Freestyle Types (freestyle.types)

This module contains core classes of the Freestyle Python API, including data types of view map components (0D and 1D elements), base classes for user-defined line stylization rules (predicates, functions, chaining iterators, and stroke shaders), and operators.

Class hierarchy:

- `BBox`
- `BinaryPredicate0D`
- `BinaryPredicate1D`
- `Id`
- `Interface0D`
  - `CurvePoint`
    - `StrokeVertex`
  - `SVertex`
  - `ViewVertex`
    - `NonTVertex`
    - `TVertex`
- `Interface1D`
  - `Curve`
    - `Chain`
  - `FEdge`
    - `FEdgeSharp`
    - `FEdgeSmooth`
  - `Stroke`
  - `ViewEdge`
- `Iterator`
  - `AdjacencyIterator`
  - `CurvePointIterator`
  - `Interface0DIterator`
  - `SVertexIterator`
  - `StrokeVertexIterator`
  - `ViewEdgeIterator`
    - `ChainingIterator`
  - `orientedViewEdgeIterator`
- `Material`
- `Noise`
- `Operators`
- `SShape`
- `StrokeAttribute`
- `StrokeShader`
- `UnaryFunction0D`
  - `UnaryFunction0DDouble`
  - `UnaryFunction0DEdgeNature`
  - `UnaryFunction0DFloat`
  - `UnaryFunction0DId`
  - `UnaryFunction0DMaterial`
  - `UnaryFunction0DUnsigned`
  - `UnaryFunction0DVec2f`

  - UnaryFunction0DVec3f
  - UnaryFunction0DVectorViewShape
  - UnaryFunction0DViewShape
- UnaryFunction1D
  - UnaryFunction1DDouble
  - UnaryFunction1DEdgeNature
  - UnaryFunction1DFloat
  - UnaryFunction1DUnsigned
  - UnaryFunction1DVec2f
  - UnaryFunction1DVec3f
  - UnaryFunction1DVectorViewShape
  - UnaryFunction1DVoid
- UnaryPredicate0D
- UnaryPredicate1D
- ViewMap
- ViewShape
- IntegrationType
- MediumType
- Nature

class freestyle.types.**AdjacencyIterator**

Class hierarchy: `Iterator` > `AdjacencyIterator`

Class for representing adjacency iterators used in the chaining process. An AdjacencyIterator is created in the increment() and decrement() method of a `ChainingIterator` and passed to the traverse() method of the ChainingIterator.

__init__()

__init__(brother)

__init__(vertex, restrict_to_selection=True, restrict_to_unvisited=True)

Builds an `AdjacencyIterator` using the default constructor, copy constructor or the overloaded constructor.

PARAMETERS:

- **brother** (`AdjacencyIterator`) – An AdjacencyIterator object.
- **vertex** (`ViewVertex`) – The vertex which is the next crossing.
- **restrict_to_selection** (*bool*) – Indicates whether to force the chaining to stay within the set of selected ViewEdges or not.
- **restrict_to_unvisited** (*bool*) – Indicates whether a ViewEdge that has already been chained must be ignored ot not.

is_incoming

True if the current ViewEdge is coming towards the iteration vertex, and False otherwise.

TYPE:

bool

object

The ViewEdge object currently pointed to by this iterator.

TYPE:

`ViewEdge`

class freestyle.types.**BBox**

Class for representing a bounding box.

__init__()

Default constructor.

class freestyle.types.**BinaryPredicate0D**

Base class for binary predicates working on `Interface0D` objects. A BinaryPredicate0D is typically an ordering relation between two Interface0D objects. The predicate evaluates a relation between the two Interface0D instances and returns a boolean value (true or false). It is usec by invoking the __call__() method.

__init__()

Default constructor.

__call__(inter1, inter2)

Must be overload by inherited classes. It evaluates a relation between two Interface0D objects.

PARAMETERS:

- **inter1** (`Interface0D`) – The first Interface0D object.
- **inter2** (`Interface0D`) – The second Interface0D object.

RETURNS:

True or false.

RETURN TYPE:

bool

name

The name of the binary 0D predicate.

TYPE:

str

class freestyle.types.**BinaryPredicate1D**

Base class for binary predicates working on `Interface1D` objects. A BinaryPredicate1D is typically an ordering relation between two Interface1D objects. The predicate evaluates a relation between the two Interface1D instances and returns a boolean value (true or false). It is usec by invoking the __call__() method.

__init__()

Default constructor.

__call__(inter1, inter2)

Must be overload by inherited classes. It evaluates a relation between two Interface1D objects.

PARAMETERS:

- **inter1** (`Interface1D`) – The first Interface1D object.
- **inter2** (`Interface1D`) – The second Interface1D object.

RETURNS:

True or false.

RETURN TYPE:

bool

name

The name of the binary 1D predicate.

TYPE:

str

class freestyle.types.**Chain**

Class hierarchy: `Interface1D` > `Curve` > `Chain`

Class hierarchy: `Interface1D` > `Curve` > `Chain`

Class to represent a 1D elements issued from the chaining process. A Chain is the last step before the `Stroke` and is used in the Splitting and Creation processes.

__init__()

__init__(brother)

__init__(id)

Builds a `Chain` using the default constructor, copy constructor or from an `Id` .

PARAMETERS:

- **brother** (`Chain`) – A Chain object.
- **id** (`Id`) – An Id object.

push_viewedge_back(viewedge, orientation)

Adds a ViewEdge at the end of the Chain.

PARAMETERS:

- **viewedge** (`ViewEdge`) – The ViewEdge that must be added.
- **orientation** (*bool*) – The orientation with which the ViewEdge must be processed.

push_viewedge_front(viewedge, orientation)

Adds a ViewEdge at the beginning of the Chain.

PARAMETERS:

- **viewedge** (`ViewEdge`) – The ViewEdge that must be added.
- **orientation** (*bool*) – The orientation with which the ViewEdge must be processed.

class freestyle.types.**ChainingIterator**

Class hierarchy: `Iterator` > `ViewEdgeIterator` > `ChainingIterator`

Base class for chaining iterators. This class is designed to be overloaded in order to describe chaining rules. It makes the description of chaining rul easier. The two main methods that need to overloaded are traverse() and init(). traverse() tells which `ViewEdge` to follow, among the adjacent ones. If you specify restriction rules (such as "Chain only ViewEdges of the selection"), they will be included in the adjacency iterator (i.e, the adjacent iterator will only stop on "valid" edges).

__init__(restrict_to_selection=True, restrict_to_unvisited=True, begin=None, orientation=True)

__init__(brother)

Builds a Chaining Iterator from the first ViewEdge used for iteration and its orientation or by using the copy constructor.

PARAMETERS:

- **restrict_to_selection** (*bool*) – Indicates whether to force the chaining to stay within the set of selected ViewEdges or not.
- **restrict_to_unvisited** (*bool*) – Indicates whether a ViewEdge that has already been chained must be ignored ot not.
- **begin** (`ViewEdge` | None) – The ViewEdge from which to start the chain.
- **orientation** (*bool*) – The direction to follow to explore the graph. If true, the direction indicated by the first ViewEdge is used.
- **brother** (*ChainingIterator*)

init()

Initializes the iterator context. This method is called each time a new chain is started. It can be used to reset some history information that you might want to keep.

traverse(it)

This method iterates over the potential next ViewEdges and returns the one that will be followed next. Returns the next ViewEdge to follow o None when the end of the chain is reached.

PARAMETERS:

it (`AdjacencyIterator`) – The iterator over the ViewEdges adjacent to the end vertex of the current ViewEdge. The adjacency iterator reflects the restriction rules by only iterating over the valid ViewEdges.

RETURNS:

Returns the next ViewEdge to follow, or None if chaining ends.

RETURN TYPE:

`ViewEdge` | None

is_incrementing

True if the current iteration is an incrementation.

TYPE:

bool

next_vertex

The ViewVertex that is the next crossing.

TYPE:

`ViewVertex`

object

The ViewEdge object currently pointed by this iterator.

