 1.1 Submodules

### 1.1.1 entente.cli module

### 1.1.2 entente.composite module

`entente.composite.composite_meshes` (*mesh\_paths*)

Create a composite as a vertex-wise average of several meshes in correspondence. Faces, groups, and other attributes are loaded from the first mesh given.

**Parameters** `mesh_paths` (*list*) – Paths of the meshes to average.

**Returns** The composite mesh.

**Return type** `lace.mesh.Mesh`

### 1.1.3 entente.equality module

Utilities related to mesh equality.

`entente.equality.attr_has_same_shape` (*first\_obj*, *second\_obj*, *attr*)

Given two objects, check if the given arraylike attributes of those objects have the same shape. If one object has an attribute value of `None`, the other must too.

**Parameters**

- **first\_obj** (*obj*) – A object with an arraylike `attr` attribute.
- **second\_obj** (*obj*) – Another object with an arraylike `attr` attribute.
- **attr** (*str*) – The name of the attribute to test.

**Returns** `True` if attributes are the same shape

**Return type** `bool`

`entente.equality.attr_is_equal (first_obj, second_obj, attr)`

Given two objects, check if the given arraylike attributes of those objects are equal. If one object has an attribute value of `None`, the other must too.

**Parameters**

- **first\_obj** (*obj*) – A object with an arraylike *attr* attribute.
- **second\_obj** (*obj*) – Another object with an arraylike *attr* attribute.
- **attr** (*str*) – The name of the attribute to test.

**Returns** *True* if attributes are equal

**Return type** `bool`

`entente.equality.have_same_topology (first_mesh, second_mesh)`

Given two meshes, check if they have the same vertex count and same faces. In other words, check if they have the same topology.

**Parameters**

- **first\_mesh** (*lace.mesh.Mesh*) – A mesh.
- **second\_mesh** (*lace.mesh.Mesh*) – Another mesh.

**Returns** *True* if meshes have the same topology

**Return type** `bool`

## 1.1.4 entente.landmarks module

## 1.1.5 entente.restore\_correspondence module

`entente.restore_correspondence.find_correspondence (a, b, atol=0.0001,  
all_must_match=True,  
ret_unmatched_b=False,  
progress=True)`

Given  $a[0], a[1], \dots, a[k]$  and  $b[0], b[1], \dots, b[j]$ , match each element of  $a$  to the corresponding element of  $b$ .

When *all\_must\_match* is *True*  $a$  and  $b$  must contain the same set of elements.  $b[\text{find\_correspondence}(a, b)]$  equals  $a$ . Otherwise, return  $-1$  for elements with no match in  $b$ .

**Parameters**

- **a** (*np.arraylike*) –  $k \times n$  array.
- **b** (*np.arraylike*) –  $j \times n$  array.
- **atol** (*float*) – Match tolerance.
- **all\_must\_match** (*bool*) – When *True*,  $a$  and  $b$  must contain the same elements.
- **ret\_unmatched\_b** (*bool*) – When *True*, return a tuple which also contains the indices of  $b$  which were not matched.
- **progress** (*bool*) – When *True*, show a progress bar.

**Returns** Indices of  $b$  as  $k \times 1$

**Return type** `np.ndarray`

---

**Note:** This relies on a brute-force algorithm.

For the interpretation of *atol*, see documentation for *np.isclose*.

---

```
entente.restore_correspondence.restore_correspondence (shuffled_mesh,      refer-
                                                         ence_mesh,      atol=0.0001,
                                                         progress=True)
```

Given a reference mesh, reorder the vertices of a shuffled copy to restore correspondence with the reference mesh. The vertex set of the shuffled mesh and reference mesh must be equal within *atol*. Mutate *reference\_mesh*. Ignore faces but preserves their integrity.

#### Parameters

- **reference\_mesh** (*lace.mesh.Mesh*) – A mesh with the vertices in the desired order.
- **shuffled\_mesh** (*lace.mesh.Mesh*) – A mesh with the same vertex set as *reference\_mesh*.
- **progress** (*bool*) – When *True*, show a progress bar.

