directional_clustering

Release 0.1.0

Rafael Pastrana, Isabel Moreira, Alex Papamatthaiou, Hui Yuan

Dec 08, 2020

CONTENTS

1 Introduction **1**

2 Getting Started **3**
- 2.1 Installation . . . 3
- 2.2 Developer Mode . . . 4
  - 2.2.1 Testing . . . 4
  - 2.2.2 Documentation . . . 4

3 API Reference **5**
- 3.1 directional_clustering . . . 5
  - 3.1.1 directional_clustering.clustering . . . 5
  - 3.1.2 directional_clustering.fields . . . 11
  - 3.1.3 directional_clustering.mesh . . . 16
  - 3.1.4 directional_clustering.plotters . . . 69
  - 3.1.5 directional_clustering.transformations . . . 69

4 License **71**

Python Module Index **73**

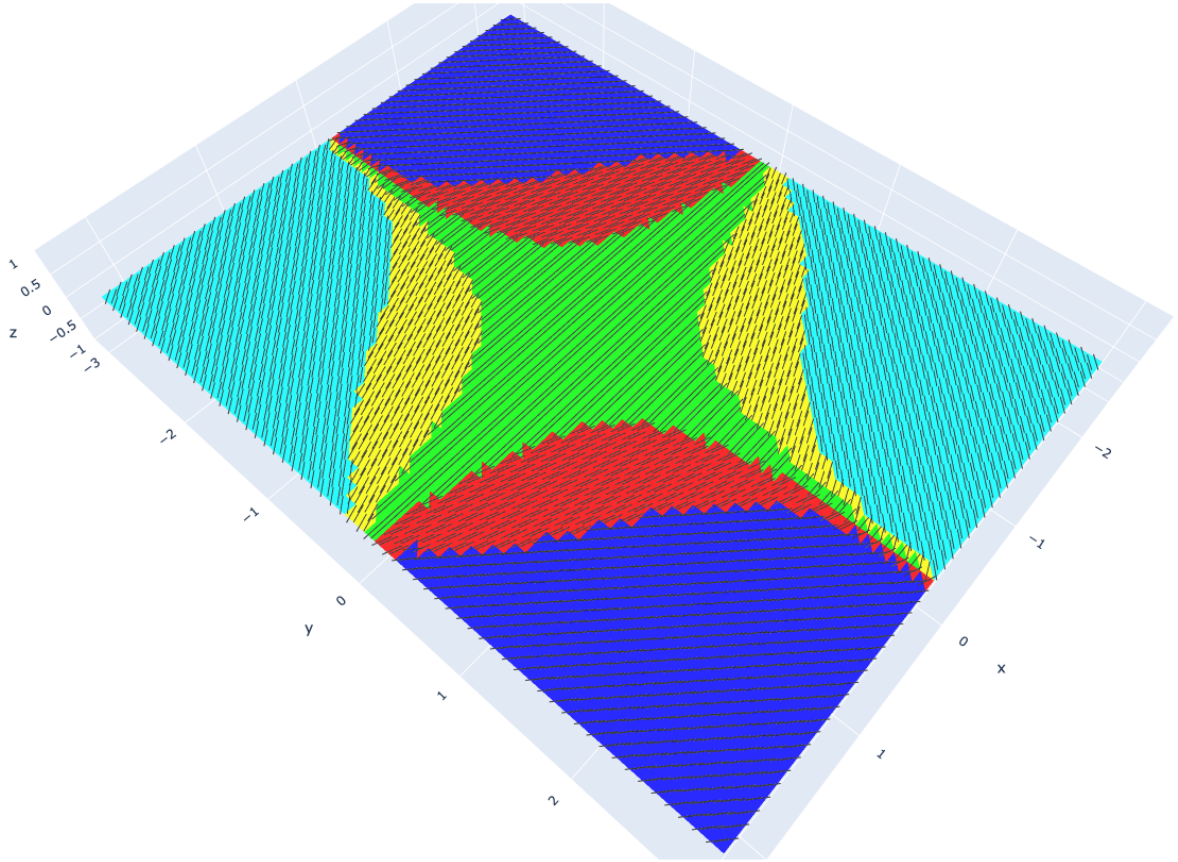
Index **75**

CHAPTER

ONE

INTRODUCTION

Directional clustering of vector fields on meshes.



The initial motivation of this work revolved around principal stress fields. In principle, they suggest good directions to orient material efficiently in architectural structures. This implies that by following these directions, less material would be used to achieve a target level of structural performance.

Principal stress fields are ubiquitously computed by off-the-shelf FEA software and are represented as a cloud of vectors (i.e. a vector field).

As principal stress fields are heterogeneous and form continuous curvilinear trajectories, it is actually difficult for fabrication reasons to place material (in the form reinforcement bars or beams) in a way that exactly match the field directions. It is almost cumbersome, and this is probably one of the reasons why we actually keep on building with orthogonal grids everywhere (take a look at the room around you, for example).

In this work we question the heterogeneity of a principal stress field and inquiry on how much we can simplify it so that we can maximize fabricability while compromising as little as possible in structural performance. In short, what we want is to find the lowest possible amount of different vectors that encode the maximum amount of directional information about a principal stress field. We leverage clustering methods to this end.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

The simplest way to install `directional_clustering` is to build it from source after cloning this repo. For developer mode, please jump to the next section.

1. First, we would need to install the latest version of Anaconda. Anaconda will take care, among many other things, of installing scientific computing packages like `numpy` and `matplotlib` for us.

2. Next, let's create a new `conda` environment from your command line interface (your terminal on macOS or from the anaconda prompt on windows). The only required dependencies are `compas` and `sklearn`.

```
conda create -n clusters python=3.7 COMPAS=0.16.9 scikit-learn
conda activate clusters
```

3. We should clone `directional_clustering` from this repository and move inside. If you are a macOS user and want to put it in your home folder:

```
cd ~
git clone https://github.com/arpastrana/directional_clustering.git
cd directional_clustering
```

4. Next, install `directional_clustering` as an editable package from source using `pip`:

```
pip install -e .
```

5. To double-check that everything is up and running, still in your command line interface, let's type the following and hit enter:

```
python -c "import directional_clustering"
```

If no errors occur, smile :)! You have a working installation of `directional_clustering`.

2.2 Developer Mode

If you are rather interested in building the documentation, testing, or making a pull request to this package, you should install this package slighly differently.

Concretely, instead of running `pip install -e .` in step 4 above, we must do:

```
pip install -r requirements-dev.txt
```

This will take care of installing additional dependencies like `sphinx` and `pytest`.

2.2.1 Testing

To run the `pytest` suite automatically, type from the command line;

```
invoke test
```

2.2.2 Documentation

To build this package's documentation in `html`, type:

```
invoke docs
```

You'll find the generated `html` data in the `docs/` folder.

If instead what we need is a manual in `pdf` format, let's run:

```
invoke pdf
```

The manual will be saved in `docs/latex` as `directional_clustering.pdf`.

CHAPTER

THREE

API REFERENCE

3.1 directional_clustering

3.1.1 directional_clustering.clustering

Clustering Classes

KMeans	Generic K-means clustering algorithm.
CosineKMeans	K-means clustering using cosine distance as the association metric.
VariationalKMeans	The variational shape approximation method for vector clustering.

KMeans

class directional_clustering.clustering.**KMeans**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)

Generic K-means clustering algorithm.

Parameters

- **mesh** (*directional_clustering.mesh.MeshPlus*) – A reference mesh.
- **vector_field** (*directional_clustering.fields.VectorField*) – The vector field to cluster.
- **n_clusters** (*int*) – The number of clusters to generate.
- **iters** (*int*) – The iterations to run the algorithm for.
- **tol** (*float*) – The tolerance to declare convergence.

Attributes

clustered_field	The clustered vector field.
labels	A mapping from a vector field's keys to the indices of the clusters centers.
loss	The total loss that k-means produced after clustering a vector field.

Inherited Attributes

—

Methods

__init__(mesh, vector_field, n_clusters, ...)	Initialize self.
cluster()	Cluster a vector field.

KMeans.__init__

KMeans.**__init__**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)
Initialize self. See help(type(self)) for accurate signature.

KMeans.cluster

KMeans.**cluster**()
Cluster a vector field.

Notes

It sets *self._clustered_field*, *self_labels*, *self.centers*, and *self.loss*. Returns *None*.

Inherited Methods

—

CosineKMeans

class directional_clustering.clustering.**CosineKMeans**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)
K-means clustering using cosine distance as the association metric.

Parameters

- **mesh** (*directional_clustering.mesh.MeshPlus*) – A reference mesh. Reserved.
- **vector_field** (*directional_clustering.fields.VectorField*) – The vector field to cluster.
- **n_clusters** (*int*) – The number of clusters to generate.
- **iters** (*int*) – The iterations to run the algorithm for.
- **tol** (*float*) – The tolerance to declare convergence.

Attributes

—

Inherited Attributes

clustered_field	The clustered vector field.
labels	A mapping from a vector field's keys to the indices of the clusters centers.
loss	The total loss that k-means produced after clustering a vector field.

Methods

__init__(mesh, vector_field, n_clusters, ...)	Initialize self.

CosineKMeans.__init__

CosineKMeans.**__init__**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)
Initialize self. See help(type(self)) for accurate signature.

Inherited Methods

cluster()	Cluster a vector field.

CosineKMeans.cluster

CosineKMeans.**cluster**()
Cluster a vector field.

Notes

It sets *self._clustered_field*, *self_labels*, *self.centers*, and *self.loss*. Returns *None*.

VariationalKMeans

class directional_clustering.clustering.**VariationalKMeans**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)

The variational shape approximation method for vector clustering.

Parameters

- **mesh** (*directional_clustering.mesh.MeshPlus*) – A reference mesh.
- **vector_field** (*directional_clustering.fields.VectorField*) – The vector field to cluster.
- **n_clusters** (*int*) – The number of clusters to generate.

- **iters** (*int*) – The iterations to run the algorithm for.
- **tol** (*float*) – The tolerance to declare convergence.

Notes

This method normalizes all vectors before doing clustering.

References

[1] Cohen-Steiner, D., Alliez, P., Desbrun, M. (2004). Variational Shape Approximation. RR-5371, INRIA. 2004, pp.29. inria-00070632

Attributes

—

Inherited Attributes

`clustered_field`	The clustered vector field.
`labels`	A mapping from a vector field's keys to the indices of the clusters centers.
`loss`	The total loss that k-means produced after clustering a vector field.

Methods

`__init__`(mesh, vector_field, n_clusters, ...)	Initialize self.
`cluster`()	Cluster a vector field.

VariationalKMeans.__init__

`VariationalKMeans.`**`__init__`**(*mesh*, *vector_field*, *n_clusters*, *iters*, *tol*)

Initialize self. See help(type(self)) for accurate signature.

VariationalKMeans.cluster

VariationalKMeans.**cluster**()
Cluster a vector field.

Notes

It sets *self._clustered_field*, *self_labels*, *self.centers*, and *self.loss*. Returns *None*.

Inherited Methods

—

Factory Classes

ClusteringFactory	A factory to unify the creation of clustering algorithms.

ClusteringFactory

class directional_clustering.clustering.**ClusteringFactory**
A factory to unify the creation of clustering algorithms.

Attributes

supported_algorithms

Inherited Attributes

—

Methods

create(name)	Creates an unitialized clustering algorithm.
register(name, algorithm)	Registers a clustering algorithm to the factory's database.

ClusteringFactory.create

classmethod ClusteringFactory.**create**(*name*)
Creates an unitialized clustering algorithm.

Parameters **name** (*str*) – The name of the clustering algorithm to generate.

Returns **algorithm** (*directional_clustering.clustering.ClusteringAlgorithm*) – A clustering algorithm to instantiate.

ClusteringFactory.register

classmethod ClusteringFactory.**register**(*name*, *algorithm*)
Registers a clustering algorithm to the factory's database.

Parameters

- **name** (*str*) – The name key by which a clustering will be stored.
- **algorithm** (*directional_clustering.clustering.ClusteringAlgorithm*) – A clustering algorithm.

Inherited Methods

__init__()	Initialize self.

ClusteringFactory.__init__

ClusteringFactory.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

Abstract Classes

ClusteringAlgorithm	Abstract base class for all clustering algorithms.

ClusteringAlgorithm

class directional_clustering.clustering.**ClusteringAlgorithm**
Abstract base class for all clustering algorithms.

Attributes

clustered_field	The clustered vector field.
labels	The labels that reference entries in the vector field to clusters.
loss	The final error of the produced by the clustering method.

Inherited Attributes

—

Methods

cluster(*args, **kwargs)	Main clustering method.

ClusteringAlgorithm.cluster

abstract ClusteringAlgorithm.**cluster**(**args*, ***kwargs*)
Main clustering method.

Parameters

- **args** (*list*, optional) – Default arguments.
- **kwargs** (*dict*, optional) – Default keyword arguments.

Inherited Methods

__init__()	Initialize self.

ClusteringAlgorithm.__init__

ClusteringAlgorithm.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

3.1.2 directional_clustering.fields

Fields

Field	A concrete field.
VectorField	A field with a fixed dimensionality of 3.

Field

class directional_clustering.fields.**Field**(*dimensionality*)

A concrete field.

Basically, a container for scalars and vectors. One key can store exclusively one value at a time.

It is crucial to have it as a datastructure where a field's entries are accessed with the keys of the Mesh they are coupled to.

Parameters **dimensionality** (*int*) – The dimensionality of the field.

Methods

__init__(dimensionality)	The constructor.
dimensionality()	The fixed dimensionality of a field.
size()	The number of items stored in the field.

Field.__init__

Field.**__init__**(*dimensionality*)

The constructor.