TYPE:

`ViewEdge`

class freestyle.types.**Curve**

Class hierarchy: `Interface1D` > `Curve`

Base class for curves made of CurvePoints. `SVertex` is the type of the initial curve vertices. A `Chain` is a specialization of a Curve.

__init__()

__init__(brother)

__init__(id)

Builds a `FrsCurve` using a default constructor, copy constructor or from an `Id` .

PARAMETERS:

- **brother** (`Curve`) – A Curve object.
- **id** (`Id`) – An Id object.

push_vertex_back(vertex)

Adds a single vertex at the end of the Curve.

PARAMETERS:

vertex (`SVertex` | `CurvePoint`) – A vertex object.

push_vertex_front(vertex)

Adds a single vertex at the front of the Curve.

PARAMETERS:

vertex (`SVertex` | `CurvePoint`) – A vertex object.

is_empty

True if the Curve doesn't have any Vertex yet.

TYPE:

bool

segments_size

The number of segments in the polyline constituting the Curve.

TYPE:

int

class freestyle.types.**CurvePoint**

Class hierarchy: `Interface0D` > `CurvePoint`

Class to represent a point of a curve. A CurvePoint can be any point of a 1D curve (it doesn't have to be a vertex of the curve). Any `Interface1D` is built upon ViewEdges, themselves built upon FEdges. Therefore, a curve is basically a polyline made of a list of `SVertex` objects. Thus, a CurvePoint is built by linearly interpolating two `SVertex` instances. CurvePoint can be used as virtual points while querying 0D information along a curve at a given resolution.

__init__()

__init__(brother)

__init__(first_vertex, second_vertex, t2d)

__init__(first_point, second_point, t2d)

Builds a CurvePoint using the default constructor, copy constructor, or one of the overloaded constructors. The over loaded constructors can either take two `SVertex` or two `CurvePoint` objects and an interpolation parameter

PARAMETERS:

- **brother** (`CurvePoint`) – A CurvePoint object.
- **first_vertex** (`SVertex`) – The first SVertex.
- **second_vertex** (`SVertex`) – The second SVertex.
- **first_point** (`CurvePoint`) – The first CurvePoint.
- **second_point** (`CurvePoint`) – The second CurvePoint.
- **t2d** (*float*) – A 2D interpolation parameter used to linearly interpolate first_vertex and second_vertex or first_point and second_point.

fedge

Gets the FEdge for the two SVertices that given CurvePoints consists out of. A shortcut for CurvePoint.first_svertex.get_fedge(CurvePoint.second_svertex).

TYPE:

`FEdge`

first_svertex

The first SVertex upon which the CurvePoint is built.

TYPE:

`SVertex`

second_svertex

The second SVertex upon which the CurvePoint is built.

TYPE:

`SVertex`

t2d

The 2D interpolation parameter.

TYPE:

float

class freestyle.types.**CurvePointIterator**

Class hierarchy: `Iterator` > `CurvePointIterator`

Class representing an iterator on a curve. Allows an iterating outside initial vertices. A CurvePoint is instantiated and returned through the .object attribute.

__init__()

__init__(brother)

__init__(step=0.0)

Builds a CurvePointIterator object using either the default constructor, copy constructor, or the overloaded constructor.

PARAMETERS:

- **brother** (`CurvePointIterator`) – A CurvePointIterator object.
- **step** (*float*) – A resampling resolution with which the curve is resampled. If zero, no resampling is done (i.e., the iterator iterates over ini vertices).

object

The CurvePoint object currently pointed by this iterator.

TYPE:

`CurvePoint`

t

The curvilinear abscissa of the current point.

TYPE:

float

u

The point parameter at the current point in the stroke (0 <= u <= 1).

TYPE:

float

class freestyle.types.**FEdge**

Class hierarchy: `Interface1D` > `FEdge`

Base Class for feature edges. This FEdge can represent a silhouette, a crease, a ridge/valley, a border or a suggestive contour. For silhouettes, the FEdge is oriented so that the visible face lies on the left of the edge. For borders, the FEdge is oriented so that the face lies on the left of the edge. FEdge can represent an initial edge of the mesh or runs across a face of the initial mesh depending on the smoothness or sharpness of the mesh. Thi class is specialized into a smooth and a sharp version since their properties slightly vary from one to the other.

FEdge()

FEdge(brother)

Builds an `FEdge` using the default constructor, copy constructor, or between two `SVertex` objects.

PARAMETERS:

- **brother** (`FEdge`) – An FEdge object.
- **first_vertex** (`SVertex`) – The first SVertex.
- **second_vertex** (`SVertex`) – The second SVertex.

first_svertex

The first SVertex constituting this FEdge.

TYPE:

`SVertex`

id

The Id of this FEdge.

TYPE:

`Id`

is_smooth

True if this FEdge is a smooth FEdge.

TYPE:

bool

nature

The nature of this FEdge.

TYPE:

`Nature`

next_fedge

The FEdge following this one in the ViewEdge. The value is None if this FEdge is the last of the ViewEdge.

TYPE:

`FEdge`

previous_fedge

The FEdge preceding this one in the ViewEdge. The value is None if this FEdge is the first one of the ViewEdge.

TYPE:

`FEdge`

second_svertex

The second SVertex constituting this FEdge.

TYPE:

`SVertex`

viewedge

The ViewEdge to which this FEdge belongs to.

TYPE:

`ViewEdge`

class freestyle.types.**FEdgeSharp**

Class hierarchy: `Interface1D` > `FEdge` > `FEdgeSharp`

Class defining a sharp FEdge. A Sharp FEdge corresponds to an initial edge of the input mesh. It can be a silhouette, a crease or a border. If it is a crease edge, then it is bordered by two faces of the mesh. Face a lies on its right whereas Face b lies on its left. If it is a border edge, then it doesn' have any face on its right, and thus Face a is None.

__init__()

__init__(brother)

__init__(first_vertex, second_vertex)

Builds an `FEdgeSharp` using the default constructor, copy constructor, or between two `SVertex` objects.

PARAMETERS:

- **brother** (`FEdgeSharp`) – An FEdgeSharp object.
- **first_vertex** (`SVertex`) – The first SVertex object.
- **second_vertex** (`SVertex`) – The second SVertex object.

face_mark_left

The face mark of the face lying on the left of the FEdge.

TYPE:
bool

face_mark_right

The face mark of the face lying on the right of the FEdge. If this FEdge is a border, it has no face on the right and thus this property is set to false.

TYPE:
bool

material_index_left

The index of the material of the face lying on the left of the FEdge.

TYPE:
int

material_index_right

The index of the material of the face lying on the right of the FEdge. If this FEdge is a border, it has no Face on its right and therefore no material.

TYPE:
int

material_left

The material of the face lying on the left of the FEdge.

TYPE:
`Material`

material_right

The material of the face lying on the right of the FEdge. If this FEdge is a border, it has no Face on its right and therefore no material.

TYPE:
`Material`

normal_left

The normal to the face lying on the left of the FEdge.

TYPE:
`mathutils.Vector`

normal_right

The normal to the face lying on the right of the FEdge. If this FEdge is a border, it has no Face on its right and therefore no normal.

TYPE:
`mathutils.Vector`

class freestyle.types.**FEdgeSmooth**

Class hierarchy: `Interface1D` > `FEdge` > `FEdgeSmooth`

Class defining a smooth edge. This kind of edge typically runs across a face of the input mesh. It can be a silhouette, a ridge or valley, a suggestive contour.

__init__()

__init__(brother)

__init__(first_vertex, second_vertex)

Builds an `FEdgeSmooth` using the default constructor, copy constructor, or between two `SVertex`.

PARAMETERS:

- **brother** (`FEdgeSmooth`) – An FEdgeSmooth object.
- **first_vertex** (`SVertex`) – The first SVertex object.
- **second_vertex** (`SVertex`) – The second SVertex object.

face_mark

The face mark of the face that this FEdge is running across.

TYPE:

bool

material

The material of the face that this FEdge is running across.

