Foronoi

Release 1.0.3

Jeroen van Hoof

Apr 08, 2021

CONTENTS

Foronoi is a Python implementation of the Fortune's algorithm based on the description of "Computational Geometry: Algorithms and Applications" by de Berg et al.

This algorithm is a sweep line algorithm that scans top down over the cell points. Every time a new cell point is scanned, a corresponding parabola (arc) is added. The intersections of this arc with other arcs are so-called "breakpoints". These breakpoints trace out the borders between two cell points. At the same time when an arc is added, a check is done to see if this arc will converge with the two arcs on the left or the arcs on the right. If that's the case, it will insert a so-called circle-event which causes a new vertex (i.e. a cross-way between edges) to be created in the middle of the circle.

If you would like to play around with a simple example to get a better understanding, I recommend visiting .

The algorithm keeps track of the status (everything above the line is handled) in a so-called status-structure. This status-structure is a balanced binary search tree that keeps track of the positions of the arcs (in its leaf nodes) and the breakpoints (in its internal nodes). This data structure allows for fast look-up times, so that the entire algorithm can run in *O(n log n)* time.

This implementation includes some additional features to the standard algorithm. For example, this implementation is able to clip the diagram to a bounding box in different shapes. And it will clean up zero-length edges that occur in edge-cases where two events happen at the same time so that it is more practical to use.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

1.1.1 Via pip

```
pip install foronoi
```

1.1.2 Manual

First, clone the repository and then install the package.

```
git clone https://github.com/Yatoom/voronoi.git
cd voronoi
python setup.py install
```

1.2 Quick start

1.2.1 Basic example

This is a basic example that quickly constructs and visualizes a voronoi graph.

```
from foronoi import Voronoi, Polygon, Visualizer, Point, VoronoiObserver
from foronoi.graph import HalfEdge, Vertex

# Define some points (a.k.a sites or cell points)
points = [
    (2.5, 2.5), (4, 7.5), (7.5, 2.5), (6, 7.5), (4, 4), (3, 3), (6, 3)
]

# Define a bounding box / polygon
polygon = Polygon([
    (2.5, 10), (5, 10), (10, 5), (10, 2.5), (5, 0), (2.5, 0), (0, 2.5), (0, 5)
])

# Initialize the algorithm
v = Voronoi(polygon)

# Optional: visualize the voronoi diagram at every step.
```

(continues on next page)

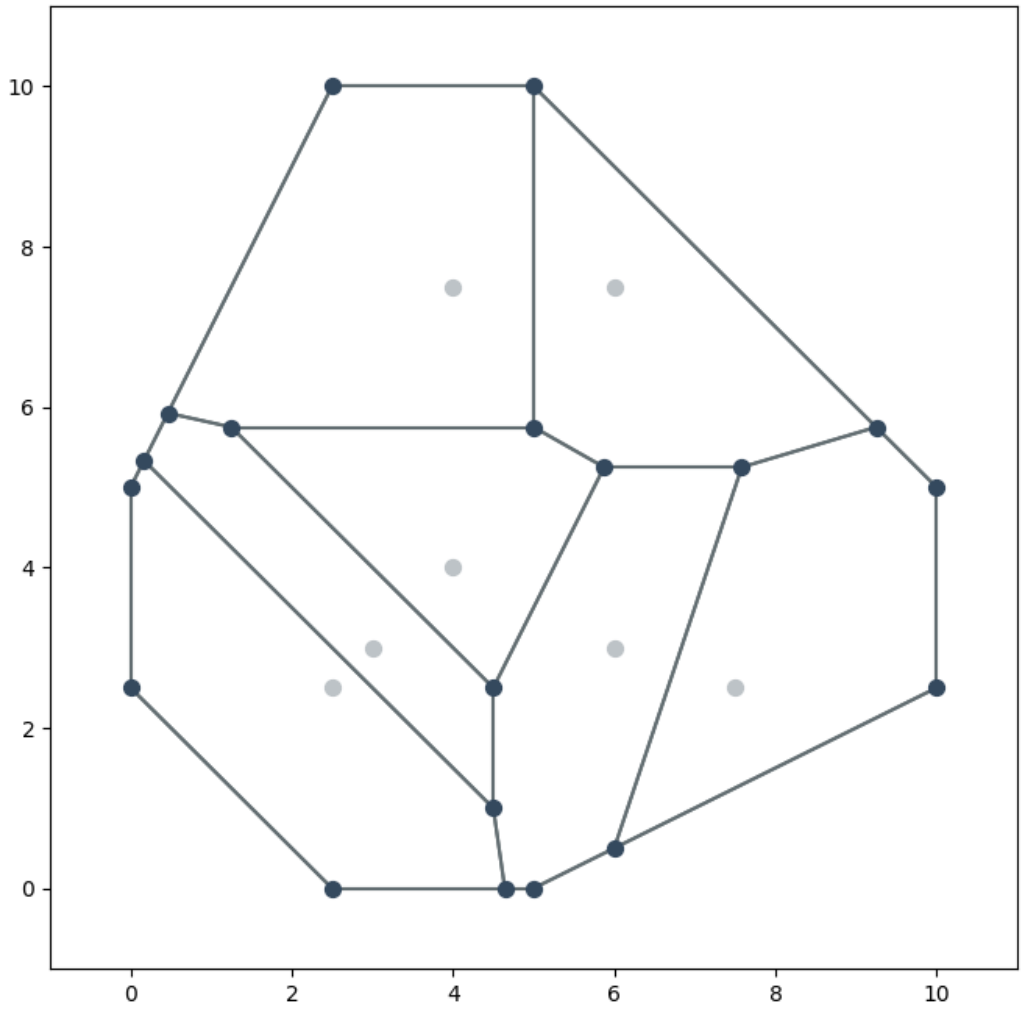
(continued from previous page)

```
# You can find more information in the observers.py example file
# v.attach_observer(
#     VoronoiObserver()
# )

# Create the Voronoi diagram
v.create_diagram(points=points)

# Visualize the Voronoi diagram
Visualizer(v) \
    .plot_sites(show_labels=False) \
    .plot_edges(show_labels=False) \
    .plot_vertices() \
    .show()
```