Field.dimensionality

Field.**dimensionality**()

The fixed dimensionality of a field.

Returns **dimensionality** (*int*) – The dimensionality of the field.

Field.size

Field.**size**()

The number of items stored in the field.

Returns **size** (*int*) – The number of items.

Inherited Methods

—

VectorField

class directional_clustering.fields.**VectorField**

A field with a fixed dimensionality of 3.

Methods

__init__()	The constructor.
add_vector(key, vector)	Adds a vector entry to a vector field.
from_mesh_faces(mesh, name)	Extracts a vector field from the faces of a mesh.
from_sequence(sequence)	Creates a vector field from a sequence.
items()	Iterates over the keys and the vectors of the field.
keys()	Iterates over they access keys of the vector field.
remove_vector(key)	Deletes a vector from the vector field.
to_sequence()	Converts a vector field into a sequence.
vector(key)	Queries a vector from a vector field.
vectors()	Iterates over the vectors of the vector field.

VectorField.__init__

VectorField.**__init__**()

The constructor.

VectorField.add_vector

VectorField.**add_vector**(*key*, *vector*)

Adds a vector entry to a vector field.

Parameters

- **key** (*int*) – The key to store the vector with.
- **vector** (*list* of *float*) – A vector in 3d space.

VectorField.from_mesh_faces

classmethod VectorField.**from_mesh_faces**(*mesh*, *name*)

Extracts a vector field from the faces of a mesh.

Parameters

- **mesh** (*directional_clustering.mesh.MeshPlus*) – A mesh.
- **name** (*str*) – The name of the face attribute to query.

Returns **vector_field** (*VectorField*) – A vector field.

Notes

Deprecated. Every vector is stored with the mesh face keys as access keys.

VectorField.from_sequence

classmethod VectorField.**from_sequence**(*sequence*)
Creates a vector field from a sequence.

Parameters **sequence** (*list* of *list*) – A list of vectors.

Returns **vector_field** (*directional_clustering.fields.VectorField*) – A vector field.

Notes

The vectors are stored in the order they are supplied. Access keys are generated in the range from 0 to the sequence length.

VectorField.items

VectorField.**items**()
Iterates over the keys and the vectors of the field.

Yields

- **key** (*int*) – The next access key in the vector field.
- **vector** (*list* of *float*) – The next vector in the vector field.

VectorField.keys

VectorField.**keys**()
Iterates over they access keys of the vector field.

Yields **key** (*int*) – The next access key in the vector field.

VectorField.remove_vector

VectorField.**remove_vector**(*key*)
Deletes a vector from the vector field.

Parameters **key** (*int*) – The key of the vector to remove.

VectorField.to_sequence

VectorField.**to_sequence**()
Converts a vector field into a sequence.

Returns **sequence** (*list* of *list*) – A list of vectors.

Notes

The output vectors are not sorted by their access keys.

VectorField.vector

VectorField.**vector**(*key*)
Queries a vector from a vector field.

Parameters **key** (*int*) – The key of the vector to retrieve.

Returns **vector** (*list* of *float*) – A vector.

VectorField.vectors

VectorField.**vectors**()
Iterates over the vectors of the vector field.

Yields **vector** (*list* of *float*) – The next vector in the vector field.

Inherited Methods

dimensionality()	The fixed dimensionality of a field.
size()	The number of items stored in the field.

VectorField.dimensionality

VectorField.**dimensionality**()
The fixed dimensionality of a field.

Returns **dimensionality** (*int*) – The dimensionality of the field.

VectorField.size

VectorField.**size**()
The number of items stored in the field.

Returns **size** (*int*) – The number of items.

Abstract Classes

AbstractField	An abstract class for all fields.

AbstractField

class directional_clustering.fields.**AbstractField**
An abstract class for all fields.

Methods

dimensionality()	The fixed dimensionality of a field.
size()	The number of entries in the field.

AbstractField.dimensionality

abstract AbstractField.**dimensionality**()
The fixed dimensionality of a field.

AbstractField.size

abstract AbstractField.**size**()
The number of entries in the field.

Inherited Methods

__init__()	Initialize self.

AbstractField.__init__

AbstractField.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

3.1.3 directional_clustering.mesh

Classes

MeshPlus	An extended COMPAS mesh with specialized methods to parse vector fields.

MeshPlus

class directional_clustering.mesh.**MeshPlus**

An extended COMPAS mesh with specialized methods to parse vector fields.

Parameters

- **args** (*list*, optional.) – Default arguments.
- **kwargs** (*dict*, optional.) – Default keyword arguments.

Notes

See *help(compas.datastructures.Mesh)* for details on the constructor's signature.

Attributes

—

Inherited Attributes

DATASCHEMA	The schema of the data of this object.
JSONSCHEMA	The schema of the JSON representation of the data of this object.
adjacency	
data	A data dict representing the mesh data structure for serialisation.
dtype	str: The type of the object in the form of a "2-level" import and a class name.
guid	The globally unique identifier of the object.
name	The name of the data structure.

Methods

clustering_label(name[, labels])	Gets or sets cluster labels on a mesh.
vector_field(name[, vector_field])	Gets or sets a vector field that lives on the mesh.
vector_fields()	Queries the names of all the vector fields stored on the mesh.

MeshPlus.clustering_label

MeshPlus.**clustering_label**(*name*, *labels=None*)

Gets or sets cluster labels on a mesh.

Parameters

- **name** (*str*) – The name of the cluster label. The format is {vector_field_name}_{algorithm}_{number_of_clusters}.
- **labels** (*dict*, optional.) – The cluster labels to store. Defaults to *None*.

Returns **labels** (*dict*) – The fetched labels only if a *name* was input.

MeshPlus.vector_field

MeshPlus.**vector_field**(*name*, *vector_field=None*)

Gets or sets a vector field that lives on the mesh.

Parameters

- **name** (*str*) – The name of the vector field to get or to set.
- **vector_field** (*directional_clustering.fields.VectorField*, optional.) – The vector field to store. Defaults to *None*.

Returns **vector_field** (*directional_clustering.fields.VectorField*.) – The fetched vector field if a *name* was input.

Notes

Vector fields are stored a face attributes of a mesh. Refer to *compas.datastructures.face_attribute()* for more details.

MeshPlus.vector_fields

MeshPlus.**vector_fields**()

Queries the names of all the vector fields stored on the mesh.

Returns **attr_vectorfield** (*list*) – A list of with the vector field names.

Inherited Methods

__init__()	Initialize self.
add_face(vertices[, fkey, attr_dict])	Add a face to the mesh object.
add_vertex([key, attr_dict])	Add a vertex to the mesh object.
area()	Calculate the total mesh area.
bounding_box()	Compute the (axis aligned) bounding box of a mesh.
bounding_box_xy()	Compute the (axis aligned) bounding box of a projection of the mesh in the XY plane.
centroid()	Calculate the mesh centroid.
clear()	Clear all the mesh data.
collapse_edge(u, v[, t, allow_boundary, fixed])	Collapse an edge to its first or second vertex, or to an intermediate point.
connected_components()	
copy([cls])	Make an independent copy of the datastructure object.
cull_vertices()	Remove all unused vertices from the mesh object.
cut(plane)	Cut a mesh with a plane and construct the resulting submeshes.
delete_face(fkey)	Delete a face from the mesh object.
delete_vertex(key)	Delete a vertex from the mesh and everything that is attached to it.

continues on next page

Table 36 – continued from previous page

dual([cls])	Construct the dual of a mesh.
edge_attribute(edge, name[, value])	Get or set an attribute of an edge.
edge_attributes(edge[, names, values])	Get or set multiple attributes of an edge.
edge_coordinates(u, v[, axes])	Return the coordinates of the start and end point of an edge.
edge_direction(u, v)	Return the direction vector of an edge.
edge_faces(u, v)	Find the two faces adjacent to an edge.
edge_length(u, v)	Return the length of an edge.
edge_loop(edge)	Find all edges on the same loop as a given edge.
edge_midpoint(u, v)	Return the location of the midpoint of an edge.
edge_point(u, v[, t])	Return the location of a point along an edge.
edge_strip(edge)	Find all edges on the same strip as a given edge.
edge_vector(u, v)	Return the vector of an edge.
edges([data])	Iterate over the edges of the mesh.
edges_attribute(name[, value, keys])	Get or set an attribute of multiple edges.
edges_attributes([names, values, keys])	Get or set multiple attributes of multiple edges.
edges_on_boundaries()	
edges_on_boundary([oriented])	Find the edges on the boundary.
edges_where(conditions[, data])	Get edges for which a certain condition or set of conditions is true.
edges_where_predicate(predicate[, data])	Get edges for which a certain condition or set of conditions is true using a lambda function.
euler()	Calculate the Euler characteristic.
face_adjacency()	Build a face adjacency dict.
face_adjacency_halfedge(f1, f2)	Find one half-edge over which two faces are adjacent.
face_adjacency_vertices(f1, f2)	Find all vertices over which two faces are adjacent.
face_area(fkey)	Compute the area of a face.
face_aspect_ratio(fkey)	Face aspect ratio as the ratio between the lengths of the maximum and minimum face edges.
face_attribute(key, name[, value])	Get or set an attribute of a face.
face_attributes(key[, names, values])	Get or set multiple attributes of a face.
face_center(fkey)	Compute the location of the center of mass of a face.
face_centroid(fkey)	Compute the location of the centroid of a face.
face_coordinates(fkey[, axes])	Compute the coordinates of the vertices of a face.
face_corners(fkey)	Return triplets of face vertices forming the corners of the face.
face_curvature(fkey)	Dimensionless face curvature as the maximum face vertex deviation from the best-fit plane of the face vertices divided by the average lengths of the face vertices to the face centroid.
face_degree(fkey)	Count the neighbors of a face.
face_flatness(fkey[, maxdev])	Compute the flatness of the mesh face.
face_halfedges(fkey)	The halfedges of a face.
face_max_degree()	Compute the maximum degree of all faces.
face_min_degree()	Compute the minimum degree of all faces.
face_neighborhood(key[, ring])	Return the faces in the neighborhood of a face.
face_neighbors(fkey)	Return the neighbors of a face across its edges.
face_normal(fkey[, unitized])	Compute the normal of a face.
face_plane(face)	A plane defined by the centroid and the normal of the face.

continues on next page

Table 36 – continued from previous page

face_skewness(fkey)	Face skewness as the maximum absolute angular deviation from the ideal polygon angle.
face_vertex_ancestor(fkey, key[, n])	Return the n-th vertex before the specified vertex in a specific face.
face_vertex_descendant(fkey, key[, n])	Return the n-th vertex after the specified vertex in a specific face.
face_vertices(fkey)	The vertices of a face.
faces([data])	Iterate over the faces of the mesh.
faces_attribute(name[, value, keys])	Get or set an attribute of multiple faces.
faces_attributes([names, values, keys])	Get or set multiple attributes of multiple faces.
faces_on_boundary()	Find the faces on the boundary.
faces_where(conditions[, data])	Get faces for which a certain condition or set of conditions is true.
faces_where_predicate(predicate[, data])	Get faces for which a certain condition or set of conditions is true using a lambda function.
flip_cycles()	Flip the cycle directions of all faces.
from_data(data)	Construct a datastructure from structured data.
from_json(filepath)	Construct a datastructure from structured data contained in a json file.
from_lines(lines[, delete_boundary_face, ...])	Construct a mesh object from a list of lines described by start and end point coordinates.
from_obj(filepath[, precision])	Construct a mesh object from the data described in an OBJ file.
from_off(filepath)	Construct a mesh object from the data described in a OFF file.
from_ply(filepath[, precision])	Construct a mesh object from the data described in a PLY file.
from_points(points[, boundary, holes])	Construct a mesh from a delaunay triangulation of a set of points.
from_polygons(polygons[, precision])	Construct a mesh from a series of polygons.
from_polyhedron(f)	Construct a mesh from a platonic solid.
from_polylines(boundary_polylines, ...)	Construct mesh from polylines.
from_shape(shape, **kwargs)	Construct a mesh from a primitive shape.
from_stl(filepath[, precision])	Construct a mesh object from the data described in a STL file.
from_vertices_and_faces(vertices, faces)	Construct a mesh object from a list of vertices and faces.
genus()	Calculate the genus.
get_any_face()	Get the identifier of a random face.
get_any_face_vertex(fkey)	Get the identifier of a random vertex of a specific face.
get_any_vertex()	Get the identifier of a random vertex.
get_any_vertices(n[, exclude_leaves])	Get a list of identifiers of a random set of n vertices.
gkey_key([precision])	Returns a dictionary that maps *geometric keys* of a certain precision to the keys of the corresponding vertices.
halfedge_face(u, v)	Find the face corresponding to a halfedge.
has_edge(key)	Verify that the mesh contains a specific edge.
has_face(fkey)	Verify that a face is part of the mesh.
has_halfedge(key)	Verify that a halfedge is part of the mesh.
has_vertex(key)	Verify that a vertex is in the mesh.