TYPE:

`Material`

material_index

The index of the material of the face that this FEdge is running across.

TYPE:

int

normal

The normal of the face that this FEdge is running across.

TYPE:

`mathutils.Vector`

class freestyle.types.**Id**

Class for representing an object Id.

__init__(brother)

__init__(first=0, second=0)

Build the Id from two numbers or another `Id` using the copy constructor.

PARAMETERS:

- **brother** (`Id` :arg first: The first number.) – An Id object.
- **second** (*int*) – The second number.

first

The first number constituting the Id.

TYPE:

int

second

The second number constituting the Id.

TYPE:

int

class freestyle.types.**IntegrationType**

Class hierarchy: int > `IntegrationType`

Different integration methods that can be invoked to integrate into a single value the set of values obtained from each 0D element of an 1D element:

- IntegrationType.MEAN: The value computed for the 1D element is the mean of the values obtained for the 0D elements.
- IntegrationType.MIN: The value computed for the 1D element is the minimum of the values obtained for the 0D elements.
- IntegrationType.MAX: The value computed for the 1D element is the maximum of the values obtained for the 0D elements.
- IntegrationType.FIRST: The value computed for the 1D element is the first of the values obtained for the 0D elements.
- IntegrationType.LAST: The value computed for the 1D element is the last of the values obtained for the 0D elements.

class freestyle.types.**Interface0D**

Base class for any 0D element.

__init__()

Default constructor.

get_fedge(inter)

Returns the FEdge that lies between this 0D element and the 0D element given as the argument.

PARAMETERS:

inter (`Interface0D`) – A 0D element.

RETURNS:

The FEdge lying between the two 0D elements.

RETURN TYPE:

`FEdge`

id

The Id of this 0D element.

TYPE:

`Id`

name

The string of the name of this 0D element.

TYPE:

str

nature

The nature of this 0D element.

TYPE:

`Nature`

point_2d

The 2D point of this 0D element.

TYPE:

`mathutils.Vector`

point_3d

The 3D point of this 0D element.

TYPE:

`mathutils.Vector`

projected_x

The X coordinate of the projected 3D point of this 0D element.

TYPE:

float

projected_y

The Y coordinate of the projected 3D point of this 0D element.

TYPE:

float

projected_z

The Z coordinate of the projected 3D point of this 0D element.

TYPE:

float

class freestyle.types.**Interface0DIterator**

Class hierarchy: `Iterator` > `Interface0DIterator`

Class defining an iterator over Interface0D elements. An instance of this iterator is always obtained from a 1D element.

__init__(brother)

__init__(it)

Construct a nested Interface0DIterator using either the copy constructor or the constructor that takes an he argument of a Function0D.

PARAMETERS:

- **brother** (`Interface0DIterator`) – An Interface0DIterator object.
- **it** (`SVertexIterator` , `CurvePointIterator` , or `StrokeVertexIterator`) – An iterator object to be nested.

at_last

True if the iterator points to the last valid element. For its counterpart (pointing to the first valid element), use it.is_begin.

TYPE:

bool

object

The 0D object currently pointed to by this iterator. Note that the object may be an instance of an Interface0D subclass. For example if the iterator has been created from the *vertices_begin()* method of the `Stroke` class, the .object property refers to a `StrokeVertex` object.

TYPE:

`Interface0D` or one of its subclasses.

t

The curvilinear abscissa of the current point.

TYPE:

float

u

The point parameter at the current point in the 1D element (0 <= u <= 1).

TYPE:

float

class freestyle.types.**Interface1D**

Base class for any 1D element.

__init__()

Default constructor.

points_begin(t=0.0)

Returns an iterator over the Interface1D points, pointing to the first point. The difference with vertices_begin() is that here we can iterate over points of the 1D element at a any given sampling. Indeed, for each iteration, a virtual point is created.

PARAMETERS:

t (*float*) – A sampling with which we want to iterate over points of this 1D element.

RETURNS:

An Interface0DIterator pointing to the first point.

RETURN TYPE:

`Interface0DIterator`

points_end(t=0.0)

Returns an iterator over the Interface1D points, pointing after the last point. The difference with vertices_end() is that here we can iterate over points of the 1D element at a given sampling. Indeed, for each iteration, a virtual point is created.

PARAMETERS:

t (*float*) – A sampling with which we want to iterate over points of this 1D element.

RETURNS:

An Interface0DIterator pointing after the last point.

RETURN TYPE:

`Interface0DIterator`

vertices_begin()

Returns an iterator over the Interface1D vertices, pointing to the first vertex.

RETURNS:

An Interface0DIterator pointing to the first vertex.

RETURN TYPE:

`Interface0DIterator`

vertices_end()

Returns an iterator over the Interface1D vertices, pointing after the last vertex.

RETURNS:

An Interface0DIterator pointing after the last vertex.

RETURN TYPE:

`Interface0DIterator`

id

The Id of this Interface1D.

TYPE:

`Id`

length_2d

The 2D length of this Interface1D.

TYPE:

float

name

The string of the name of the 1D element.

TYPE:

str

nature

The nature of this Interface1D.

TYPE:

`Nature`

time_stamp

The time stamp of the 1D element, mainly used for selection.

TYPE:

int

class freestyle.types.**Iterator**

Base class to define iterators.

__init__()

Default constructor.

decrement()

Makes the iterator point the previous element.

increment()

Makes the iterator point the next element.

is_begin

True if the iterator points to the first element.

TYPE:

bool

is_end

True if the iterator points to the last element.

TYPE:

bool

name

The string of the name of this iterator.

TYPE:

str

class freestyle.types.**Material**

Class defining a material.

__init__()

__init__(brother)

__init__(line, diffuse, ambient, specular, emission, shininess, priority)

Creates a `FrsMaterial` using either default constructor, copy constructor, or an overloaded constructor

PARAMETERS:

- **brother** (`Material`) – A Material object to be used as a copy constructor.
- **line** (`mathutils.Vector` | tuple[float, float, float, float] | list[float]) – The line color.
- **diffuse** – The diffuse color.
- **ambient** (`mathutils.Vector` | tuple[float, float, float, float] | list[float]) – The ambient color.
- **specular** (`mathutils.Vector` | tuple[float, float, float, float] | list[float]) – The specular color.
- **emission** (`mathutils.Vector` | tuple[float, float, float, float] | list[float]) – The emissive color.
- **shininess** (*float*) – The shininess coefficient.
- **priority** (*int*) – The line color priority.

ambient

RGBA components of the ambient color of the material.

TYPE:

`mathutils.Color`

diffuse

RGBA components of the diffuse color of the material.

TYPE:

`mathutils.Vector`

emission

RGBA components of the emissive color of the material.

TYPE:

`mathutils.Color`

line

RGBA components of the line color of the material.

TYPE:

`mathutils.Vector`

priority

Line color priority of the material.

TYPE:

int

shininess

Shininess coefficient of the material.

TYPE:

float

specular

RGBA components of the specular color of the material.

TYPE:

`mathutils.Vector`

class freestyle.types.**MediumType**

Class hierarchy: int > `MediumType`

The different blending modes available to simulate the interaction media-medium:

- Stroke.DRY_MEDIUM: To simulate a dry medium such as Pencil or Charcoal.
- Stroke.HUMID_MEDIUM: To simulate ink painting (color subtraction blending).
- Stroke.OPAQUE_MEDIUM: To simulate an opaque medium (oil, spray…).

class freestyle.types.**Nature**

Class hierarchy: int > `Nature`

Different possible natures of 0D and 1D elements of the ViewMap.

Vertex natures:

- Nature.POINT: True for any 0D element.
- Nature.S_VERTEX: True for SVertex.
- Nature.VIEW_VERTEX: True for ViewVertex.
- Nature.NON_T_VERTEX: True for NonTVertex.
- Nature.T_VERTEX: True for TVertex.
- Nature.CUSP: True for CUSP.

