bentley_ottmann

Release 0.9.0

Azat Ibrakov

Sep 24, 2020

CONTENTS

Note: If object is not listed in documentation it should be considered as implementation detail that can change and should not be relied upon.

`bentley_ottmann.planar.`**`edges_intersect`**(*contour: Sequence[Tuple[numbers.Real, numbers.Real]], *, accurate: bool = True, validate: bool = True*) → bool

Checks if polygonal contour has self-intersection.

Based on Shamos-Hoey algorithm.

Time complexity: `O(len(contour) * log len(contour))`

Memory complexity: `O(len(contour))`

Reference: https://en.wikipedia.org/wiki/Sweep_line_algorithm

Parameters

- **`contour`** – contour to check.
- **`accurate`** – flag that tells whether to use slow but more accurate arithmetic for floating point numbers.
- **`validate`** – flag that tells whether to check contour for degeneracies and raise an exception in case of occurrence.

Raises `ValueError` – if `validate` flag is set and contour is degenerate.

Returns true if contour is self-intersecting, false otherwise.

Note: Consecutive equal vertices like `(2., 0.)` in

```
[(0., 0.), (2., 0.), (2., 0.), (2., 2.)]
```

will be considered as self-intersection, if you don't want them to be treated as such – filter out before passing as argument.

```
>>> edges_intersect([(0., 0.), (2., 0.), (2., 2.)])
False
>>> edges_intersect([(0., 0.), (2., 0.), (1., 0.)])
True
```

`bentley_ottmann.planar.`**`segments_intersect`**(*segments: Sequence[Tuple[Tuple[numbers.Real, numbers.Real], Tuple[numbers.Real, numbers.Real]]], *, accurate: bool = True, validate: bool = True*) → bool

Checks if segments have at least one intersection.

Based on Shamos-Hoey algorithm.

Time complexity: `O(len(segments) * log len(segments))`

Memory complexity: `O(len(segments))`

Reference: https://en.wikipedia.org/wiki/Sweep_line_algorithm

Parameters

- **segments** – sequence of segments.
- **accurate** – flag that tells whether to use slow but more accurate arithmetic for floating point numbers.
- **validate** – flag that tells whether to check segments for degeneracies and raise an exception in case of occurrence.

Raises **ValueError** – if `validate` flag is set and degenerate segment found.

Returns true if segments intersection found, false otherwise.

```
>>> segments_intersect([])
False
>>> segments_intersect([((0., 0.), (2., 2.))])
False
>>> segments_intersect([((0., 0.), (2., 0.)), ((0., 2.), (2., 2.))])
False
>>> segments_intersect([((0., 0.), (2., 2.)), ((0., 0.), (2., 2.))])
True
>>> segments_intersect([((0., 0.), (2., 2.)), ((2., 0.), (0., 2.))])
True
```

`bentley_ottmann.planar.`**`segments_cross_or_overlap`**(*segments: Sequence[Tuple[Tuple[numbers.Real, numbers.Real], Tuple[numbers.Real, numbers.Real]]]*, ***, *accurate: bool = True*, *validate: bool = True*) → bool

Checks if at least one pair of segments crosses or overlaps.

Based on Shamos-Hoey algorithm.

Time complexity: `O((len(segments) + len(intersections)) * log len(segments))`

Memory complexity: `O(len(segments))`

Reference: https://en.wikipedia.org/wiki/Sweep_line_algorithm

Parameters

- **segments** – sequence of segments.
- **accurate** – flag that tells whether to use slow but more accurate arithmetic for floating point numbers.
- **validate** – flag that tells whether to check segments for degeneracies and raise an exception in case of occurrence.

Raises **ValueError** – if `validate` flag is set and degenerate segment found.

Returns true if segments overlap or cross found, false otherwise.

```
>>> segments_cross_or_overlap([])
False
>>> segments_cross_or_overlap([((0., 0.), (2., 2.))])
False
>>> segments_cross_or_overlap([((0., 0.), (2., 0.)), ((0., 2.), (2., 2.))])
False
>>> segments_cross_or_overlap([((0., 0.), (2., 2.)), ((0., 0.), (2., 2.))])
True
```

(continues on next page)

(continued from previous page)

```
>>> segments_cross_or_overlap([((0., 0.), (2., 2.)), ((2., 0.), (0., 2.))])
True
```

`bentley_ottmann.planar.`**`segments_intersections`**(*segments: Sequence[Tuple[Tuple[numbers.Real, numbers.Real], Tuple[numbers.Real, numbers.Real]]], *, accurate: bool = True, validate: bool = True*) → Dict[Tuple[numbers.Real, numbers.Real], Set[Tuple[int, int]]]

Returns mapping between intersection points and corresponding segments indices.

Based on Bentley-Ottmann algorithm.

Time complexity: `O((len(segments) + len(intersections)) * log len(segments))`

Memory complexity: `O(len(segments) + len(intersections))`

Reference: https://en.wikipedia.org/wiki/Bentley%E2%80%93Ottmann_algorithm

```
>>> segments_intersections([])
{}
>>> segments_intersections([((0., 0.), (2., 2.))])
{}
>>> segments_intersections([((0., 0.), (2., 0.)), ((0., 2.), (2., 2.))])
{}
>>> segments_intersections([((0., 0.), (2., 2.)), ((0., 0.), (2., 2.))])
{(0.0, 0.0): {(0, 1)}, (2.0, 2.0): {(0, 1)}}
>>> segments_intersections([((0., 0.), (2., 2.)), ((2., 0.), (0., 2.))])
{(1.0, 1.0): {(0, 1)}}
```

Parameters

- **`segments`** – sequence of segments.
- **`accurate`** – flag that tells whether to use slow but more accurate arithmetic for floating point numbers.
- **`validate`** – flag that tells whether to check segments for degeneracies and raise an exception in case of occurrence.

Raises `ValueError` – if `validate` flag is set and degenerate segment found.

Returns mapping between intersection points and corresponding segments indices.

PYTHON MODULE INDEX

b

INDEX