continues on next page

Table 36 – continued from previous page

index_key()	Returns a dictionary that maps the indices of a vertex list to keys in a vertex dictionary.
index_vertex()	Returns a dictionary that maps the indices of a vertex list to keys in a vertex dictionary.
insert_vertex(fkey[, key, xyz, return_fkeys])	Insert a vertex in the specified face.
is_connected()	Verify that the mesh is connected.
is_edge_on_boundary(u, v)	Verify that an edge is on the boundary.
is_empty()	Boolean whether the mesh is empty.
is_face_on_boundary(key)	Verify that a face is on a boundary.
is_manifold()	Verify that the mesh is manifold.
is_orientable()	Verify that the mesh is orientable.
is_quadmesh()	Verify that the mesh consists of only quads.
is_regular()	Verify that the mesh is regular.
is_trimesh()	Verify that the mesh consists of only triangles.
is_valid()	Verify that the mesh is valid.
is_vertex_connected(key)	Verify that a vertex is connected.
is_vertex_on_boundary(key)	Verify that a vertex is on a boundary.
join(other)	Add the vertices and faces of another mesh to the current mesh.
key_gkey([precision])	Returns a dictionary that maps vertex dictionary keys to the corresponding *geometric key* up to a certain precision.
key_index()	Returns a dictionary that maps vertex dictionary keys to the corresponding index in a vertex list or array.
normal()	Calculate the average mesh normal.
number_of_edges()	Count the number of edges in the mesh.
number_of_faces()	Count the number of faces in the mesh.
number_of_vertices()	Count the number of vertices in the mesh.
quads_to_triangles([check_angles])	
remove_unused_vertices()	Remove all unused vertices from the mesh object.
smooth_area([fixed, kmax, damping, . . .])	Smooth a mesh by moving each vertex to the barycenter of the centroids of the surrounding faces, weighted by area.
smooth_centroid([fixed, kmax, damping, . . .])	Smooth a mesh by moving every free vertex to the centroid of its neighbors.
split_edge(u, v[, t, allow_boundary])	Split and edge by inserting a vertex along its length.
split_face(fkey, u, v)	Split a face by inserting an edge between two specified vertices.
summary()	Print a summary of the mesh.
to_data()	Returns a dictionary of structured data representing the data structure.
to_json(filepath[, pretty])	Serialise the structured data representing the datastructure to json.
to_lines(filepath)	
to_obj(filepath[, precision, unweld])	Write the mesh to an OBJ file.
to_off(filepath, **kwargs)	Write a mesh object to an OFF file.
to_ply(filepath, **kwargs)	Write a mesh object to a PLY file.
to_points()	
to_polygons()	
to_polylines()	
to_quadmesh()	

continues on next page

Table 36 – continued from previous page

to_stl(filepath[, precision, binary])	Write a mesh to an STL file.
to_trimesh()	
to_vertices_and_faces()	Return the vertices and faces of a mesh.
transform(transformation)	Transform a mesh.
transform_numpy(M)	
transformed(transformation)	Transform a copy of mesh.
unify_cycles([root])	Unify the cycle directions of all faces.
unset_edge_attribute(edge, name)	Unset the attribute of an edge.
unset_face_attribute(key, name)	Unset the attribute of a face.
unset_vertex_attribute(key, name)	Unset the attribute of a vertex.
update_default_edge_attributes([attr_dict])	Update the default edge attributes.
update_default_face_attributes([attr_dict])	Update the default face attributes.
update_default_vertex_attributes([attr_dict])	Update the default vertex attributes.
validate_data()	Validate the data of this object against its data schema (*self.DATASCHEMA*).
validate_json()	Validate the data loaded from a JSON representation of the data of this object against its data schema (*self.DATASCHEMA*).
vertex_area(key)	Compute the tributary area of a vertex.
vertex_attribute(key, name[, value])	Get or set an attribute of a vertex.
vertex_attributes(key[, names, values])	Get or set multiple attributes of a vertex.
vertex_coordinates(key[, axes])	Return the coordinates of a vertex.
vertex_curvature(vkey)	Dimensionless vertex curvature.
vertex_degree(key)	Count the neighbors of a vertex.
vertex_faces(key[, ordered, include_none])	The faces connected to a vertex.
vertex_index()	Returns a dictionary that maps vertex dictionary keys to the corresponding index in a vertex list or array.
vertex_laplacian(key)	Compute the vector from a vertex to the centroid of its neighbors.
vertex_max_degree()	Compute the maximum degree of all vertices.
vertex_min_degree()	Compute the minimum degree of all vertices.
vertex_neighborhood(key[, ring])	Return the vertices in the neighborhood of a vertex.
vertex_neighborhood_centroid(key)	Compute the centroid of the neighbors of a vertex.
vertex_neighbors(key[, ordered])	Return the neighbors of a vertex.
vertex_normal(key)	Return the normal vector at the vertex as the weighted average of the normals of the neighboring faces.
vertices([data])	Iterate over the vertices of the mesh.
vertices_attribute(name[, value, keys])	Get or set an attribute of multiple vertices.
vertices_attributes([names, values, keys])	Get or set multiple attributes of multiple vertices.
vertices_on_boundaries()	Find the vertices on all boundaries of the mesh.
vertices_on_boundary([ordered])	Find the vertices on the boundary.
vertices_where(conditions[, data])	Get vertices for which a certain condition or set of conditions is true.
vertices_where_predicate(predicate[, data])	Get vertices for which a certain condition or set of conditions is true using a lambda function.

MeshPlus.__init__

MeshPlus.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

MeshPlus.add_face

MeshPlus.**add_face**(*vertices*, *fkey=None*, *attr_dict=None*, ***kwattr*)
Add a face to the mesh object.

Parameters

- **vertices** (*list*) – A list of vertex keys.
- **attr_dict** (*dict, optional*) – Face attributes.
- **kwattr** (*dict, optional*) – Additional named face attributes. Named face attributes overwrite corresponding attributes in the attribute dict (`attr_dict`).

Returns *int* – The key of the face.

Raises **TypeError** – If the provided face key is of an unhashable type.

Notes

If no key is provided for the face, one is generated automatically. An automatically generated key is an integer that increments the highest integer value of any key used so far by 1.

If a key with an integer value is provided that is higher than the current highest integer key value, then the highest integer value is updated accordingly.

Examples

```
>>>
```

MeshPlus.add_vertex

MeshPlus.**add_vertex**(*key=None*, *attr_dict=None*, ***kwattr*)
Add a vertex to the mesh object.

Parameters

- **key** (*int, optional*) – The vertex identifier.
- **attr_dict** (*dict, optional*) – Vertex attributes.
- **kwattr** (*dict, optional*) – Additional named vertex attributes. Named vertex attributes overwrite corresponding attributes in the attribute dict (`attr_dict`).

Returns *int* – The identifier of the vertex.

Notes

If no key is provided for the vertex, one is generated automatically. An automatically generated key is an integer that increments the highest integer value of any key used so far by 1.

If a key with an integer value is provided that is higher than the current highest integer key value, then the highest integer value is updated accordingly.

Examples

```
>>> mesh.add_vertex()
0
>>> mesh.add_vertex(x=0, y=0, z=0)
1
>>> mesh.add_vertex(key=2)
2
>>> mesh.add_vertex(key=0, x=1)
0
```

MeshPlus.area

MeshPlus.**area**()

Calculate the total mesh area.

Returns *float* – The area.

MeshPlus.bounding_box

MeshPlus.**bounding_box**()

Compute the (axis aligned) bounding box of a mesh.

Parameters **mesh** (*compas.datastructures.Mesh*) – The mesh data structure.

Returns *list of point* – The 8 corners of the bounding box of the mesh.

Examples

```
>>> mesh_bounding_box(mesh)
[[0.0, 0.0, 0.0], [10.0, 0.0, 0.0], [10.0, 10.0, 0.0], [0.0, 10.0, 0.0], [0.0,
↪ 0.0, 0.0], [10.0, 0.0, 0.0], [10.0, 10.0, 0.0], [0.0, 10.0, 0.0]]
```

MeshPlus.bounding_box_xy

MeshPlus.**bounding_box_xy**()

Compute the (axis aligned) bounding box of a projection of the mesh in the XY plane.

Parameters **mesh** (*compas.datastructures.Mesh*) – The mesh data structure.

Returns *list of point* – The 4 corners of the bounding polygon in the XY plane.

Examples

```
>>> mesh_bounding_box_xy(mesh)
[[0.0, 0.0, 0.0], [10.0, 0.0, 0.0], [10.0, 10.0, 0.0], [0.0, 10.0, 0.0]]
```

MeshPlus.centroid

MeshPlus.**centroid**()
Calculate the mesh centroid.

Returns *list* – The coordinates of the mesh centroid.

MeshPlus.clear

MeshPlus.**clear**()
Clear all the mesh data.

MeshPlus.collapse_edge

MeshPlus.**collapse_edge**(*u*, *v*, *t=0.5*, *allow_boundary=False*, *fixed=None*)
Collapse an edge to its first or second vertex, or to an intermediate point.

Parameters

- **mesh** (*compas.datastructures.Mesh*) – Instance of a mesh.
- **u** (*str*) – The first vertex of the (half-) edge.
- **v** (*str*) – The second vertex of the (half-) edge.
- **t** (*float (0.5)*) – Determines where to collapse to. If *t == 0.0* collapse to *u*. If *t == 1.0* collapse to *v*. If *0.0 < t < 1.0*, collapse to a point between *u* and *v*.
- **allow_boundary** (*bool (False)*) – Allow collapses involving boundary vertices.
- **fixed** (*list (None)*) – A list of identifiers of vertices that should stay fixed.

Returns *None*

Raises **ValueError** – If *u* and *v* are not neighbors.

MeshPlus.connected_components

MeshPlus.**connected_components**()

MeshPlus.copy

MeshPlus.**copy**(*cls=None*)

Make an independent copy of the datastructure object.

Parameters **cls** (compas.datastructure.Datastructure, optional) – The type of datastructure to return. Defaults to the type of the current datastructure.

Returns compas.datastructure.Datastructure – A separate, but identical datastructure object.

MeshPlus.cull_vertices

MeshPlus.**cull_vertices**()

Remove all unused vertices from the mesh object.

MeshPlus.cut

MeshPlus.**cut**(*plane*)

Cut a mesh with a plane and construct the resulting submeshes.

Parameters

- **mesh** (*compas.datastructures.Mesh*) – The original mesh.
- **plane** (*compas.geometry.Plane*) – The cutting plane.

Returns *None or tuple of compas.datastructures.Mesh* – If the mesh and plane do not intersect, or if the intersection is degenerate (point or line), the function returns None. Otherwise, the "positive" and "negative" submeshes are returned.

Examples

```
>>> from compas.geometry import Plane
>>> from compas.geometry import Box
>>> from compas.datastructures import Mesh
>>> plane = Plane((0, 0, 0), (1, 0, 0))
>>> box = Box.from_width_height_depth(1, 1, 1)
>>> mesh = Mesh.from_shape(box)
>>> result = mesh_cut_by_plane(mesh, plane)
>>> len(result) == 2
True
```

MeshPlus.delete_face

MeshPlus.**delete_face**(*fkey*)

Delete a face from the mesh object.

Parameters **fkey** (*int*) – The identifier of the face.

Notes

In some cases, disconnected vertices can remain after application of this method. To remove these vertices as well, combine this method with vertex culling (*cull_vertices()*).

Examples

```
>>>
```

MeshPlus.delete_vertex

MeshPlus.**delete_vertex**(*key*)

Delete a vertex from the mesh and everything that is attached to it.

Parameters **key** (*int*) – The identifier of the vertex.

Notes

In some cases, disconnected vertices can remain after application of this method. To remove these vertices as well, combine this method with vertex culling (*cull_vertices()*).

Examples

```
>>>
```

MeshPlus.dual

MeshPlus.**dual**(*cls=None*)

Construct the dual of a mesh.

Parameters

- **mesh** (*Mesh*) – A mesh object.
- **cls** (*Mesh, optional [None]*) – The type of the dual mesh. Defaults to the type of the provided mesh object.

Returns *Mesh* – The dual mesh object.

Examples

```
>>>
```

MeshPlus.edge_attribute

MeshPlus.**edge_attribute**(*edge*, *name*, *value=None*)

Get or set an attribute of an edge.

Parameters

- **edge** (*2-tuple of int*) – The identifier of the edge as a pair of vertex identifiers.
- **name** (*str*) – The name of the attribute.
- **value** (*obj, optional*) – The value of the attribute. Default is `None`.

Returns *object or None* – The value of the attribute, or `None` when the function is used as a "setter".

Raises **KeyError** – If the edge does not exist.

MeshPlus.edge_attributes

MeshPlus.**edge_attributes**(*edge*, *names=None*, *values=None*)

Get or set multiple attributes of an edge.

Parameters

- **edge** (*2-tuple of int*) – The identifier of the edge.
- **names** (*list, optional*) – A list of attribute names.
- **values** (*list, optional*) – A list of attribute values.

Returns *dict, list or None* – If the parameter `names` is empty, a dictionary of all attribute name-value pairs of the edge. If the parameter `names` is not empty, a list of the values corresponding to the provided names. `None` if the function is used as a "setter".

Raises **KeyError** – If the edge does not exist.

MeshPlus.edge_coordinates

MeshPlus.**edge_coordinates**(*u*, *v*, *axes='xyz'*)

Return the coordinates of the start and end point of an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.
- **axes** (*str (xyz)*) – The axes along which the coordinates should be included.

Returns *tuple* – The coordinates of the start point and the coordinates of the end point.

MeshPlus.edge_direction

MeshPlus.**edge_direction**(*u*, *v*)

Return the direction vector of an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.

Returns *list* – The direction vector of the edge.

MeshPlus.edge_faces

MeshPlus.**edge_faces**(*u*, *v*)

Find the two faces adjacent to an edge.

Parameters

- **u** (*int*) – The identifier of the first vertex.
- **v** (*int*) – The identifier of the second vertex.

Returns *tuple* – The identifiers of the adjacent faces. If the edge is on the boundary, one of the identifiers is `None`.

MeshPlus.edge_length

MeshPlus.**edge_length**(*u*, *v*)

Return the length of an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.

Returns *float* – The length of the edge.

MeshPlus.edge_loop

MeshPlus.**edge_loop**(*edge*)

Find all edges on the same loop as a given edge.