Edge natures:

- Nature.NO_FEATURE: True for non feature edges (always false for 1D elements of the ViewMap).
- Nature.SILHOUETTE: True for silhouettes.
- Nature.BORDER: True for borders.
- Nature.CREASE: True for creases.
- Nature.RIDGE: True for ridges.
- Nature.VALLEY: True for valleys.
- Nature.SUGGESTIVE_CONTOUR: True for suggestive contours.
- Nature.MATERIAL_BOUNDARY: True for edges at material boundaries.
- Nature.EDGE_MARK: True for edges having user-defined edge marks.

class freestyle.types.**Noise**

Class to provide Perlin noise functionalities.

__init__(seed=-1)

Builds a Noise object. Seed is an optional argument. The seed value is used as a seed for random number generation if it is equal to or greater than zero; otherwise, time is used as a seed.

PARAMETERS:

seed (*int*) – Seed for random number generation.

Undocumented, consider contributing.

smoothNoise1(v)

Returns a smooth noise value for a 1D element.

PARAMETERS:

v (*float*) – One-dimensional sample point.

RETURNS:

A smooth noise value.

RETURN TYPE:

float

smoothNoise2(v)

Returns a smooth noise value for a 2D element.

Returns a smooth noise value for a 2D element.

PARAMETERS:

v (`mathutils.Vector` | tuple[float, float] | list[float]) – Two-dimensional sample point.

RETURNS:

A smooth noise value.

RETURN TYPE:

float

smoothNoise3(v)

Returns a smooth noise value for a 3D element.

PARAMETERS:

v (`mathutils.Vector` | tuple[float, float, float] | list[float]) – Three-dimensional sample point.

RETURNS:

A smooth noise value.

RETURN TYPE:

float

turbulence1(v, freq, amp, oct=4)

Returns a noise value for a 1D element.

PARAMETERS:

- **v** (*float*) – One-dimensional sample point.
- **freq** (*float*) – Noise frequency.
- **amp** (*float*) – Amplitude.
- **oct** (*int*) – Number of octaves.

RETURNS:

A noise value.

RETURN TYPE:

float

turbulence2(v, freq, amp, oct=4)

Returns a noise value for a 2D element.

PARAMETERS:

- **v** (`mathutils.Vector` | tuple[float, float] | list[float]) – Two-dimensional sample point.
- **freq** (*float*) – Noise frequency.
- **amp** (*float*) – Amplitude.
- **oct** (*int*) – Number of octaves.

RETURNS:

A noise value.

RETURN TYPE:

float

turbulence3(v, freq, amp, oct=4)

Returns a noise value for a 3D element.

PARAMETERS:

- **v** (`mathutils.Vector` | tuple[float, float, float] | list[float]) – Three-dimensional sample point.
- **freq** (*float*) – Noise frequency.
- **amp** (*float*) – Amplitude.

- **oct** (*int*) – Number of octaves.

RETURNS:

A noise value.

RETURN TYPE:

float

Undocumented, consider contributing.

class freestyle.types.**NonTVertex**

Class hierarchy: `Interface0D` > `ViewVertex` > `NonTVertex`

View vertex for corners, cusps, etc. associated to a single SVertex. Can be associated to 2 or more view edges.

__init__()

__init__(svertex)

Builds a `NonTVertex` using the default constructor or a `SVertex` .

PARAMETERS:

svertex (`SVertex`) – An SVertex object.

svertex

The SVertex on top of which this NonTVertex is built.

TYPE:

`SVertex`

class freestyle.types.**Operators**

Class defining the operators used in a style module. There are five types of operators: Selection, chaining, splitting, sorting and creation. All these operators are user controlled through functors, predicates and shaders that are taken as arguments.

static bidirectional_chain(it, pred)

static bidirectional_chain(it)

Builds a set of chains from the current set of ViewEdges. Each ViewEdge of the current list potentially starts a new chain. The chaining operat then iterates over the ViewEdges of the ViewMap using the user specified iterator. This operator iterates both using the increment and decrement operators and is therefore bidirectional. This operator works with a ChainingIterator which contains the chaining rules. It is this last one which can be told to chain only edges that belong to the selection or not to process twice a ViewEdge during the chaining. Each time a ViewEdge is added to a chain, its chaining time stamp is incremented. This allows you to keep track of the number of chains to which a ViewEdge belongs to.

PARAMETERS:

- **it** (`ChainingIterator`) – The ChainingIterator on the ViewEdges of the ViewMap. It contains the chaining rule.
- **pred** (`UnaryPredicate1D`) – The predicate on the ViewEdge that expresses the stopping condition. This parameter is optional, you make not want to pass a stopping criterion when the stopping criterion is already contained in the iterator definition.

static chain(it, pred, modifier)

static chain(it, pred)

Builds a set of chains from the current set of ViewEdges. Each ViewEdge of the current list starts a new chain. The chaining operator then iterates over the ViewEdges of the ViewMap using the user specified iterator. This operator only iterates using the increment operator and is therefore unidirectional.

PARAMETERS:

- **it** (`ViewEdgeIterator`) – The iterator on the ViewEdges of the ViewMap. It contains the chaining rule.
- **pred** (`UnaryPredicate1D`) – The predicate on the ViewEdge that expresses the stopping condition.
- **modifier** (`UnaryFunction1DVoid`) – A function that takes a ViewEdge as argument and that is used to modify the processed ViewEdge state (the timestamp incrementation is a typical illustration of such a modifier). If this argument is not given, the time stamp is

ViewEdge state (the timestamp incrementation is a typical illustration of such a modifier). If this argument is not given, the time stamp is automatically managed.

static create(pred, shaders)

Creates and shades the strokes from the current set of chains. A predicate can be specified to make a selection pass on the chains.

PARAMETERS:

- **pred** (`UnaryPredicate1D`) – The predicate that a chain must verify in order to be transform as a stroke.
- **shaders** (list[`StrokeShader`]) – The list of shaders used to shade the strokes.

static get_chain_from_index(i)

Returns the Chain at the index in the current set of Chains.

PARAMETERS:

i (*int*) – index (0 <= i < Operators.get_chains_size()).

RETURNS:

The Chain object.

RETURN TYPE:

`Chain`

static get_chains_size()

Returns the number of Chains.

RETURNS:

The number of Chains.

RETURN TYPE:

int

static get_stroke_from_index(i)

Returns the Stroke at the index in the current set of Strokes.

PARAMETERS:

i (*int*) – index (0 <= i < Operators.get_strokes_size()).

RETURNS:

The Stroke object.

RETURN TYPE:

`Stroke`

static get_strokes_size()

Returns the number of Strokes.

RETURNS:

The number of Strokes.

RETURN TYPE:

int

static get_view_edges_size()

Returns the number of ViewEdges.

RETURNS:

The number of ViewEdges.

RETURN TYPE:

int

static get_viewedge_from_index(i)

Returns the ViewEdge at the index in the current set of ViewEdges.

PARAMETERS:

i (*int*) – index (0 <= i < Operators.get_view_edges_size()).

RETURNS:

The ViewEdge object.

RETURN TYPE:

`ViewEdge`

static recursive_split(func, pred_1d, sampling=0.0)

static recursive_split(func, pred_0d, pred_1d, sampling=0.0)

Splits the current set of chains in a recursive way. We process the points of each chain (with a specified sampling) to find the point minimizing a specified function. The chain is split in two at this point and the two new chains are processed in the same way. The recursivity level is controll through a predicate 1D that expresses a stopping condition on the chain that is about to be processed.

The user can also specify a 0D predicate to make a first selection on the points that can potentially be split. A point that doesn't verify the 0D predicate won't be candidate in realizing the min.

PARAMETERS:

- **func** (`UnaryFunction0DDouble`) – The Unary Function evaluated at each point of the chain. The splitting point is the point minimizing this function.
- **pred_0d** (`UnaryPredicate0D`) – The Unary Predicate 0D used to select the candidate points where the split can occur. For example, it is very likely that would rather have your chain splitting around its middle point than around one of its extremities. A 0D predicate working on the curvilinear abscissa allows to add this kind of constraints.
- **pred_1d** (`UnaryPredicate1D`) – The Unary Predicate expressing the recursivity stopping condition. This predicate is evaluated each curve before it actually gets split. If pred_1d(chain) is true, the curve won't be split anymore.
- **sampling** (*float*) – The resolution used to sample the chain for the predicates evaluation. (The chain is not actually resampled; a virtual point only progresses along the curve using this resolution.)

static reset(delete_strokes=True)

Resets the line stylization process to the initial state. The results of stroke creation are accumulated if **delete_strokes** is set to False.

