entente Documentation

Metabolize

Oct 02, 2018

Contents

CHAPTER 1

entente package

1.1 Submodules

1.1.1 entente.cgal_search module

cgal_search provides spatial search for vertices and faces. This is implemented atop CGAL's axis-aligned-bounding-box search tree.

CGAL is a heavy dependency and therefore is optional. Before using this module, you must install CGAL and its Python bindings (which take quite some time to build).

On Mac OS:

```
brew install cgal swig
pip install cgal-bindings
# wait approximately one year
```

Note: The AABB tree is in the GPLv3-licensed portion of CGAL

entente.cgal_search.**create_aabb_tree**(*mesh*)

Create a CGAL AABB tree from the given mesh.

Reports suggest trees may rely on some shared internal storage in CGAL, so to be conservative, *finish using one before creating another*.

See also:

- https://doc.cgal.org/latest/AABB_tree/index.html
- https://github.com/CGAL/cgal-swig-bindings/blob/master/examples/python/AABB_triangle_3_example.py

Returns A CGAL AABB tree.

Return type CGAL.CGAL_AABB_tree

entente.cgal_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.cgal_search.**require_cgal**()
Check that CGAL is installed, and raise an error with a helpful error message if it is not.

1.1.2 entente.cli module

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.geometry module

Functions relating to mesh geometry.

entente.geometry.**compute_barycentric_coordinates**(*vertices_of_tris*, *points*)

Compute barycentric coordinates for the projection of a set of points to a given set of triangles specfied by their vertices.

These barycentric coordinates can refer to points outside the triangle. This happens when one of the coordinates is negative. However they can't specify points outside the triangle's plane. (That requires tetrahedral coordinates.)

The returned coordinates supply a linear combination which, applied to the vertices, returns the projection of the original point the plane of the triangle.

Parameters

- **vertices_of_tris** (*np.arraylike*) – A set of triangle vertices as *kx3x3*.
- **points** (*np.arraylike*) – Coordinates of points as *kx3*.

Returns Barycentric coordinates as *kx3*

Return type np.ndarray

See also:

https://en.wikipedia.org/wiki/Barycentric_coordinate_system

Note: A function with this signature probably belongs in *blmath*.

1.1.5 entente.landmarks module

1.1.6 entente.validation module

entente.validation.**validate_shape**(*a*, **shape*, ***kwargs*)

Check that the given argument has the expected shape. Shape dimensions can be ints or -1 for a wildcard. The wildcard dimensions are returned, which allows them to be used for subsequent validation or elsewhere in the function.

Parameters

- **a** (*np.arraylike*) – An array-like input.
- **shape** (*list*) – Shape to validate. To require 3 by 1, pass *3*. To require n by 3, pass *-1, 3*.
- **name** (*str*) – Variable name to embed in the error message.

Returns The wildcard dimension (if one) or a tuple of wildcard dimensions (if more than one).

Return type object

entente.validation.**validate_shape_from_ns**(*namespace*, *name*, **shape*)

Convenience function for invoking *validate_shape()* with a *locals()* dict.

Parameters

- **namespace** (*dict*) – A subscriptable object, typically *locals()*.
- **name** (*str*) – Key to pull from *namespace*.
- **shape** (*list*) – Shape to validate. To require 3 by 1, pass *3*. To require n by 3, pass *-1, 3*.

Returns The wildcard dimension (if one) or a tuple of wildcard dimensions (if more than one).

Return type object

Example

validate_shape_from_namespace(locals(), 'points', -1, 3)

1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index