Parameters **edge** (*tuple of int*) – The identifier of the starting edge.

Returns *list of tuple of int* – The edges on the same loop as the given edge.

MeshPlus.edge_midpoint

MeshPlus.**edge_midpoint**(*u*, *v*)

Return the location of the midpoint of an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.

Returns *list* – The XYZ coordinates of the midpoint.

MeshPlus.edge_point

MeshPlus.**edge_point**(*u*, *v*, *t=0.5*)

Return the location of a point along an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.
- **t** (*float (0.5)*) – The location of the point on the edge. If the value of `t` is outside the range `0-1`, the point will lie in the direction of the edge, but not on the edge vector.

Returns *list* – The XYZ coordinates of the point.

MeshPlus.edge_strip

MeshPlus.**edge_strip**(*edge*)

Find all edges on the same strip as a given edge.

Parameters **edge** (*tuple of int*) – The identifier of the starting edge.

Returns *list of tuple of int* – The edges on the same strip as the given edge.

MeshPlus.edge_vector

MeshPlus.**edge_vector**(*u*, *v*)

Return the vector of an edge.

Parameters

- **u** (*int*) – The key of the start vertex.
- **v** (*int*) – The key of the end vertex.

Returns *list* – The vector from u to v.

MeshPlus.edges

MeshPlus.**edges**(*data=False*)

Iterate over the edges of the mesh.

Parameters **data** (*bool, optional*) – Return the edge data as well as the edge vertex keys.

Yields *tuple* – The next edge as a (u, v) tuple, if `data` is false. The next edge as a ((u, v), data) tuple, if `data` is true.

Notes

Mesh edges have no topological meaning. They are only used to store data. Edges are not automatically created when vertices and faces are added to the mesh. Instead, they are created when data is stored on them, or when they are accessed using this method.

This method yields the directed edges of the mesh. Unless edges were added explicitly using `add_edge()` the order of edges is *as they come out*. However, as long as the toplogy remains unchanged, the order is consistent.

Examples

```
>>>
```

MeshPlus.edges_attribute

MeshPlus.**edges_attribute**(*name*, *value=None*, *keys=None*)

Get or set an attribute of multiple edges.

Parameters

- **name** (*str*) – The name of the attribute.
- **value** (*obj, optional*) – The value of the attribute. Default is `None`.
- **keys** (*list of edges, optional*) – A list of edge identifiers.

Returns *list or None* – A list containing the value per edge of the requested attribute, or `None` if the function is used as a "setter".

Raises **KeyError** – If any of the edges does not exist.

MeshPlus.edges_attributes

MeshPlus.**edges_attributes**(*names=None*, *values=None*, *keys=None*)

Get or set multiple attributes of multiple edges.

Parameters

- **names** (*list of str, optional*) – The names of the attribute. Default is `None`.
- **values** (*list of obj, optional*) – The values of the attributes. Default is `None`.
- **keys** (*list of edges, optional*) – A list of edge identifiers.

Returns *dict, list or None* – If the parameter `names` is `None`, a list containing per edge an attribute dict with all attributes (default + custom) of the edge. If the parameter `names` is `None`, a list containing per edge a list of attribute values corresponding to the requested names. `None` if the function is used as a "setter".

Raises **`KeyError`** – If any of the edges does not exist.

MeshPlus.edges_on_boundaries

`MeshPlus.`**`edges_on_boundaries`**`()`

MeshPlus.edges_on_boundary

`MeshPlus.`**`edges_on_boundary`**`(`*oriented=False*`)`

Find the edges on the boundary.

Parameters **oriented** (*bool, optional*) – If `False` (default) the edges are aligned head-to-tail along the boundary. If `True` the edges have the same orientation as in the mesh.

Returns **edges** (*list*) – The boundary edges.

MeshPlus.edges_where

`MeshPlus.`**`edges_where`**`(`*conditions*, *data=False*`)`

Get edges for which a certain condition or set of conditions is true.

Parameters

- **conditions** (*dict*) – A set of conditions in the form of key-value pairs. The keys should be attribute names. The values can be attribute values or ranges of attribute values in the form of min/max pairs.
- **data** (*bool, optional*) – Yield the edges and their data attributes. Default is `False`.

Yields

- *2-tuple* – The next edge as a (u, v) tuple, if `data=False`.
- *3-tuple* – The next edge as a (u, v, data) tuple, if `data=True`.

MeshPlus.edges_where_predicate

`MeshPlus.`**`edges_where_predicate`**`(`*predicate*, *data=False*`)`

Get edges for which a certain condition or set of conditions is true using a lambda function.

Parameters

- **predicate** (*callable*) – The condition you want to evaluate. The callable takes 3 parameters: `u`, `v`, `attr` and should return `True` or `False`.
- **data** (*bool, optional*) – Yield the vertices and their data attributes. Default is `False`.

Yields

- *2-tuple* – The next edge as a (u, v) tuple, if `data=False`.
- *3-tuple* – The next edge as a (u, v, data) tuple, if `data=True`.

Examples

```
>>>
```

MeshPlus.euler

MeshPlus.**euler**()
Calculate the Euler characteristic.

Returns *int* – The Euler characteristic.

MeshPlus.face_adjacency

MeshPlus.**face_adjacency**()
Build a face adjacency dict.

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

Returns *dict* – A dictionary mapping face identifiers (keys) to lists of neighboring faces.

Notes

This algorithm is used primarily to unify the cycle directions of a given mesh. Therefore, the premise is that the topological information of the mesh is corrupt and cannot be used to construct the adjacency structure. The algorithm is thus purely geometrical, but uses a spatial indexing tree to speed up the search.

MeshPlus.face_adjacency_halfedge

MeshPlus.**face_adjacency_halfedge**(*f1*, *f2*)
Find one half-edge over which two faces are adjacent.

Parameters

- **f1** (*hashable*) – The identifier of the first face.
- **f2** (*hashable*) – The identifier of the second face.

Returns

- *tuple* – The half-edge separating face 1 from face 2.
- *None* – If the faces are not adjacent.

Notes

For use in form-finding algorithms, that rely on form-force duality information, further checks relating to the orientation of the corresponding are required.

MeshPlus.face_adjacency_vertices

MeshPlus.**face_adjacency_vertices**(*f1*, *f2*)
Find all vertices over which two faces are adjacent.

Parameters

- **f1** (*int*) – The identifier of the first face.
- **f2** (*int*) – The identifier of the second face.

Returns

- *list* – The vertices separating face 1 from face 2.
- *None* – If the faces are not adjacent.

MeshPlus.face_area

MeshPlus.**face_area**(*fkey*)
Compute the area of a face.

Parameters **fkey** (*int*) – The identifier of the face.

Returns *float* – The area of the face.

MeshPlus.face_aspect_ratio

MeshPlus.**face_aspect_ratio**(*fkey*)
Face aspect ratio as the ratio between the lengths of the maximum and minimum face edges.

Parameters **fkey** (*Key*) – The face key.

Returns *float* – The aspect ratio.

References

MeshPlus.face_attribute

MeshPlus.**face_attribute**(*key*, *name*, *value=None*)
Get or set an attribute of a face.

Parameters

- **key** (*int*) – The face identifier.
- **name** (*str*) – The name of the attribute.
- **value** (*obj, optional*) – The value of the attribute.

Returns *object or None* – The value of the attribute, or `None` when the function is used as a "setter".

Raises **KeyError** – If the face does not exist.

MeshPlus.face_attributes

MeshPlus.**face_attributes**(*key*, *names=None*, *values=None*)
Get or set multiple attributes of a face.

Parameters

- **key** (*int*) – The identifier of the face.
- **names** (*list, optional*) – A list of attribute names.
- **values** (*list, optional*) – A list of attribute values.

Returns *dict, list or None* – If the parameter `names` is empty, a dictionary of all attribute name-value pairs of the face. If the parameter `names` is not empty, a list of the values corresponding to the provided names. `None` if the function is used as a "setter".

Raises **KeyError** – If the face does not exist.

MeshPlus.face_center

MeshPlus.**face_center**(*fkey*)
Compute the location of the center of mass of a face.

Parameters **fkey** (*int*) – The identifier of the face.

Returns *list* – The coordinates of the center of mass.

MeshPlus.face_centroid

MeshPlus.**face_centroid**(*fkey*)
Compute the location of the centroid of a face.

Parameters **fkey** (*int*) – The identifier of the face.

Returns *list* – The coordinates of the centroid.

MeshPlus.face_coordinates

MeshPlus.**face_coordinates**(*fkey*, *axes='xyz'*)
Compute the coordinates of the vertices of a face.

Parameters

- **fkey** (*int*) – The identifier of the face.
- **axes** (*str, optional*) – The axes along which to take the coordinates. Should be a combination of `'x'`, `'y'`, `'z'`. Default is `'xyz'`.

Returns *list of list* – The coordinates of the vertices of the face.

MeshPlus.face_corners

MeshPlus.**face_corners**(*fkey*)

Return triplets of face vertices forming the corners of the face.

Parameters **fkey** (*int*) – Identifier of the face.

Returns *list* – The corners of the face in the form of a list of vertex triplets.

MeshPlus.face_curvature

MeshPlus.**face_curvature**(*fkey*)

Dimensionless face curvature as the maximum face vertex deviation from the best-fit plane of the face vertices divided by the average lengths of the face vertices to the face centroid.

Parameters **fkey** (*Key*) – The face key.

Returns *float* – The dimensionless curvature.

MeshPlus.face_degree

MeshPlus.**face_degree**(*fkey*)

Count the neighbors of a face.

Parameters **fkey** (*int*) – Identifier of the face.

Returns *int* – The count.

MeshPlus.face_flatness

MeshPlus.**face_flatness**(*fkey*, *maxdev=0.02*)

Compute the flatness of the mesh face.

Parameters

- **fkey** (*int*) – The identifier of the face.
- **maxdev** (*float, optional*) – A maximum value for the allowed deviation from flatness. Default is `0.02`.

Returns *float* – The flatness.

Notes

Flatness is computed as the ratio of the distance between the diagonals of the face to the average edge length. A practical limit on this value realted to manufacturing is 0.02 (2%).

Warning: This method only makes sense for quadrilateral faces.

MeshPlus.face_halfedges

MeshPlus.**face_halfedges**(*fkey*)
The halfedges of a face.

Parameters **fkey** (*int*) – Identifier of the face.

Returns *list* – The halfedges of a face.

MeshPlus.face_max_degree

MeshPlus.**face_max_degree**()
Compute the maximum degree of all faces.

Returns *int* – The highest degree.

MeshPlus.face_min_degree

MeshPlus.**face_min_degree**()
Compute the minimum degree of all faces.

Returns *int* – The lowest degree.

MeshPlus.face_neighborhood

MeshPlus.**face_neighborhood**(*key*, *ring=1*)
Return the faces in the neighborhood of a face.

Parameters

- **key** (*int*) – The identifier of the face.
- **ring** (*int, optional*) – The size of the neighborhood. Default is `1`.

Returns *list* – A list of face identifiers.

MeshPlus.face_neighbors

MeshPlus.**face_neighbors**(*fkey*)
Return the neighbors of a face across its edges.

Parameters **fkey** (*int*) – Identifier of the face.

Returns *list* – The identifiers of the neighboring faces.

Examples

```
>>>
```

MeshPlus.face_normal

MeshPlus.**face_normal**(*fkey*, *unitized=True*)

Compute the normal of a face.

Parameters

- **fkey** (*int*) – The identifier of the face.
- **unitized** (*bool, optional*) – Unitize the normal vector. Default is `True`.

Returns *list* – The components of the normal vector.

MeshPlus.face_plane

MeshPlus.**face_plane**(*face*)

A plane defined by the centroid and the normal of the face.

Parameters **face** (*int*) – The face identifier.

Returns *tuple* – point, vector

MeshPlus.face_skewness

MeshPlus.**face_skewness**(*fkey*)

Face skewness as the maximum absolute angular deviation from the ideal polygon angle.

Parameters **fkey** (*Key*) – The face key.

Returns *float* – The skewness.

References

MeshPlus.face_vertex_ancestor

MeshPlus.**face_vertex_ancestor**(*fkey*, *key*, *n=1*)

Return the n-th vertex before the specified vertex in a specific face.

Parameters

- **fkey** (*int*) – Identifier of the face.
- **key** (*int*) – The identifier of the vertex.
- **n** (*int, optional*) – The index of the vertex ancestor. Default is 1, meaning the previous vertex.

Returns *int* – The identifier of the vertex before the given vertex in the face cycle.

Raises **ValueError** – If the vertex is not part of the face.

MeshPlus.face_vertex_descendant

MeshPlus.**face_vertex_descendant**(*fkey*, *key*, *n=1*)

Return the n-th vertex after the specified vertex in a specific face.

Parameters

- **fkey** (*int*) – Identifier of the face.
- **key** (*int*) – The identifier of the vertex.
- **n** (*int, optional*) – The index of the vertex descendant. Default is 1, meaning the next vertex.

Returns *int* – The identifier of the vertex after the given vertex in the face cycle.

Raises **ValueError** – If the vertex is not part of the face.

MeshPlus.face_vertices

MeshPlus.**face_vertices**(*fkey*)

The vertices of a face.

Parameters **fkey** (*int*) – Identifier of the face.

Returns *list* – Ordered vertex identifiers.

MeshPlus.faces

MeshPlus.**faces**(*data=False*)

Iterate over the faces of the mesh.

Parameters **data** (*bool, optional*) – Return the face data as well as the face keys.

Yields *int or tuple* – The next face identifier, if `data` is `False`. The next face as a (fkey, attr) tuple, if `data` is `True`.