PARAMETERS:

delete_strokes (*bool*) – Delete the strokes that are currently stored.

static select(pred)

Selects the ViewEdges of the ViewMap verifying a specified condition.

PARAMETERS:

pred (`UnaryPredicate1D`) – The predicate expressing this condition.

static sequential_split(starting_pred, stopping_pred, sampling=0.0)

static sequential_split(pred, sampling=0.0)

Splits each chain of the current set of chains in a sequential way. The points of each chain are processed (with a specified sampling) sequential The first point of the initial chain is the first point of one of the resulting chains. The splitting ends when no more chain can start.

Tip

By specifying a starting and stopping predicate allows the chains to overlap rather than chains partitioning.

PARAMETERS:

- **starting_pred** (`UnaryPredicate0D`) – The predicate on a point that expresses the starting condition. Each time this condition is verified, a new chain begins
- **stopping_pred** (`UnaryPredicate0D`) – The predicate on a point that expresses the stopping condition. The chain ends as soon

this predicate is verified.

- **pred** (`UnaryPredicate0D`) – The predicate on a point that expresses the splitting condition. Each time the condition is verified, tl chain is split into two chains. The resulting set of chains is a partition of the initial chain
- **sampling** (*float*) – The resolution used to sample the chain for the predicates evaluation. (The chain is not actually resampled; a virtual point only progresses along the curve using this resolution.)

static sort(pred)

Sorts the current set of chains (or viewedges) according to the comparison predicate given as argument.

PARAMETERS:

pred (`BinaryPredicate1D`) – The binary predicate used for the comparison.

class freestyle.types.**SShape**

Class to define a feature shape. It is the gathering of feature elements from an identified input shape.

__init__()

__init__(brother)

Creates a `SShape` class using either a default constructor or copy constructor.

PARAMETERS:

brother (`SShape`) – An SShape object.

add_edge(edge)

Adds an FEdge to the list of FEdges.

PARAMETERS:

edge (`FEdge`) – An FEdge object.

add_vertex(vertex)

Adds an SVertex to the list of SVertex of this Shape. The SShape attribute of the SVertex is also set to this SShape.

PARAMETERS:

vertex (`SVertex`) – An SVertex object.

compute_bbox()

Compute the bbox of the SShape.

bbox

The bounding box of the SShape.

TYPE:

`BBox`

edges

The list of edges constituting this SShape.

TYPE:

List of `FEdge`

id

The Id of this SShape.

TYPE:

`Id`

name

The name of the SShape.

TYPE:

str

vertices

The list of vertices constituting this SShape.

TYPE:

List of `SVertex`

class freestyle.types.**SVertex**

Class hierarchy: `Interface0D` > `SVertex`

Class to define a vertex of the embedding.

__init__()

__init__(brother)

__init__(point_3d, id)

Builds a `SVertex` using the default constructor, copy constructor or the overloaded constructor which builds a `SVertex` from 3D coordinates and an Id.

PARAMETERS:

- **brother** (`SVertex`) – A SVertex object.
- **point_3d** (`mathutils.Vector`) – A three-dimensional vector.
- **id** (`Id`) – An Id object.

add_fedge(fedge)

Add an FEdge to the list of edges emanating from this SVertex.

PARAMETERS:

fedge (`FEdge`) – An FEdge.

add_normal(normal)

Adds a normal to the SVertex's set of normals. If the same normal is already in the set, nothing changes.

PARAMETERS:

normal (`mathutils.Vector` | tuple[float, float, float] | list[float]) – A three-dimensional vector.

curvatures

Curvature information expressed in the form of a seven-element tuple (K1, e1, K2, e2, Kr, er, dKr), where K1 and K2 are scalar values representing the first (maximum) and second (minimum) principal curvatures at this SVertex, respectively; e1 and e2 are three-dimensional vectors representing the first and second principal directions, i.e. the directions of the normal plane where the curvature takes its maximum anc minimum values, respectively; and Kr, er and dKr are the radial curvature, radial direction, and the derivative of the radial curvature at this SVertex, respectively.

TYPE:

tuple

id

The Id of this SVertex.

TYPE:

`Id`

normals

The normals for this Vertex as a list. In a sharp surface, an SVertex has exactly one normal. In a smooth surface, an SVertex can have any number of normals.

TYPE:

list of `mathutils.Vector`

normals_size

The number of different normals for this SVertex.

TYPE:

int

point_2d

The projected 3D coordinates of the SVertex.

TYPE:

`mathutils.Vector`

point_3d

The 3D coordinates of the SVertex.

TYPE:

`mathutils.Vector`

viewvertex

If this SVertex is also a ViewVertex, this property refers to the ViewVertex, and None otherwise.

TYPE:

`ViewVertex`

class freestyle.types.**SVertexIterator**

Class hierarchy: `Iterator` > `SVertexIterator`

Class representing an iterator over `SVertex` of a `ViewEdge` . An instance of an SVertexIterator can be obtained from a ViewEdge by calling verticesBegin() or verticesEnd().

__init__()

__init__(brother)

__init__(vertex, begin, previous_edge, next_edge, t)

Build an SVertexIterator using either the default constructor, copy constructor,

or the overloaded constructor that starts iteration from an SVertex object vertex.

PARAMETERS:

- **brother** (`SVertexIterator`) – An SVertexIterator object.
- **vertex** (`SVertex`) – The SVertex from which the iterator starts iteration.
- **begin** (`SVertex`) – The first SVertex of a ViewEdge.
- **previous_edge** (`FEdge`) – The previous FEdge coming to vertex.
- **next_edge** (`FEdge`) – The next FEdge going out from vertex.
- **t** (*float*) – The curvilinear abscissa at vertex.

object

The SVertex object currently pointed by this iterator.

TYPE:

`SVertex`

t

The curvilinear abscissa of the current point.

TYPE:
float

u

The point parameter at the current point in the 1D element (0 <= u <= 1).

TYPE:
float

class freestyle.types.**Stroke**

Class hierarchy: `Interface1D` > `Stroke`

Class to define a stroke. A stroke is made of a set of 2D vertices (`StrokeVertex`), regularly spaced out. This set of vertices defines the stroke's backbone geometry. Each of these stroke vertices defines the stroke's shape and appearance at this vertex position.

Stroke()

Stroke(brother)

Creates a `Stroke` using the default constructor or copy constructor

compute_sampling(n)

Compute the sampling needed to get N vertices. If the specified number of vertices is less than the actual number of vertices, the actual sampli
value is returned. (To remove Vertices, use the RemoveVertex() method of this class.)

PARAMETERS:
n (*int*) – The number of stroke vertices we eventually want in our Stroke.

RETURNS:
The sampling that must be used in the Resample(float) method.

RETURN TYPE:
float

insert_vertex(vertex, next)

Inserts the StrokeVertex given as argument into the Stroke before the point specified by next. The length and curvilinear abscissa are updated consequently.

PARAMETERS:

- **vertex** (`StrokeVertex`) – The StrokeVertex to insert in the Stroke.
- **next** (`StrokeVertexIterator`) – A StrokeVertexIterator pointing to the StrokeVertex before which vertex must be inserted.

remove_all_vertices()

Removes all vertices from the Stroke.

remove_vertex(vertex)

Removes the StrokeVertex given as argument from the Stroke. The length and curvilinear abscissa are updated consequently.

PARAMETERS:
vertex (`StrokeVertex`) – the StrokeVertex to remove from the Stroke.

resample(n)

resample(sampling)

Resamples the stroke so using one of two methods with the goal of creating a stroke with fewer points and the same shape.