**Returns** *vxI* which maps old vertices in *shuffled\_mesh* to new.

**Return type** *np.ndarray*

---

**Note:** This was designed to assist in extracting face ordering and groups from a *shuffled\_mesh* that “work” with *reference\_mesh*, so the face ordering and groups can be used with different vertices.

It relies on a brute-force algorithm.

---

## 1.1.6 entente.shuffle module

```
entente.shuffle.shuffle_faces (mesh)
```

Shuffle the mesh’s face ordering. The mesh is mutated.

**Parameters** *mesh* (*lace.mesh.Mesh*) – A mesh.

**Returns** *fxI* mapping of old face indices to new.

**Return type** *np.ndarray*

```
entente.shuffle.shuffle_vertices (mesh)
```

Shuffle the mesh’s vertex ordering, preserving the integrity of the faces. The mesh is mutated.

**Parameters** *mesh* (*lace.mesh.Mesh*) – A mesh.

**Returns** *vxI* mapping of old vertex indices to new.

**Return type** *np.ndarray*

## 1.1.7 entente.testing module

```
entente.testing.assert_same_face_set (a, b)
```

```
entente.testing.assert_same_vertex_set (a, b)
```

```
entente.testing.coord_set (a)
```

```
entente.testing.mesh_asset (*components)
```

```
entente.testing.relative_to_project(*components)
entente.testing.vitra_mesh()
```

### 1.1.8 entente.trimesh\_search module

On Mac OS:

```
brew install spatialindex
pip install rtree trimesh
```

`entente.trimesh_search.faces_nearest_to_points(mesh, query_points, ret_points=False)`  
Find the triangular faces on a mesh which are nearest to the given query points.

**Parameters**

- **query\_points** (*np.arraylike*) – The points to query, with shape *kx3*
- **ret\_points** (*bool*) – When *True*, return both the indices of the nearest faces and the closest points to the query points, which are not necessarily vertices. When *False*, return only the face indices.

**Returns** face indices as *kx1 np.ndarray*, or when *ret\_points* is *True*, a tuple also including the coordinates of the closest points as *kx3 np.ndarray*.

**Return type** object

```
entente.trimesh_search.require_trimesh_with_rtree()
```

Check that trimesh and rtree are installed and can be imported, and raise an error with a helpful error message if they are not.



## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### e

- `entente`, [1](#)
- `entente.composite`, [1](#)
- `entente.equality`, [1](#)
- `entente.restore_correspondence`, [2](#)
- `entente.shuffle`, [3](#)
- `entente.testing`, [3](#)
- `entente.trimesh_search`, [4](#)



## A

`assert_same_face_set()` (in module *entente.testing*), 3

`assert_same_vertex_set()` (in module *entente.testing*), 3

`attr_has_same_shape()` (in module *entente.equality*), 1

`attr_is_equal()` (in module *entente.equality*), 1

## C

`composite_meshes()` (in module *entente.composite*), 1

`coord_set()` (in module *entente.testing*), 3

## E

*entente* (module), 1

*entente.composite* (module), 1

*entente.equality* (module), 1

*entente.restore\_correspondence* (module), 2

*entente.shuffle* (module), 3

*entente.testing* (module), 3

*entente.trimesh\_search* (module), 4

## F

`faces_nearest_to_points()` (in module *entente.trimesh\_search*), 4

`find_correspondence()` (in module *entente.restore\_correspondence*), 2

## H

`have_same_topology()` (in module *entente.equality*), 2

## M

`mesh_asset()` (in module *entente.testing*), 3

## R

`relative_to_project()` (in module *entente.testing*), 3

`require_trimesh_with_rtree()` (in module *entente.trimesh\_search*), 4

`restore_correspondence()` (in module *entente.restore\_correspondence*), 3

## S

`shuffle_faces()` (in module *entente.shuffle*), 3

`shuffle_vertices()` (in module *entente.shuffle*), 3

## V

`vitra_mesh()` (in module *entente.testing*), 4