MeshPlus.faces_attribute

MeshPlus.**faces_attribute**(*name*, *value=None*, *keys=None*)

Get or set an attribute of multiple faces.

Parameters

- **name** (*str*) – The name of the attribute.
- **value** (*obj, optional*) – The value of the attribute. Default is `None`.
- **keys** (*list of int, optional*) – A list of face identifiers.

Returns *list or None* – A list containing the value per face of the requested attribute, or `None` if the function is used as a "setter".

Raises **KeyError** – If any of the faces does not exist.

MeshPlus.faces_attributes

MeshPlus.**faces_attributes**(*names=None*, *values=None*, *keys=None*)

Get or set multiple attributes of multiple faces.

Parameters

- **names** (*list of str, optional*) – The names of the attribute. Default is `None`.
- **values** (*list of obj, optional*) – The values of the attributes. Default is `None`.
- **keys** (*list of int, optional*) – A list of face identifiers.

Returns *dict, list or None* – If the parameter `names` is `None`, a list containing per face an attribute dict with all attributes (default + custom) of the face. If the parameter `names` is `None`, a list containing per face a list of attribute values corresponding to the requested names. `None` if the function is used as a "setter".

Raises **KeyError** – If any of the faces does not exist.

MeshPlus.faces_on_boundary

MeshPlus.**faces_on_boundary**()

Find the faces on the boundary.

Returns *list* – The faces on the boundary.

MeshPlus.faces_where

MeshPlus.**faces_where**(*conditions*, *data=False*)

Get faces for which a certain condition or set of conditions is true.

Parameters

- **conditions** (*dict*) – A set of conditions in the form of key-value pairs. The keys should be attribute names. The values can be attribute values or ranges of attribute values in the form of min/max pairs.
- **data** (*bool, optional*) – Yield the faces and their data attributes. Default is `False`.

Yields

- **key** (*hashable*) – The next face that matches the condition.
- *2-tuple* – The next face and its attributes, if `data=True`.

MeshPlus.faces_where_predicate

MeshPlus.**faces_where_predicate**(*predicate*, *data=False*)

Get faces for which a certain condition or set of conditions is true using a lambda function.

Parameters

- **predicate** (*callable*) – The condition you want to evaluate. The callable takes 2 parameters: `key`, `attr` and should return `True` or `False`.
- **data** (*bool, optional*) – Yield the faces and their data attributes. Default is `False`.

Yields

- **key** (*hashable*) – The next face that matches the condition.
- *2-tuple* – The next face and its attributes, if `data=True`.

Examples

```
>>>
```

MeshPlus.flip_cycles

`MeshPlus.`**`flip_cycles`**`()`

Flip the cycle directions of all faces.

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

Notes

This function does not care about the directions being unified or not. It just reverses whatever direction it finds.

MeshPlus.from_data

`classmethod` `MeshPlus.`**`from_data`**`(`*data*`)`

Construct a datastructure from structured data.

Parameters **data** (*dict*) – The data dictionary.

Returns `compas.datastructures.Datastructure` – An object of the type of `cls`.

Notes

This constructor method is meant to be used in conjunction with the corresponding *to_data* method.

MeshPlus.from_json

`classmethod` `MeshPlus.`**`from_json`**`(`*filepath*`)`

Construct a datastructure from structured data contained in a json file.

Parameters **filepath** (*str*) – The path to the json file.

Returns `compas.datastructures.Datastructure` – An object of the type of `cls`.

Notes

This constructor method is meant to be used in conjunction with the corresponding *to_json* method.

MeshPlus.from_lines

classmethod MeshPlus.**from_lines**(*lines*, *delete_boundary_face=False*, *precision=None*)

Construct a mesh object from a list of lines described by start and end point coordinates.

Parameters

- **lines** (*list*) – A list of pairs of point coordinates.
- **delete_boundary_face** (*bool, optional*) – The algorithm that finds the faces formed by the connected lines first finds the face *on the outside*. In most cases this face is not expected to be there. Therefore, there is the option to have it automatically deleted.
- **precision** (*str, optional*) – The precision of the geometric map that is used to connect the lines.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_obj

classmethod MeshPlus.**from_obj**(*filepath*, *precision=None*)

Construct a mesh object from the data described in an OBJ file.

Parameters

- **filepath** (*str*) – The path to the file.
- **precision** (*str, optional*) – The precision of the geometric map that is used to connect the lines.

Returns *Mesh* – A mesh object.

Notes

There are a few sample files available for testing and debugging:

- faces.obj
- faces_big.obj
- faces_reversed.obj
- hypar.obj
- mesh.obj
- quadmesh.obj

Examples

```
>>>
```

MeshPlus.from_off

classmethod MeshPlus.**from_off**(*filepath*)

Construct a mesh object from the data described in a OFF file.

Parameters **filepath** (*str*) – The path to the file.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_ply

classmethod MeshPlus.**from_ply**(*filepath*, *precision=None*)

Construct a mesh object from the data described in a PLY file.

Parameters **filepath** (*str*) – The path to the file.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_points

classmethod MeshPlus.**from_points**(*points*, *boundary=None*, *holes=None*)

Construct a mesh from a delaunay triangulation of a set of points.

Parameters **points** (*list*) – XYZ coordinates of the points. Z coordinates should be zero.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_polygons

classmethod MeshPlus.**from_polygons**(*polygons*, *precision=None*)

Construct a mesh from a series of polygons.

Parameters

- **polygons** (*list*) – A list of polygons, with each polygon defined as an ordered list of XYZ coordinates of its corners.
- **precision** (*str, optional*) – The precision of the geometric map that is used to connect the lines.

Returns *Mesh* – A mesh object.

MeshPlus.from_polyhedron

classmethod MeshPlus.**from_polyhedron**(*f*)

Construct a mesh from a platonic solid.

Parameters **f** (*int*) – The number of faces. Should be one of `4, 6, 8, 12, 20`.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_polylines

classmethod MeshPlus.**from_polylines**(*boundary_polylines*, *other_polylines*)

Construct mesh from polylines.

Based on construction from_lines, with removal of vertices that are not polyline extremities and of faces that represent boundaries.

This specific method is useful to get the mesh connectivity from a set of (discretised) curves, that could overlap and yield a wrong connectivity if using from_lines based on the polyline extremities only.

Parameters

- **boundary_polylines** (*list*) – List of polylines representing boundaries as lists of vertex coordinates.
- **other_polylines** (*list*) – List of the other polylines as lists of vertex coordinates.

Returns *Mesh* – A mesh object.

MeshPlus.from_shape

classmethod MeshPlus.**from_shape**(*shape*, ***kwargs*)

Construct a mesh from a primitive shape.

Parameters

- **shape** (:class: *compas.geometry.shape*) – The input shape to generate a mesh from.
- **kwargs** – Optional keyword arguments `u` and `v` for the resolution in u (Torus, Sphere, Cylinder, Cone) and v direction (Torus and Sphere).

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_stl

classmethod MeshPlus.**from_stl**(*filepath*, *precision=None*)

Construct a mesh object from the data described in a STL file.

Parameters

- **filepath** (*str*) – The path to the file.
- **precision** (*str, optional*) – The precision of the geometric map that is used to connect the lines.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.from_vertices_and_faces

classmethod MeshPlus.**from_vertices_and_faces**(*vertices*, *faces*)

Construct a mesh object from a list of vertices and faces.

Parameters

- **vertices** (*list, dict*) – A list of vertices, represented by their XYZ coordinates, or a dictionary of vertex keys pointing to their XYZ coordinates.
- **faces** (*list, dict*) – A list of faces, represented by a list of indices referencing the list of vertex coordinates, or a dictionary of face keys pointing to a list of indices referencing the list of vertex coordinates.

Returns *Mesh* – A mesh object.

Examples

```
>>>
```

MeshPlus.genus

`MeshPlus.genus()`
Calculate the genus.

Returns *int* – The genus.

References

MeshPlus.get_any_face

`MeshPlus.get_any_face()`
Get the identifier of a random face.

Returns *hashable* – The identifier of the face.

MeshPlus.get_any_face_vertex

`MeshPlus.get_any_face_vertex(fkey)`
Get the identifier of a random vertex of a specific face.

Parameters **fkey** (*hashable*) – The identifier of the face.

Returns *hashable* – The identifier of the vertex.

MeshPlus.get_any_vertex

`MeshPlus.get_any_vertex()`
Get the identifier of a random vertex.

Returns *hashable* – The identifier of the vertex.

MeshPlus.get_any_vertices

`MeshPlus.get_any_vertices(n, exclude_leaves=False)`
Get a list of identifiers of a random set of n vertices.

Parameters

- **n** (*int*) – The number of random vertices.
- **exclude_leaves** (*bool (False)*) – Exclude the leaves (vertices with only one connected edge) from the set. Default is to include the leaves.

Returns *list* – The identifiers of the vertices.

MeshPlus.gkey_key

MeshPlus.**gkey_key**(*precision=None*)

Returns a dictionary that maps *geometric keys* of a certain precision to the keys of the corresponding vertices.

Parameters **precision** (*str (3f)*) – The float precision specifier used in string formatting.

Returns *dict* – A dictionary of geometric key-key pairs.

MeshPlus.halfedge_face

MeshPlus.**halfedge_face**(*u*, *v*)

Find the face corresponding to a halfedge.

Parameters

- **u** (*int*) – The identifier of the first vertex.
- **v** (*int*) – The identifier of the second vertex.

Returns *int or None* – The identifier of the face corresponding to the halfedge. None, if the halfedge is on the outside of a boundary.

Raises **KeyError** – If the halfedge does not exist.

Examples

```
>>>
```

MeshPlus.has_edge

MeshPlus.**has_edge**(*key*)

Verify that the mesh contains a specific edge.

Warning: This method may produce unexpected results.

Parameters **key** (*tuple of int*) – The identifier of the edge.

Returns *bool* – True if the edge exists. False otherwise.

MeshPlus.has_face

MeshPlus.**has_face**(*fkey*)

Verify that a face is part of the mesh.

Parameters **fkey** (*int*) – The identifier of the face.

Returns *bool* – True if the face exists. False otherwise.

Examples

```
>>>
```

MeshPlus.has_halfedge

MeshPlus.**has_halfedge**(*key*)
Verify that a halfedge is part of the mesh.

Parameters **key** (*tuple of int*) – The identifier of the halfedge.

Returns *bool* – True if the halfedge is part of the mesh. False otherwise.

MeshPlus.has_vertex

MeshPlus.**has_vertex**(*key*)
Verify that a vertex is in the mesh.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *bool* – True if the vertex is in the mesh. False otherwise.

MeshPlus.index_key

MeshPlus.**index_key**()
Returns a dictionary that maps the indices of a vertex list to keys in a vertex dictionary.

Returns *dict* – A dictionary of index-key pairs.

MeshPlus.index_vertex

MeshPlus.**index_vertex**()
Returns a dictionary that maps the indices of a vertex list to keys in a vertex dictionary.

Returns *dict* – A dictionary of index-key pairs.

MeshPlus.insert_vertex

MeshPlus.**insert_vertex**(*fkey*, *key=None*, *xyz=None*, *return_fkeys=False*)
Insert a vertex in the specified face.

Parameters

- **fkey** (*int*) – The key of the face in which the vertex should be inserted.
- **key** (*int, optional*) – The key to be used to identify the inserted vertex.
- **xyz** (*list, optional*) – Specific XYZ coordinates for the inserted vertex.
- **return_fkeys** (*bool, optional*) – By default, this method returns only the key of the inserted vertex. This flag can be used to indicate that the keys of the newly created faces should be returned as well.

Returns

- *int* – The key of the inserted vertex, if `return_fkeys` is false.
- *tuple* – The key of the newly created vertex and a list with the newly created faces, if `return_fkeys` is true.

Examples

```
>>>
```

MeshPlus.is_connected

`MeshPlus.`**`is_connected`**`()`

Verify that the mesh is connected.

Parameters **mesh** (*compas.datastructures.Mesh*) – A mesh data structure.

Returns *bool* – True, if the mesh is connected. False, otherwise.

Notes

A mesh is connected if for every two vertices a path exists connecting them.

Examples

```
>>> mesh_is_connected(m1)
True
>>> mesh_is_connected(m2)
True
>>> mesh_is_connected(m3)
False
```

MeshPlus.is_edge_on_boundary

`MeshPlus.`**`is_edge_on_boundary`**`(`*u*, *v*`)`

Verify that an edge is on the boundary.

Parameters

- **u** (*int*) – The identifier of the first vertex.
- **v** (*int*) – The identifier of the second vertex.

Returns *bool* – True if the edge is on the boundary. False otherwise.

MeshPlus.is_empty

MeshPlus.**is_empty**()
Boolean whether the mesh is empty.

Returns *bool* – True if no vertices. False otherwise.

MeshPlus.is_face_on_boundary

MeshPlus.**is_face_on_boundary**(*key*)
Verify that a face is on a boundary.

Parameters **key** (*int*) – The identifier of the face.

Returns *bool* – True if the face is on the boundary. False otherwise.

MeshPlus.is_manifold

MeshPlus.**is_manifold**()
Verify that the mesh is manifold.

A mesh is manifold if the following conditions are fulfilled:

- Each edge is incident to only one or two faces.
- The faces incident to a vertex form a closed or an open fan.

Returns *bool* – True, if the mesh is manifold. False, otherwise.

MeshPlus.is_orientable

MeshPlus.**is_orientable**()
Verify that the mesh is orientable.

A manifold mesh is orientable if the following conditions are fulfilled:

- Any two adjacent faces have compatible orientation, i.e. the faces have a unified cycle direction.

Returns *bool* – True, if the mesh is orientable. False, otherwise.