PARAMETERS:

- **n** (*int*) – Resamples the stroke so that it eventually has N points. That means it is going to add N-vertices_size, where vertices_size is th
number of points we already have. If vertices_size >= N, no resampling is done.
- **sampling** (*float*) – Resamples the stroke with a given sampling value. If the sampling is smaller than the actual sampling value, no

sampling (*float*) – Resamples the Stroke with a given sampling value. If the sampling is smaller than the actual sampling value, no resampling is done.

stroke_vertices_begin(t=0.0)

Returns a StrokeVertexIterator pointing on the first StrokeVertex of the Stroke. One can specify a sampling value to re-sample the Stroke on the fly if needed.

PARAMETERS:

t (*float*) – The resampling value with which we want our Stroke to be resampled. If 0 is specified, no resampling is done.

RETURNS:

A StrokeVertexIterator pointing on the first StrokeVertex.

RETURN TYPE:

`StrokeVertexIterator`

stroke_vertices_end()

Returns a StrokeVertexIterator pointing after the last StrokeVertex of the Stroke.

RETURNS:

A StrokeVertexIterator pointing after the last StrokeVertex.

RETURN TYPE:

`StrokeVertexIterator`

stroke_vertices_size()

Returns the number of StrokeVertex constituting the Stroke.

RETURNS:

The number of stroke vertices.

RETURN TYPE:

int

update_length()

Updates the 2D length of the Stroke.

id

The Id of this Stroke.

TYPE:

`Id`

length_2d

The 2D length of the Stroke.

TYPE:

float

medium_type

The MediumType used for this Stroke.

TYPE:

`MediumType`

texture_id

The ID of the texture used to simulate th marks system for this Stroke.

TYPE:

int

tips

True if this Stroke uses a texture with tips, and false otherwise.

TYPE:

bool

class freestyle.types.**StrokeAttribute**

Class to define a set of attributes associated with a `StrokeVertex`. The attribute set stores the color, alpha and thickness values for a Stroke Vertex.

__init__()

__init__(brother)

__init__(red, green, blue, alpha, thickness_right, thickness_left)

__init__(attribute1, attribute2, t)

Creates a `StrokeAttribute` object using either a default constructor, copy constructor, overloaded constructor, or and interpolation constructor to interpolate between two `StrokeAttribute` objects.

PARAMETERS:

- **brother** (`StrokeAttribute`) – A StrokeAttribute object to be used as a copy constructor.
- **red** (*float*) – Red component of a stroke color.
- **green** (*float*) – Green component of a stroke color.
- **blue** (*float*) – Blue component of a stroke color.
- **alpha** (*float*) – Alpha component of a stroke color.
- **thickness_right** (*float*) – Stroke thickness on the right.
- **thickness_left** (*float*) – Stroke thickness on the left.
- **attribute1** (`StrokeAttribute`) – The first StrokeAttribute object.
- **attribute2** (`StrokeAttribute`) – The second StrokeAttribute object.
- **t** (*float*) – The interpolation parameter (0 <= t <= 1).

get_attribute_real(name)

Returns an attribute of float type.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

The attribute value.

RETURN TYPE:

float

get_attribute_vec2(name)

Returns an attribute of two-dimensional vector type.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

The attribute value.

RETURN TYPE:

`mathutils.Vector`

get_attribute_vec3(name)

Returns an attribute of three-dimensional vector type.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

The attribute value.

RETURN TYPE:

`mathutils.Vector`

has_attribute_real(name)

Checks whether the attribute name of float type is available.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

True if the attribute is available.

RETURN TYPE:

bool

has_attribute_vec2(name)

Checks whether the attribute name of two-dimensional vector type is available.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

True if the attribute is available.

RETURN TYPE:

bool

has_attribute_vec3(name)

Checks whether the attribute name of three-dimensional vector type is available.

PARAMETERS:

name (*str*) – The name of the attribute.

RETURNS:

True if the attribute is available.

RETURN TYPE:

bool

set_attribute_real(name, value)

Adds a user-defined attribute of float type. If there is no attribute of the given name, it is added. Otherwise, the new value replaces the old on

PARAMETERS:

- **name** (*str*) – The name of the attribute.
- **value** (*float*) – The attribute value.

set_attribute_vec2(name, value)

Adds a user-defined attribute of two-dimensional vector type. If there is no attribute of the given name, it is added. Otherwise, the new value replaces the old one.

PARAMETERS:

- **name** (*str*) – The name of the attribute.
- **value** (`mathutils.Vector` | tuple[float, float, float] | list[float]) – The attribute value.

set_attribute_vec3(name, value)

Adds a user-defined attribute of three-dimensional vector type. If there is no attribute of the given name, it is added. Otherwise, the new value replaces the old one.

PARAMETERS:

- **name** (*str*) – The name of the attribute.
- **value** (`mathutils.Vector` | tuple[float, float, float] | list[float]) – The attribute value as a 3D vector.

alpha

Alpha component of the stroke color.

TYPE:

float

color

RGB components of the stroke color.

TYPE:

`mathutils.Color`

thickness

Right and left components of the stroke thickness. The right (left) component is the thickness on the right (left) of the vertex when following the stroke.

TYPE:

`mathutils.Vector`

visible

The visibility flag. True if the StrokeVertex is visible.

TYPE:

bool

class freestyle.types.**StrokeShader**

Base class for stroke shaders. Any stroke shader must inherit from this class and overload the shade() method. A StrokeShader is designed to modify stroke attributes such as thickness, color, geometry, texture, blending mode, and so on. The basic way for this operation is to iterate over th stroke vertices of the `Stroke` and to modify the `StrokeAttribute` of each vertex. Here is a code example of such an iteration:

```
it = ioStroke.strokeVerticesBegin()
while not it.is_end:
    att = it.object.attribute
    ## perform here any attribute modification
    it.increment()
```

__init__()

Default constructor.

shade(stroke)

The shading method. Must be overloaded by inherited classes.

PARAMETERS:

stroke (`Stroke`) – A Stroke object.

name

The name of the stroke shader.

TYPE:

str

class freestyle.types.**StrokeVertex**

Class hierarchy: `Interface0D` > `CurvePoint` > `StrokeVertex`

Class to define a stroke vertex.

__init__()

__init__(brother)

__init__(first_vertex, second_vertex, t3d)

__init__(point)

__init__(svertex)

__init__(svertex, attribute)

Builds a `StrokeVertex` using the default constructor, copy constructor, from 2 `StrokeVertex` and an interpolation parameter, frc
a CurvePoint, from a SVertex, or a `SVertex` and a `StrokeAttribute` object.

PARAMETERS:

- **brother** (`StrokeVertex`) – A StrokeVertex object.
- **first_vertex** (`StrokeVertex`) – The first StrokeVertex.
- **second_vertex** (`StrokeVertex`) – The second StrokeVertex.
- **t3d** (*float*) – An interpolation parameter.
- **point** (`CurvePoint`) – A CurvePoint object.
- **svertex** (`SVertex`) – An SVertex object.
- **svertex** – An SVertex object.
- **attribute** (`StrokeAttribute`) – A StrokeAttribute object.

attribute

StrokeAttribute for this StrokeVertex.

TYPE:

`StrokeAttribute`

curvilinear_abscissa

Curvilinear abscissa of this StrokeVertex in the Stroke.

TYPE:

float

point

2D point coordinates.

TYPE:

`mathutils.Vector`

stroke_length

Stroke length (it is only a value retained by the StrokeVertex, and it won't change the real stroke length).

TYPE:

float

u

Curvilinear abscissa of this StrokeVertex in the Stroke.

TYPE:

float

class freestyle.types.**StrokeVertexIterator**

Class hierarchy: `Iterator` > `StrokeVertexIterator`

Class defining an iterator designed to iterate over the `StrokeVertex` of a `Stroke`. An instance of a StrokeVertexIterator can be obtained from a Stroke by calling iter(), stroke_vertices_begin() or stroke_vertices_begin(). It is iterating over the same vertices as an `Interface0DIterator`. The difference resides in the object access: an Interface0DIterator only allows access to an Interface0D while on might need to access the specialized StrokeVertex type. In this case, one should use a StrokeVertexIterator. To call functions of the UnaryFuntion(type, a StrokeVertexIterator can be converted to an Interface0DIterator by by calling Interface0DIterator(it).