MeshPlus.is_quadmesh

MeshPlus.**is_quadmesh**()
Verify that the mesh consists of only quads.

Returns *bool* – True, if the mesh is a quad mesh. False, otherwise.

MeshPlus.is_regular

MeshPlus.**is_regular**()
Verify that the mesh is regular.

A mesh is regular if the following conditions are fulfilled:

- All faces have the same number of edges.
- All vertices have the same degree, i.e. they are incident to the same number of edges.

Returns *bool* – True, if the mesh is regular. False, otherwise.

MeshPlus.is_trimesh

MeshPlus.**is_trimesh**()
Verify that the mesh consists of only triangles.

Returns *bool* – True, if the mesh is a triangle mesh. False, otherwise.

MeshPlus.is_valid

MeshPlus.**is_valid**()
Verify that the mesh is valid.

A mesh is valid if the following conditions are fulfilled:

- halfedges don't point at non-existing faces
- all vertices are in the halfedge dict
- there are no None-None halfedges
- all faces have corresponding halfedge entries

Returns *bool* – True, if the mesh is valid. False, otherwise.

MeshPlus.is_vertex_connected

MeshPlus.**is_vertex_connected**(*key*)
Verify that a vertex is connected.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *bool* – True if the vertex is connected to at least one other vertex. False otherwise.

MeshPlus.is_vertex_on_boundary

MeshPlus.**is_vertex_on_boundary**(*key*)

Verify that a vertex is on a boundary.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *bool* – True if the vertex is on the boundary. False otherwise.

MeshPlus.join

MeshPlus.**join**(*other*)

Add the vertices and faces of another mesh to the current mesh.

Parameters **other** (*compas.datastructures.Mesh*) – The other mesh.

Returns *None* – The mesh is modified in place.

Examples

```
>>> from compas.geometry import Box
>>> from compas.geometry import Translation
>>> from compas.datastructures import Mesh
>>> a = Box.from_width_height_depth(1, 1, 1)
>>> b = Box.from_width_height_depth(1, 1, 1)
>>> T = Translation([2, 0, 0])
>>> b.transform(T)
>>> a = Mesh.from_shape(a)
>>> b = Mesh.from_shape(b)
>>> a.number_of_vertices()
8
>>> a.number_of_faces()
6
>>> b.number_of_vertices()
8
>>> b.number_of_faces()
6
>>> a.join(b)
>>> a.number_of_vertices()
16
>>> a.number_of_faces()
12
```

MeshPlus.key_gkey

MeshPlus.**key_gkey**(*precision=None*)

Returns a dictionary that maps vertex dictionary keys to the corresponding *geometric key* up to a certain precision.

Parameters **precision** (*str (3f)*) – The float precision specifier used in string formatting.

Returns *dict* – A dictionary of key-geometric key pairs.

MeshPlus.key_index

MeshPlus.**key_index**()
Returns a dictionary that maps vertex dictionary keys to the corresponding index in a vertex list or array.

Returns *dict* – A dictionary of key-index pairs.

MeshPlus.normal

MeshPlus.**normal**()
Calculate the average mesh normal.

Returns *list* – The coordinates of the mesh normal.

MeshPlus.number_of_edges

MeshPlus.**number_of_edges**()
Count the number of edges in the mesh.

MeshPlus.number_of_faces

MeshPlus.**number_of_faces**()
Count the number of faces in the mesh.

MeshPlus.number_of_vertices

MeshPlus.**number_of_vertices**()
Count the number of vertices in the mesh.

MeshPlus.quads_to_triangles

MeshPlus.**quads_to_triangles**(*check_angles=False*)

MeshPlus.remove_unused_vertices

MeshPlus.**remove_unused_vertices**()
Remove all unused vertices from the mesh object.

MeshPlus.smooth_area

MeshPlus.**smooth_area**(*fixed=None*, *kmax=100*, *damping=0.5*, *callback=None*, *callback_args=None*)
Smooth a mesh by moving each vertex to the barycenter of the centroids of the surrounding faces, weighted by area.

Parameters

- **mesh** (*Mesh*) – A mesh object.

- **fixed** (*list, optional*) – The fixed vertices of the mesh.
- **kmax** (*int, optional*) – The maximum number of iterations.
- **damping** (*float, optional*) – The damping factor.
- **callback** (*callable, optional*) – A user-defined callback function to be executed after every iteration.
- **callback_args** (*list, optional*) – A list of arguments to be passed to the callback.

Raises **Exception** – If a callback is provided, but it is not callable.

Examples

```
>>>
```

MeshPlus.smooth_centroid

MeshPlus.**smooth_centroid**(*fixed=None*, *kmax=100*, *damping=0.5*, *callback=None*, *callback_args=None*)

Smooth a mesh by moving every free vertex to the centroid of its neighbors.

Parameters

- **mesh** (*Mesh*) – A mesh object.
- **fixed** (*list, optional*) – The fixed vertices of the mesh.
- **kmax** (*int, optional*) – The maximum number of iterations.
- **damping** (*float, optional*) – The damping factor.
- **callback** (*callable, optional*) – A user-defined callback function to be executed after every iteration.
- **callback_args** (*list, optional*) – A list of arguments to be passed to the callback.

Raises **Exception** – If a callback is provided, but it is not callable.

Examples

```
>>>
```

MeshPlus.split_edge

MeshPlus.**split_edge**(*u*, *v*, *t=0.5*, *allow_boundary=False*)

Split and edge by inserting a vertex along its length.

Parameters

- **mesh** (*compas.datastructures.Mesh*) – Instance of a mesh.
- **u** (*str*) – The key of the first vertex of the edge.
- **v** (*str*) – The key of the second vertex of the edge.

- **t** (*float (0.5)*) – The position of the inserted vertex. The value should be between 0.0 and 1.0
- **allow_boundary** (*bool (False)*) – Split edges on the boundary.

Returns *int* – The key of the inserted vertex.

Raises **ValueError** – If u and v are not neighbors.

MeshPlus.split_face

MeshPlus.**split_face**(*fkey*, *u*, *v*)

Split a face by inserting an edge between two specified vertices.

Parameters

- **mesh** (Mesh) – Instance of a mesh
- **fkey** (str) – The face key.
- **u** (*hashable*) – The key of the first split vertex.
- **v** (*hashable*) – The key of the second split vertex.

Returns tuple of int – Keys of the created faces.

Raises **ValueError** – If the split vertices does not belong to the split face or if the split vertices are neighbors.

Examples

```
>>> import compas
>>> from compas.datastructures import Mesh
>>> mesh = Mesh.from_obj(compas.get("faces.obj"))
>>> fkey = mesh.get_any_face()
>>> # u and v defines the new edge after splitting
>>> u = mesh.get_any_face_vertex(fkey)
>>> v = mesh.face_vertex_descendant(fkey, u, n=2)
>>> mesh.number_of_faces()  # faces before split
25
>>> mesh_split_face(mesh, fkey, u, v)
(25, 26)
>>> mesh.number_of_faces()  # faces after split
26
```

MeshPlus.summary

MeshPlus.**summary**()

Print a summary of the mesh.

Returns *str*

MeshPlus.to_data

MeshPlus.**to_data**()
Returns a dictionary of structured data representing the data structure.

Returns *dict* – The structured data.

Notes

This method produces the data that can be used in conjunction with the corresponding *from_data* class method.

MeshPlus.to_json

MeshPlus.**to_json**(*filepath*, *pretty=False*)
Serialise the structured data representing the datastructure to json.

Parameters **filepath** (*str*) – The path to the json file.

MeshPlus.to_lines

MeshPlus.**to_lines**(*filepath*)

MeshPlus.to_obj

MeshPlus.**to_obj**(*filepath*, *precision=None*, *unweld=False*, ***kwargs*)
Write the mesh to an OBJ file.

Parameters

- **filepath** (*str*) – Full path of the file.
- **precision** (*str, optional*) – The precision of the geometric map that is used to connect the lines.
- **unweld** (*bool, optional*) – If true, all faces have their own unique vertices. If false, vertices are shared between faces if this is also the case in the mesh. Default is `False`.

Warning: This function only writes geometric data about the vertices and the faces to the file.

MeshPlus.to_off

MeshPlus.**to_off**(*filepath*, ***kwargs*)
Write a mesh object to an OFF file.

Parameters **filepath** (*str*) – The path to the file.

Examples

```
>>>
```

MeshPlus.to_ply

`MeshPlus.`**`to_ply`**(*filepath*, ***kwargs*)

Write a mesh object to a PLY file.

Parameters **filepath** (*str*) – The path to the file.

Examples

```
>>>
```

MeshPlus.to_points

`MeshPlus.`**`to_points`**()

MeshPlus.to_polygons

`MeshPlus.`**`to_polygons`**()

MeshPlus.to_polylines

`MeshPlus.`**`to_polylines`**()

MeshPlus.to_quadmesh

`MeshPlus.`**`to_quadmesh`**()

MeshPlus.to_stl

`MeshPlus.`**`to_stl`**(*filepath*, *precision=None*, *binary=False*, ***kwargs*)

Write a mesh to an STL file.

Parameters

- **filepath** (*str*) – The path to the file.
- **precision** (*str, optional*) – Rounding precision for the vertex coordinates. Default is `"3f"`.
- **binary** (*bool, optional*) – When `False`, the file will be written in ASCII encoding, when `True`, binary. Default is `False`.

Returns *None*

Notes

STL files only support triangle faces. It is your responsibility to convert all faces of your mesh to triangles. For example, with `compas.datastructures.mesh_quads_to_triangles()`.

MeshPlus.to_trimesh

`MeshPlus.`**`to_trimesh`**`()`

MeshPlus.to_vertices_and_faces

`MeshPlus.`**`to_vertices_and_faces`**`()`

Return the vertices and faces of a mesh.

Returns

tuple – A 2-tuple containing

- a list of vertices, represented by their XYZ coordinates, and
- a list of faces.

Each face is a list of indices referencing the list of vertex coordinates.

Examples

```
>>>
```

MeshPlus.transform

`MeshPlus.`**`transform`**`(`*transformation*`)`

Transform a mesh.

Parameters

- **mesh** (*compas.datastructures.Mesh*) – The mesh.
- **transformation** (*compas.geometry.Transformation*) – The transformation.

Notes

The mesh is modified in-place.

Examples

```
>>> mesh = Mesh.from_obj(compas.get('cube.obj'))
>>> T = matrix_from_axis_and_angle([0, 0, 1], pi / 4)
>>> tmesh = mesh.copy()
>>> mesh_transform(tmesh, T)
```

MeshPlus.transform_numpy

MeshPlus.**transform_numpy**(*M*)

MeshPlus.transformed

MeshPlus.**transformed**(*transformation*)

Transform a copy of `mesh`.

Parameters

- **mesh** (*compas.datastructures.Mesh*) – The mesh.
- **transformation** (*compas.geometry.Transformation*) – The transformation.

Returns *Mesh* – A transformed independent copy of `mesh`.

Notes

The original mesh is not modified. Instead a transformed independent copy is returned.

Examples

```
>>> mesh = Mesh.from_obj(compas.get('cube.obj'))
>>> T = matrix_from_axis_and_angle([0, 0, 1], pi / 4)
>>> tmesh = mesh_transformed(mesh, T)
```

MeshPlus.unify_cycles

MeshPlus.**unify_cycles**(*root=None*)

Unify the cycle directions of all faces.

Unified cycle directions is a necessary condition for the data structure to work properly. When in doubt, run this function on your mesh.

Parameters

- **mesh** (*Mesh*) – A mesh object.
- **root** (*str, optional [None]*) – The key of the root face.

MeshPlus.unset_edge_attribute

MeshPlus.**unset_edge_attribute**(*edge*, *name*)
Unset the attribute of an edge.

Parameters

- **edge** (*tuple of int*) – The edge identifier.
- **name** (*str*) – The name of the attribute.

Raises **KeyError** – If the edge does not exist.

Notes

Unsetting the value of an edge attribute implicitly sets it back to the value stored in the default edge attribute dict.

MeshPlus.unset_face_attribute

MeshPlus.**unset_face_attribute**(*key*, *name*)
Unset the attribute of a face.

Parameters

- **key** (*int*) – The face identifier.
- **name** (*str*) – The name of the attribute.

Raises **KeyError** – If the face does not exist.

Notes

Unsetting the value of a face attribute implicitly sets it back to the value stored in the default face attribute dict.

MeshPlus.unset_vertex_attribute

MeshPlus.**unset_vertex_attribute**(*key*, *name*)
Unset the attribute of a vertex.

Parameters

- **key** (*int*) – The vertex identifier.
- **name** (*str*) – The name of the attribute.

Raises **KeyError** – If the vertex does not exist.

Notes

Unsetting the value of a vertex attribute implicitly sets it back to the value stored in the default vertex attribute dict.

MeshPlus.update_default_edge_attributes

MeshPlus.**update_default_edge_attributes**(*attr_dict=None*, ***kwattr*)
Update the default edge attributes.

Parameters

- **attr_dict** (*dict, optional*) – A dictionary of attributes with their default values. Defaults to an empty `dict`.
- **kwattr** (*dict*) – A dictionary compiled of remaining named arguments. Defaults to an empty dict.

Notes

Named arguments overwrite corresponding key-value pairs in the attribute dictionary, if they exist.

MeshPlus.update_default_face_attributes

MeshPlus.**update_default_face_attributes**(*attr_dict=None*, ***kwattr*)
Update the default face attributes.

Parameters

- **attr_dict** (*dict (None)*) – A dictionary of attributes with their default values.
- **kwattr** (*dict*) – A dictionary compiled of remaining named arguments. Defaults to an empty dict.

Notes

Named arguments overwrite corresponding key-value pairs in the attribute dictionary, if they exist.

MeshPlus.update_default_vertex_attributes

MeshPlus.**update_default_vertex_attributes**(*attr_dict=None*, ***kwattr*)
Update the default vertex attributes.

Parameters

- **attr_dict** (*dict, optional*) – A dictionary of attributes with their default values. Defaults to an empty `dict`.
- **kwattr** (*dict*) – A dictionary compiled of remaining named arguments. Defaults to an empty dict.

Notes

Named arguments overwrite corresponding key-value pairs in the attribute dictionary, if they exist.

MeshPlus.validate_data

MeshPlus.**validate_data**()
Validate the data of this object against its data schema (*self.DATASCHEMA*).

Returns *dict* – The validated data.

Raises **SchemaError** –

MeshPlus.validate_json

MeshPlus.**validate_json**()
Validate the data loaded from a JSON representation of the data of this object against its data schema (*self.DATASCHEMA*).

Returns *None*

Raises **SchemaError** –

MeshPlus.vertex_area

MeshPlus.**vertex_area**(*key*)
Compute the tributary area of a vertex.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *float* – The tributary are.

Examples

```
>>>
```

MeshPlus.vertex_attribute

MeshPlus.**vertex_attribute**(*key*, *name*, *value=None*)
Get or set an attribute of a vertex.

Parameters

- **key** (*int*) – The vertex identifier.
- **name** (*str*) – The name of the attribute
- **value** (*obj, optional*) – The value of the attribute.

Returns *object or None* – The value of the attribute, or `None` if the vertex does not exist or when the function is used as a "setter".

Raises **KeyError** – If the vertex does not exist.

MeshPlus.vertex_attributes

`MeshPlus.`**`vertex_attributes`**(*key*, *names=None*, *values=None*)

Get or set multiple attributes of a vertex.

Parameters

- **key** (*int*) – The identifier of the vertex.
- **names** (*list, optional*) – A list of attribute names.
- **values** (*list, optional*) – A list of attribute values.

Returns *dict, list or None* – If the parameter `names` is empty, the function returns a dictionary of all attribute name-value pairs of the vertex. If the parameter `names` is not empty, the function returns a list of the values corresponding to the requested attribute names. The function returns `None` if it is used as a "setter".

Raises **`KeyError`** – If the vertex does not exist.

MeshPlus.vertex_coordinates

`MeshPlus.`**`vertex_coordinates`**(*key*, *axes='xyz'*)

Return the coordinates of a vertex.

Parameters

- **key** (*int*) – The identifier of the vertex.
- **axes** (*str, optional*) – The axes along which to take the coordinates. Should be a combination of `'x'`, `'y'`, `'z'`. Default is `'xyz'`.

Returns *list* – Coordinates of the vertex.

MeshPlus.vertex_curvature

`MeshPlus.`**`vertex_curvature`**(*vkey*)

Dimensionless vertex curvature.

Parameters **fkey** (*int*) – The face key.

Returns *float* – The dimensionless curvature.

References

Based on[1]

[1] Botsch, Mario, et al. *Polygon mesh processing.* AK Peters/CRC Press, 2010.

MeshPlus.vertex_degree

MeshPlus.**vertex_degree**(*key*)
Count the neighbors of a vertex.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *int* – The degree of the vertex.

MeshPlus.vertex_faces

MeshPlus.**vertex_faces**(*key*, *ordered=False*, *include_none=False*)
The faces connected to a vertex.

Parameters

- **key** (*int*) – The identifier of the vertex.
- **ordered** (*bool, optional*) – Return the faces in cycling order. Default is `False`.
- **include_none** (*bool, optional*) – Include *outside* faces in the list. Default is `False`.

Returns *list* – The faces connected to a vertex.

Examples

```
>>>
```

MeshPlus.vertex_index

MeshPlus.**vertex_index**()
Returns a dictionary that maps vertex dictionary keys to the corresponding index in a vertex list or array.

Returns *dict* – A dictionary of key-index pairs.

MeshPlus.vertex_laplacian

MeshPlus.**vertex_laplacian**(*key*)
Compute the vector from a vertex to the centroid of its neighbors.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *list* – The components of the vector.

MeshPlus.vertex_max_degree

MeshPlus.**vertex_max_degree**()
Compute the maximum degree of all vertices.

Returns *int* – The highest degree of all vertices.

MeshPlus.vertex_min_degree

MeshPlus.**vertex_min_degree**()
Compute the minimum degree of all vertices.

Returns *int* – The lowest degree of all vertices.

MeshPlus.vertex_neighborhood

MeshPlus.**vertex_neighborhood**(*key*, *ring=1*)
Return the vertices in the neighborhood of a vertex.

Parameters

- **key** (*int*) – The identifier of the vertex.
- **ring** (*int, optional*) – The number of neighborhood rings to include. Default is `1`.

Returns *list* – The vertices in the neighborhood.

Notes

The vertices in the neighborhood are unordered.

Examples

```
>>>
```

MeshPlus.vertex_neighborhood_centroid

MeshPlus.**vertex_neighborhood_centroid**(*key*)
Compute the centroid of the neighbors of a vertex.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *list* – The coordinates of the centroid.

MeshPlus.vertex_neighbors

MeshPlus.**vertex_neighbors**(*key*, *ordered=False*)

Return the neighbors of a vertex.

Parameters

- **key** (*int*) – The identifier of the vertex.
- **ordered** (*bool, optional*) – Return the neighbors in the cycling order of the faces. Default is false.

Returns *list* – The list of neighboring vertices. If the vertex lies on the boundary of the mesh, an ordered list always starts and ends with with boundary vertices.

Notes

Due to the nature of the ordering algorithm, the neighbors cycle around the node in the opposite direction as the cycling direction of the faces. For some algorithms this produces the expected results. For others it doesn't. For example, a dual mesh constructed relying on these conventions will have oposite face cycle directions compared to the original.

Examples

```
>>>
```

MeshPlus.vertex_normal

MeshPlus.**vertex_normal**(*key*)

Return the normal vector at the vertex as the weighted average of the normals of the neighboring faces.

Parameters **key** (*int*) – The identifier of the vertex.

Returns *list* – The components of the normal vector.

MeshPlus.vertices

MeshPlus.**vertices**(*data=False*)

Iterate over the vertices of the mesh.

Parameters **data** (*bool, optional*) – Return the vertex data as well as the vertex keys.

Yields *int or tuple* – The next vertex identifier, if `data` is false. The next vertex as a (key, attr) tuple, if `data` is true.

MeshPlus.vertices_attribute

MeshPlus.**vertices_attribute**(*name*, *value=None*, *keys=None*)
Get or set an attribute of multiple vertices.

Parameters

- **name** (*str*) – The name of the attribute.
- **value** (*obj, optional*) – The value of the attribute. Default is `None`.
- **keys** (*list of int, optional*) – A list of vertex identifiers.

Returns *list or None* – The value of the attribute for each vertex, or `None` if the function is used as a "setter".

Raises **KeyError** – If any of the vertices does not exist.

MeshPlus.vertices_attributes

MeshPlus.**vertices_attributes**(*names=None*, *values=None*, *keys=None*)
Get or set multiple attributes of multiple vertices.

Parameters

- **names** (*list of str, optional*) – The names of the attribute. Default is `None`.
- **values** (*list of obj, optional*) – The values of the attributes. Default is `None`.
- **keys** (*list of int, optional*) – A list of vertex identifiers.

Returns *list or None* – If the parameter `names` is `None`, the function returns a list containing an attribute dict per vertex. If the parameter `names` is not `None`, the function returns a list containing a list of attribute values per vertex corresponding to the provided attribute names. The function returns `None` if it is used as a "setter".

Raises **KeyError** – If any of the vertices does not exist.

MeshPlus.vertices_on_boundaries

MeshPlus.**vertices_on_boundaries**()
Find the vertices on all boundaries of the mesh.

Returns *list of list* – A list of vertex keys per boundary.

Examples

```
>>>
```

MeshPlus.vertices_on_boundary

MeshPlus.**vertices_on_boundary**(*ordered=False*)

Find the vertices on the boundary.

Parameters **ordered** (*bool, optional*) – If `True`, Return the vertices in the same order as they are found on the boundary. Default is `False`.

Returns *list* – The vertices of the boundary.

Warning: If the vertices are requested in order, and the mesh has multiple borders, currently only the vertices of one of the borders will be returned.

Examples

```
>>>
```

MeshPlus.vertices_where

MeshPlus.**vertices_where**(*conditions*, *data=False*)

Get vertices for which a certain condition or set of conditions is true.

Parameters

- **conditions** (*dict*) – A set of conditions in the form of key-value pairs. The keys should be attribute names. The values can be attribute values or ranges of attribute values in the form of min/max pairs.
- **data** (*bool, optional*) – Yield the vertices and their data attributes. Default is `False`.

Yields

- **key** (*hashable*) – The next vertex that matches the condition.
- *2-tuple* – The next vertex and its attributes, if `data=True`.

MeshPlus.vertices_where_predicate

MeshPlus.**vertices_where_predicate**(*predicate*, *data=False*)

Get vertices for which a certain condition or set of conditions is true using a lambda function.

Parameters

- **predicate** (*callable*) – The condition you want to evaluate. The callable takes 2 parameters: `key`, `attr` and should return `True` or `False`.
- **data** (*bool, optional*) – Yield the vertices and their data attributes. Default is `False`.

Yields

- **key** (*hashable*) – The next vertex that matches the condition.
- *2-tuple* – The next vertex and its attributes, if `data=True`.

Examples

```
>>>
```

3.1.4 directional_clustering.plotters

Classes

—

3.1.5 directional_clustering.transformations

Functions

align_vector_field	Aligns vectors to match the orientation of reference vector.
smoothen_vector_field	Apply Laplacian smoothing to a vector field.

align_vector_field

directional_clustering.transformations.**align_vector_field**(*vector_field*, *reference_vector*)

Aligns vectors to match the orientation of reference vector.

Parameters

- **vector_field** (*directional_clustering.fields.VectorField*) – A vector field.
- **reference_vector** (*list* of *float*) – The vector whose orientation is to be matched.

Notes

Comparison made with dot products. Modifies vector field in place.

smoothen_vector_field

directional_clustering.transformations.**smoothen_vector_field**(*vector_field*, *adjacency*, *iters*, *damping=0.5*)

Apply Laplacian smoothing to a vector field.

Parameters

- **vector_field** (*directional_clustering.clustering.VectorField*) – A vector field.
- **adjacency** (*dict*) – A dictionary that maps a key to all the other keys neighboring it.
- **iters** (*int*) – The number of iterations to run this algorithm for.
- **damping** (*float*, optional.) – A coefficient between 0.0 and 1.0 that controls the smoothing strength. 1.0 is maximum smoothing. Defaults to 0.5

Notes

Modifies vector field in place.