__init__()

__init__(brother)

Creates a `StrokeVertexIterator` using either the default constructor or the copy constructor.

PARAMETERS:

brother (`StrokeVertexIterator`) – A StrokeVertexIterator object.

decremented()

Returns a copy of a decremented StrokeVertexIterator.

RETURNS:

A StrokeVertexIterator pointing the previous StrokeVertex.

RETURN TYPE:

`StrokeVertexIterator`

incremented()

Returns a copy of an incremented StrokeVertexIterator.

RETURNS:

A StrokeVertexIterator pointing the next StrokeVertex.

RETURN TYPE:

`StrokeVertexIterator`

reversed()

Returns a StrokeVertexIterator that traverses stroke vertices in the reversed order.

RETURNS:

A StrokeVertexIterator traversing stroke vertices backward.

RETURN TYPE:

`StrokeVertexIterator`

at_last

True if the iterator points to the last valid element. For its counterpart (pointing to the first valid element), use it.is_begin.

TYPE:

bool

object

The StrokeVertex object currently pointed to by this iterator.

TYPE:

`StrokeVertex`

t

The curvilinear abscissa of the current point.

The curvilinear abscissa of the current point.

TYPE:

float

u

The point parameter at the current point in the stroke (0 <= u <= 1).

TYPE:

float

class freestyle.types.**TVertex**

Class hierarchy: `Interface0D` > `ViewVertex` > `TVertex`

Class to define a T vertex, i.e. an intersection between two edges. It points towards two SVertex and four ViewEdges. Among the ViewEdges, tw are front and the other two are back. Basically a front edge hides part of a back edge. So, among the back edges, one is of invisibility N and the other of invisibility N+1.

__init__()

Default constructor.

get_mate(viewedge)

Returns the mate edge of the ViewEdge given as argument. If the ViewEdge is frontEdgeA, frontEdgeB is returned. If the ViewEdge is frontEdgeB, frontEdgeA is returned. Same for back edges.

PARAMETERS:

viewedge (`ViewEdge`) – A ViewEdge object.

RETURNS:

The mate edge of the given ViewEdge.

RETURN TYPE:

`ViewEdge`

get_svertex(fedge)

Returns the SVertex (among the 2) belonging to the given FEdge.

PARAMETERS:

fedge (`FEdge`) – An FEdge object.

RETURNS:

The SVertex belonging to the given FEdge.

RETURN TYPE:

`SVertex`

back_svertex

The SVertex that is further away from the viewpoint.

TYPE:

`SVertex`

front_svertex

The SVertex that is closer to the viewpoint.

TYPE:

`SVertex`

id

The Id of this TVertex.

TYPE:

`Id`

class freestyle.types.**UnaryFunction0D**

Base class for Unary Functions (functors) working on `Interface0DIterator`. A unary function will be used by invoking __call__() on an Interface0DIterator. In Python, several different subclasses of UnaryFunction0D are used depending on the types of functors' return values. For example, you would inherit from a `UnaryFunction0DDouble` if you wish to define a function that returns a double value. Available UnaryFunction0D subclasses are:

- `UnaryFunction0DDouble`
- `UnaryFunction0DEdgeNature`
- `UnaryFunction0DFloat`
- `UnaryFunction0DId`
- `UnaryFunction0DMaterial`
- `UnaryFunction0DUnsigned`
- `UnaryFunction0DVec2f`
- `UnaryFunction0DVec3f`
- `UnaryFunction0DVectorViewShape`
- `UnaryFunction0DViewShape`

name

The name of the unary 0D function.

TYPE:

str

class freestyle.types.**UnaryFunction0DDouble**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DDouble`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a float value.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DEdgeNature**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DEdgeNature`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a `Nature` object.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DFloat**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DFloat`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a float value.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DId**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DId`

Base class for unary functions (functors) that work on `Interface0DIterator` and return an `Id` object.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DMaterial**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DMaterial`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a `Material` object.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DUnsigned**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DUnsigned`

Base class for unary functions (functors) that work on `Interface0DIterator` and return an int value.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DVec2f**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DVec2f`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a 2D vector.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DVec3f**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DVec3f`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a 3D vector.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DVectorViewShape**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DVectorViewShape`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a list of `ViewShape` objects.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction0DViewShape**

Class hierarchy: `UnaryFunction0D` > `UnaryFunction0DViewShape`

Base class for unary functions (functors) that work on `Interface0DIterator` and return a `ViewShape` object.

__init__()

Default constructor.

class freestyle.types.**UnaryFunction1D**

Base class for Unary Functions (functors) working on `Interface1D`. A unary function will be used by invoking __call__() on an Interface1D. Python, several different subclasses of UnaryFunction1D are used depending on the types of functors' return values. For example, you would inher from a `UnaryFunction1DDouble` if you wish to define a function that returns a double value. Available UnaryFunction1D subclasses are:

- `UnaryFunction1DDouble`
- `UnaryFunction1DEdgeNature`
- `UnaryFunction1DFloat`

- `UnaryFunction1DUnsigned`
- `UnaryFunction1DVec2f`
- `UnaryFunction1DVec3f`
- `UnaryFunction1DVectorViewShape`
- `UnaryFunction1DVoid`

name

The name of the unary 1D function.

TYPE:

str

class freestyle.types.**UnaryFunction1DDouble**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DDouble`

Base class for unary functions (functors) that work on `Interface1D` and return a float value.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DEdgeNature**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DEdgeNature`

Base class for unary functions (functors) that work on `Interface1D` and return a `Nature` object.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DFloat**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DFloat`

Base class for unary functions (functors) that work on `Interface1D` and return a float value.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DUnsigned**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DUnsigned`

Base class for unary functions (functors) that work on `Interface1D` and return an int value.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DVec2f**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DVec2f`

Base class for unary functions (functors) that work on `Interface1D` and return a 2D vector.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DVec3f**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DVec3f`

Base class for unary functions (functors) that work on `Interface1D` and return a 3D vector.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DVectorViewShape**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DVectorViewShape`

Base class for unary functions (functors) that work on `Interface1D` and return a list of `ViewShape` objects.

__init__()

__init__(integration_type)

Builds a unary 1D function using the default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryFunction1DVoid**

Class hierarchy: `UnaryFunction1D` > `UnaryFunction1DVoid`

Base class for unary functions (functors) working on `Interface1D` .

__init__()

__init__(integration_type)

Builds a unary 1D function using either a default constructor or the integration method given as an argument.

PARAMETERS:

integration_type (`IntegrationType`) – An integration method.

integration_type

The integration method.

TYPE:

`IntegrationType`

class freestyle.types.**UnaryPredicate0D**

Base class for unary predicates that work on `Interface0DIterator` . A UnaryPredicate0D is a functor that evaluates a condition on an Interface0DIterator and returns true or false depending on whether this condition is satisfied or not. The UnaryPredicate0D is used by invoking its __call__() method. Any inherited class must overload the __call__() method.

__init__()

Default constructor.

__call__(it)

Must be overload by inherited classes.

PARAMETERS:

it (`Interface0DIterator`) – The Interface0DIterator pointing onto the Interface0D at which we wish to evaluate the predicate

RETURNS:

True if the condition is satisfied, false otherwise.

RETURN TYPE:

bool

name

The name of the unary 0D predicate.

TYPE:

str

class freestyle.types.**UnaryPredicate1D**

Base class for unary predicates that work on `Interface1D`. A UnaryPredicate1D is a functor that evaluates a condition on a Interface1D and returns true or false depending on whether this condition is satisfied or not. The UnaryPredicate1D is used by invoking its __call__() method. Any inherited class must overload the __call__() method.

__init__()

Default constructor.

__call__(inter)

Must be overload by inherited classes.

PARAMETERS:

inter (`Interface1D`) – The Interface1D on which we wish to evaluate the predicate.

RETURNS:

True if the condition is satisfied, false otherwise.

RETURN TYPE:

bool

name

The name of the unary 1D predicate.

TYPE:

str

class freestyle.types.**ViewEdge**

Class hierarchy: `Interface1D` > `ViewEdge`

Class defining a ViewEdge. A ViewEdge in an edge of the image graph. it connects two `ViewVertex` objects. It is made by connecting a set c
FEdges.

__init__()

__init__(brother)

Builds a `ViewEdge` using the default constructor or the copy constructor.

PARAMETERS:

brother (`ViewEdge`) – A ViewEdge object.

update_fedges()

Sets Viewedge to this for all embedded fedges.

chaining_time_stamp

The time stamp of this ViewEdge.

TYPE:

int

first_fedge

The first FEdge that constitutes this ViewEdge.

TYPE:
`FEdge`

first_viewvertex

The first ViewVertex.

TYPE:
`ViewVertex`

id

The Id of this ViewEdge.

TYPE:
`Id`

is_closed

True if this ViewEdge forms a closed loop.

TYPE:
bool

last_fedge

The last FEdge that constitutes this ViewEdge.

TYPE:
`FEdge`

last_viewvertex

The second ViewVertex.

TYPE:
`ViewVertex`

nature

The nature of this ViewEdge.

TYPE:
`Nature`

occludee

The shape that is occluded by the ViewShape to which this ViewEdge belongs to. If no object is occluded, this property is set to None.

TYPE:
`ViewShape`

qi

The quantitative invisibility.

TYPE:
int

viewshape

The ViewShape to which this ViewEdge belongs to.

TYPE:
`ViewShape`

class freestyle.types.**ViewEdgeIterator**

Class hierarchy: `Iterator` > `ViewEdgeIterator`

Base class for iterators over ViewEdges of the `ViewMap` Graph. Basically the increment() operator of this class should be able to take the decision of "where" (on which ViewEdge) to go when pointing on a given ViewEdge.

__init__(begin=None, orientation=True)

__init__(brother)

Builds a ViewEdgeIterator from a starting ViewEdge and its orientation or the copy constructor.

PARAMETERS:

- **begin** (`ViewEdge` | None) – The ViewEdge from where to start the iteration.
- **orientation** (*bool*) – If true, we'll look for the next ViewEdge among the ViewEdges that surround the ending ViewVertex of begin. If false, we'll search over the ViewEdges surrounding the ending ViewVertex of begin.
- **brother** (`ViewEdgeIterator`) – A ViewEdgeIterator object.

change_orientation()

Changes the current orientation.

begin

The first ViewEdge used for the iteration.

TYPE:

`ViewEdge`

current_edge

The ViewEdge object currently pointed by this iterator.

TYPE:

`ViewEdge`

object

The ViewEdge object currently pointed by this iterator.

TYPE:

`ViewEdge`

orientation

The orientation of the pointed ViewEdge in the iteration. If true, the iterator looks for the next ViewEdge among those ViewEdges that surrou the ending ViewVertex of the "begin" ViewEdge. If false, the iterator searches over the ViewEdges surrounding the ending ViewVertex of the "begin" ViewEdge.

TYPE:

bool

class freestyle.types.**ViewMap**

Class defining the ViewMap.

__init__()

Default constructor.

get_closest_fedge(x, y)

Gets the FEdge nearest to the 2D point specified as arguments.

PARAMETERS:

- **x** (*float*) – X coordinate of a 2D point.

- **y** (*float*) – Y coordinate of a 2D point.

RETURNS:

The FEdge nearest to the specified 2D point.

RETURN TYPE:

`FEdge`

get_closest_viewedge(x, y)

Gets the ViewEdge nearest to the 2D point specified as arguments.

PARAMETERS:

- **x** (*float*) – X coordinate of a 2D point.
- **y** (*float*) – Y coordinate of a 2D point.

RETURNS:

The ViewEdge nearest to the specified 2D point.

RETURN TYPE:

`ViewEdge`

scene_bbox

The 3D bounding box of the scene.

TYPE:

`BBox`

class freestyle.types.**ViewShape**

Class gathering the elements of the ViewMap (i.e., `ViewVertex` and `ViewEdge`) that are issued from the same input shape.

__init__()

__init__(brother)

__init__(sshape)

Builds a `ViewShape` using the default constructor, copy constructor, or from a `SShape` .

PARAMETERS:

- **brother** (`ViewShape`) – A ViewShape object.
- **sshape** (`SShape`) – An SShape object.

add_edge(edge)

Adds a ViewEdge to the list of ViewEdge objects.

PARAMETERS:

edge (`ViewEdge`) – A ViewEdge object.

add_vertex(vertex)

Adds a ViewVertex to the list of the ViewVertex objects.

PARAMETERS:

vertex (`ViewVertex`) – A ViewVertex object.

edges

The list of ViewEdge objects contained in this ViewShape.

TYPE:

List of `ViewEdge`

id

The Id of this ViewShape.

TYPE:
`Id`

library_path

The library path of the ViewShape.

TYPE:
str, or None if the ViewShape is not part of a library

name

The name of the ViewShape.

TYPE:
str

sshape

The SShape on top of which this ViewShape is built.

TYPE:
`SShape`

vertices

The list of ViewVertex objects contained in this ViewShape.

TYPE:
List of `ViewVertex`

class freestyle.types.**ViewVertex**

Class hierarchy: `Interface0D` > `ViewVertex`

Class to define a view vertex. A view vertex is a feature vertex corresponding to a point of the image graph, where the characteristics of an edge (e.g., nature and visibility) might change. A `ViewVertex` can be of two kinds: A `TVertex` when it corresponds to the intersection between two ViewEdges or a `NonTVertex` when it corresponds to a vertex of the initial input mesh (it is the case for vertices such as corners for example). Thus, this class can be specialized into two classes, the `TVertex` class and the `NonTVertex` class.

edges_begin()

Returns an iterator over the ViewEdges that goes to or comes from this ViewVertex pointing to the first ViewEdge of the list. The orientedViewEdgeIterator allows to iterate in CCW order over these ViewEdges and to get the orientation for each ViewEdge (incoming/outgoing).

RETURNS:
An orientedViewEdgeIterator pointing to the first ViewEdge.

RETURN TYPE:
`orientedViewEdgeIterator`

edges_end()

Returns an orientedViewEdgeIterator over the ViewEdges around this ViewVertex, pointing after the last ViewEdge.

RETURNS:
An orientedViewEdgeIterator pointing after the last ViewEdge.

RETURN TYPE:
`orientedViewEdgeIterator`

edges_iterator(edge)

Returns an orientedViewEdgeIterator pointing to the ViewEdge given as argument.

Returns an orientedViewEdgeIterator pointing to the ViewEdge given as argument.

PARAMETERS:

edge (`ViewEdge`) – A ViewEdge object.

RETURNS:

An orientedViewEdgeIterator pointing to the given ViewEdge.

RETURN TYPE:

`orientedViewEdgeIterator`

nature

The nature of this ViewVertex.

TYPE:

`Nature`

class freestyle.types.**orientedViewEdgeIterator**

Class hierarchy: `Iterator` > `orientedViewEdgeIterator`

Class representing an iterator over oriented ViewEdges around a `ViewVertex` . This iterator allows a CCW iteration (in the image plane). An instance of an orientedViewEdgeIterator can only be obtained from a ViewVertex by calling edges_begin() or edges_end().

__init__()

__init__(iBrother)

Creates an `orientedViewEdgeIterator` using either the default constructor or the copy constructor.

PARAMETERS:

iBrother (`orientedViewEdgeIterator`) – An orientedViewEdgeIterator object.

object

The oriented ViewEdge (i.e., a tuple of the pointed ViewEdge and a boolean value) currently pointed to by this iterator. If the boolean value is true, the ViewEdge is incoming.

TYPE:

(`ViewEdge` , bool)