CHAPTER

FOUR

LICENSE

```
MIT License

Princeton University

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

PYTHON MODULE INDEX

d

directional_clustering, 5
directional_clustering.clustering, 5
directional_clustering.fields, 11
directional_clustering.mesh, 16
directional_clustering.plotters, 69
directional_clustering.transformations, 69

INDEX

Symbols

__init__() (*directional_clustering.clustering.ClusteringAlgorithm method*), 11
__init__() (*directional_clustering.clustering.ClusteringFactory method*), 10
__init__() (*directional_clustering.clustering.CosineKMeans method*), 7
__init__() (*directional_clustering.clustering.KMeans method*), 6
__init__() (*directional_clustering.clustering.VariationalKMeans method*), 8
__init__() (*directional_clustering.fields.AbstractField method*), 16
__init__() (*directional_clustering.fields.Field method*), 12
__init__() (*directional_clustering.fields.VectorField method*), 13
__init__() (*directional_clustering.mesh.MeshPlus method*), 23

A

AbstractField (*class in directional_clustering.fields*), 16
add_face() (*directional_clustering.mesh.MeshPlus method*), 23
add_vector() (*directional_clustering.fields.VectorField method*), 13
add_vertex() (*directional_clustering.mesh.MeshPlus method*), 23
align_vector_field() (*in module directional_clustering.transformations*), 69
area() (*directional_clustering.mesh.MeshPlus method*), 24

B

bounding_box() (*directional_clustering.mesh.MeshPlus method*), 24
bounding_box_xy() (*directional_clustering.mesh.MeshPlus method*), 24

C

centroid() (*directional_clustering.mesh.MeshPlus method*), 25
clear() (*directional_clustering.mesh.MeshPlus method*), 25
cluster() (*directional_clustering.clustering.ClusteringAlgorithm method*), 11
cluster() (*directional_clustering.clustering.CosineKMeans method*), 7
cluster() (*directional_clustering.clustering.KMeans method*), 6
cluster() (*directional_clustering.clustering.VariationalKMeans method*), 9
clustering_label() (*directional_clustering.mesh.MeshPlus method*), 17
ClusteringAlgorithm (*class in directional_clustering.clustering*), 10
ClusteringFactory (*class in directional_clustering.clustering*), 9
collapse_edge() (*directional_clustering.mesh.MeshPlus method*), 25
connected_components() (*directional_clustering.mesh.MeshPlus method*), 25
copy() (*directional_clustering.mesh.MeshPlus method*), 26
CosineKMeans (*class in directional_clustering.clustering*), 6
create() (*directional_clustering.clustering.ClusteringFactory class method*), 10
cull_vertices() (*directional_clustering.mesh.MeshPlus method*), 26
cut() (*directional_clustering.mesh.MeshPlus method*), 26

D

delete_face() (*directional_clustering.mesh.MeshPlus method*),

26
delete_vertex() (directional_clustering.mesh.MeshPlus method), 27
dimensionality() (directional_clustering.fields.AbstractField method), 16
dimensionality() (directional_clustering.fields.Field method), 12
dimensionality() (directional_clustering.fields.VectorField method), 15
directional_clustering module, 5
directional_clustering.clustering module, 5
directional_clustering.fields module, 11
directional_clustering.mesh module, 16
directional_clustering.plotters module, 69
directional_clustering.transformations module, 69
dual() (directional_clustering.mesh.MeshPlus method), 27

E

edge_attribute() (directional_clustering.mesh.MeshPlus method), 28
edge_attributes() (directional_clustering.mesh.MeshPlus method), 28
edge_coordinates() (directional_clustering.mesh.MeshPlus method), 28
edge_direction() (directional_clustering.mesh.MeshPlus method), 29
edge_faces() (directional_clustering.mesh.MeshPlus method), 29
edge_length() (directional_clustering.mesh.MeshPlus method), 29
edge_loop() (directional_clustering.mesh.MeshPlus method), 29
edge_midpoint() (directional_clustering.mesh.MeshPlus method), 30
edge_point() (directional_clustering.mesh.MeshPlus method), 30
edge_strip() (directional_clustering.mesh.MeshPlus method), 30
edge_vector() (directional_clustering.mesh.MeshPlus method), 30
edges() (directional_clustering.mesh.MeshPlus method), 31
edges_attribute() (directional_clustering.mesh.MeshPlus method), 31
edges_attributes() (directional_clustering.mesh.MeshPlus method), 31
edges_on_boundaries() (directional_clustering.mesh.MeshPlus method), 32
edges_on_boundary() (directional_clustering.mesh.MeshPlus method), 32
edges_where() (directional_clustering.mesh.MeshPlus method), 32
edges_where_predicate() (directional_clustering.mesh.MeshPlus method), 32
euler() (directional_clustering.mesh.MeshPlus method), 33

F

face_adjacency() (directional_clustering.mesh.MeshPlus method), 33
face_adjacency_halfedge() (directional_clustering.mesh.MeshPlus method), 33
face_adjacency_vertices() (directional_clustering.mesh.MeshPlus method), 34
face_area() (directional_clustering.mesh.MeshPlus method), 34
face_aspect_ratio() (directional_clustering.mesh.MeshPlus method), 34
face_attribute() (directional_clustering.mesh.MeshPlus method), 34
face_attributes() (directional_clustering.mesh.MeshPlus method), 35
face_center() (directional_clustering.mesh.MeshPlus method), 35
face_centroid() (direc-

tional_clustering.mesh.MeshPlus method), 35
face_coordinates() (*directional_clustering.mesh.MeshPlus method*), 35
face_corners() (*directional_clustering.mesh.MeshPlus method*), 36
face_curvature() (*directional_clustering.mesh.MeshPlus method*), 36
face_degree() (*directional_clustering.mesh.MeshPlus method*), 36
face_flatness() (*directional_clustering.mesh.MeshPlus method*), 36
face_halfedges() (*directional_clustering.mesh.MeshPlus method*), 37
face_max_degree() (*directional_clustering.mesh.MeshPlus method*), 37
face_min_degree() (*directional_clustering.mesh.MeshPlus method*), 37
face_neighborhood() (*directional_clustering.mesh.MeshPlus method*), 37
face_neighbors() (*directional_clustering.mesh.MeshPlus method*), 37
face_normal() (*directional_clustering.mesh.MeshPlus method*), 38
face_plane() (*directional_clustering.mesh.MeshPlus method*), 38
face_skewness() (*directional_clustering.mesh.MeshPlus method*), 38
face_vertex_ancestor() (*directional_clustering.mesh.MeshPlus method*), 38
face_vertex_descendant() (*directional_clustering.mesh.MeshPlus method*), 39
face_vertices() (*directional_clustering.mesh.MeshPlus method*), 39
faces() (*directional_clustering.mesh.MeshPlus method*), 39
faces_attribute() (*directional_clustering.mesh.MeshPlus method*), 39
faces_attributes() (*directional_clustering.mesh.MeshPlus method*), 40
faces_on_boundary() (*directional_clustering.mesh.MeshPlus method*), 40
faces_where() (*directional_clustering.mesh.MeshPlus method*), 40
faces_where_predicate() (*directional_clustering.mesh.MeshPlus method*), 40
Field (*class in directional_clustering.fields*), 12
flip_cycles() (*directional_clustering.mesh.MeshPlus method*), 41
from_data() (*directional_clustering.mesh.MeshPlus class method*), 41
from_json() (*directional_clustering.mesh.MeshPlus class method*), 41
from_lines() (*directional_clustering.mesh.MeshPlus class method*), 42
from_mesh_faces() (*directional_clustering.fields.VectorField class method*), 13
from_obj() (*directional_clustering.mesh.MeshPlus class method*), 42
from_off() (*directional_clustering.mesh.MeshPlus class method*), 43
from_ply() (*directional_clustering.mesh.MeshPlus class method*), 43
from_points() (*directional_clustering.mesh.MeshPlus class method*), 43
from_polygons() (*directional_clustering.mesh.MeshPlus class method*), 44
from_polyhedron() (*directional_clustering.mesh.MeshPlus class method*), 44
from_polylines() (*directional_clustering.mesh.MeshPlus class method*), 44
from_sequence() (*directional_clustering.fields.VectorField class method*), 14
from_shape() (*directional_clustering.mesh.MeshPlus class method*), 45
from_stl() (*directional_clustering.mesh.MeshPlus class method*), 45
from_vertices_and_faces() (*direc-*

tional_clustering.mesh.MeshPlus class method), 45

G

genus() (*directional_clustering.mesh.MeshPlus method*), 46
get_any_face() (*directional_clustering.mesh.MeshPlus method*), 46
get_any_face_vertex() (*directional_clustering.mesh.MeshPlus method*), 46
get_any_vertex() (*directional_clustering.mesh.MeshPlus method*), 46
get_any_vertices() (*directional_clustering.mesh.MeshPlus method*), 46
gkey_key() (*directional_clustering.mesh.MeshPlus method*), 47

H

halfedge_face() (*directional_clustering.mesh.MeshPlus method*), 47
has_edge() (*directional_clustering.mesh.MeshPlus method*), 47
has_face() (*directional_clustering.mesh.MeshPlus method*), 47
has_halfedge() (*directional_clustering.mesh.MeshPlus method*), 48
has_vertex() (*directional_clustering.mesh.MeshPlus method*), 48

I

index_key() (*directional_clustering.mesh.MeshPlus method*), 48
index_vertex() (*directional_clustering.mesh.MeshPlus method*), 48
insert_vertex() (*directional_clustering.mesh.MeshPlus method*), 48
is_connected() (*directional_clustering.mesh.MeshPlus method*), 49
is_edge_on_boundary() (*directional_clustering.mesh.MeshPlus method*), 49
is_empty() (*directional_clustering.mesh.MeshPlus method*), 50
is_face_on_boundary() (*directional_clustering.mesh.MeshPlus method*), 50
is_manifold() (*directional_clustering.mesh.MeshPlus method*), 50
is_orientable() (*directional_clustering.mesh.MeshPlus method*), 50
is_quadmesh() (*directional_clustering.mesh.MeshPlus method*), 50
is_regular() (*directional_clustering.mesh.MeshPlus method*), 51
is_trimesh() (*directional_clustering.mesh.MeshPlus method*), 51
is_valid() (*directional_clustering.mesh.MeshPlus method*), 51
is_vertex_connected() (*directional_clustering.mesh.MeshPlus method*), 51
is_vertex_on_boundary() (*directional_clustering.mesh.MeshPlus method*), 52
items() (*directional_clustering.fields.VectorField method*), 14

J

join() (*directional_clustering.mesh.MeshPlus method*), 52

K

key_gkey() (*directional_clustering.mesh.MeshPlus method*), 52
key_index() (*directional_clustering.mesh.MeshPlus method*), 53
keys() (*directional_clustering.fields.VectorField method*), 14
KMeans (*class in directional_clustering.clustering*), 5

M

MeshPlus (*class in directional_clustering.mesh*), 17
module
    directional_clustering, 5
    directional_clustering.clustering, 5
    directional_clustering.fields, 11
    directional_clustering.mesh, 16
    directional_clustering.plotters, 69
    directional_clustering.transformations, 69

N

normal() (*directional_clustering.mesh.MeshPlus method*), 53
number_of_edges() (*directional_clustering.mesh.MeshPlus method*), 53
number_of_faces() (*directional_clustering.mesh.MeshPlus method*), 53
number_of_vertices() (*directional_clustering.mesh.MeshPlus method*), 53

Q

quads_to_triangles() (*directional_clustering.mesh.MeshPlus method*), 53

R

register() (*directional_clustering.clustering.ClusteringFactory class method*), 10
remove_unused_vertices() (*directional_clustering.mesh.MeshPlus method*), 53
remove_vector() (*directional_clustering.fields.VectorField method*), 14

S

size() (*directional_clustering.fields.AbstractField method*), 16
size() (*directional_clustering.fields.Field method*), 12
size() (*directional_clustering.fields.VectorField method*), 15
smooth_area() (*directional_clustering.mesh.MeshPlus method*), 53
smooth_centroid() (*directional_clustering.mesh.MeshPlus method*), 54
smoothen_vector_field() (*in module directional_clustering.transformations*), 69
split_edge() (*directional_clustering.mesh.MeshPlus method*), 54
split_face() (*directional_clustering.mesh.MeshPlus method*), 55
summary() (*directional_clustering.mesh.MeshPlus method*), 55

T

to_data() (*directional_clustering.mesh.MeshPlus method*), 56
to_json() (*directional_clustering.mesh.MeshPlus method*), 56
to_lines() (*directional_clustering.mesh.MeshPlus method*), 56
to_obj() (*directional_clustering.mesh.MeshPlus method*), 56
to_off() (*directional_clustering.mesh.MeshPlus method*), 56
to_ply() (*directional_clustering.mesh.MeshPlus method*), 57
to_points() (*directional_clustering.mesh.MeshPlus method*), 57
to_polygons() (*directional_clustering.mesh.MeshPlus method*), 57
to_polylines() (*directional_clustering.mesh.MeshPlus method*), 57
to_quadmesh() (*directional_clustering.mesh.MeshPlus method*), 57
to_sequence() (*directional_clustering.fields.VectorField method*), 15
to_stl() (*directional_clustering.mesh.MeshPlus method*), 57
to_trimesh() (*directional_clustering.mesh.MeshPlus method*), 58
to_vertices_and_faces() (*directional_clustering.mesh.MeshPlus method*), 58
transform() (*directional_clustering.mesh.MeshPlus method*), 58
transform_numpy() (*directional_clustering.mesh.MeshPlus method*), 59
transformed() (*directional_clustering.mesh.MeshPlus method*), 59

U

unify_cycles() (*directional_clustering.mesh.MeshPlus method*), 59
unset_edge_attribute() (*directional_clustering.mesh.MeshPlus method*), 60
unset_face_attribute() (*directional_clustering.mesh.MeshPlus method*), 60
unset_vertex_attribute() (*directional_clustering.mesh.MeshPlus method*), 60

update_default_edge_attributes() (*directional_clustering.mesh.MeshPlus method*), 61
update_default_face_attributes() (*directional_clustering.mesh.MeshPlus method*), 61
update_default_vertex_attributes() (*directional_clustering.mesh.MeshPlus method*), 61

V

validate_data() (*directional_clustering.mesh.MeshPlus method*), 62
validate_json() (*directional_clustering.mesh.MeshPlus method*), 62
VariationalKMeans (*class in directional_clustering.clustering*), 7
vector() (*directional_clustering.fields.VectorField method*), 15
vector_field() (*directional_clustering.mesh.MeshPlus method*), 18
vector_fields() (*directional_clustering.mesh.MeshPlus method*), 18
VectorField (*class in directional_clustering.fields*), 13
vectors() (*directional_clustering.fields.VectorField method*), 15
vertex_area() (*directional_clustering.mesh.MeshPlus method*), 62
vertex_attribute() (*directional_clustering.mesh.MeshPlus method*), 62
vertex_attributes() (*directional_clustering.mesh.MeshPlus method*), 63
vertex_coordinates() (*directional_clustering.mesh.MeshPlus method*), 63
vertex_curvature() (*directional_clustering.mesh.MeshPlus method*), 63
vertex_degree() (*directional_clustering.mesh.MeshPlus method*), 64
vertex_faces() (*directional_clustering.mesh.MeshPlus method*), 64
vertex_index() (*directional_clustering.mesh.MeshPlus method*), 64
vertex_laplacian() (*directional_clustering.mesh.MeshPlus method*), 64
vertex_max_degree() (*directional_clustering.mesh.MeshPlus method*), 65
vertex_min_degree() (*directional_clustering.mesh.MeshPlus method*), 65
vertex_neighborhood() (*directional_clustering.mesh.MeshPlus method*), 65
vertex_neighborhood_centroid() (*directional_clustering.mesh.MeshPlus method*), 65
vertex_neighbors() (*directional_clustering.mesh.MeshPlus method*), 66
vertex_normal() (*directional_clustering.mesh.MeshPlus method*), 66
vertices() (*directional_clustering.mesh.MeshPlus method*), 66
vertices_attribute() (*directional_clustering.mesh.MeshPlus method*), 67
vertices_attributes() (*directional_clustering.mesh.MeshPlus method*), 67
vertices_on_boundaries() (*directional_clustering.mesh.MeshPlus method*), 67
vertices_on_boundary() (*directional_clustering.mesh.MeshPlus method*), 68
vertices_where() (*directional_clustering.mesh.MeshPlus method*), 68
vertices_where_predicate() (*directional_clustering.mesh.MeshPlus method*), 68