Result:



1.2.2 Properties

Below are some examples of how to retrieve certain components and properties from the voronoi graph and what kind of operations are possible.

```
from typing import List

# Some examples of how to access properties from the Voronoi diagram:
edges: List[HalfEdge] = v.edges                    # A list of all edges
vertices: List[Vertex] = v.vertices                # A list of all vertices
sites: List[Point] = v.sites                       # A list of all cell points (a.k.a.
↪ sites)
```

(continues on next page)

(continued from previous page)

```
edge, vertex, site = edges[0], vertices[0], sites[0]

# Edge operations
origin: Vertex = edge.origin                # The vertex in which the edge originates
target: Vertex = edge.twin.origin           # The twin is the edge that goes in the␣
↪other direction
target_alt: Vertex = edge.target            # Same as above, but more convenient
twin: HalfEdge = edge.twin                  # Get the twin of this edge
next_edge: HalfEdge = edge.next             # Get the next edge
prev_edge: HalfEdge = edge.twin.next        # Get the previous edge
prev_alt: HalfEdge = edge.prev              # Same as above, but more convenient

# Site operations
size: float = site.area()                   # The area of the cell
borders: List[HalfEdge] = site.borders()    # A list of all the borders that surround␣
↪this cell point
vertices: List[Vertex] = site.vertices()    # A list of all the vertices around this␣
↪cell point
site_x: float = site.x                      # X-coordinate of the site
site_xy: [float, float] = site.xy           # (x, y)-coordinates of the site
first_edge: HalfEdge = site.first_edge      # Points to the first edge that is part of␣
↪the border around the site

# Vertex operations
connected_edges: List[HalfEdge] = vertex.connected_edges  # A list of all edges that␣
↪are connected to this vertex
vertex_x: float = vertex.x                                # x-coordinate of the vertex
vertex_xy: [float, float] = vertex.xy                     # (x, y)-coordinates of the␣
↪vertex
```

1.2.3 Observers

Observers allow you to observe the state of the algorithm and visualize components during the construction of the voronoi graph. Below you can see an example where we attach an observer that visualizes the voronoi graph at every step (event).

```
import os

from foronoi import Polygon, Voronoi, VoronoiObserver
from foronoi.visualization import Presets

# Define some points (a.k.a sites or cell points)
points = [
    (2.5, 2.5), (4, 7.5), (7.5, 2.5), (6, 7.5), (4, 4), (3, 3), (6, 3)
]

# Define a bounding box / polygon
polygon = Polygon([
    (2.5, 10), (5, 10), (10, 5), (10, 2.5), (5, 0), (2.5, 0), (0, 2.5), (0, 5)
])

# Initialize the algorithm
v = Voronoi(polygon)

# Attach a Voronoi observer that visualizes the Voronoi diagram every step
```

(continues on next page)

(continued from previous page)

```
v.attach_observer(
    VoronoiObserver(

        # Settings to pass into the visualizer's plot_all() method.
        # - By default, the observer uses a set of minimalistic presets
        #   that are useful for visualizing during construction, clipping
        #   and the final result.
        # - The settings below will update the default presets used by the
        #   observer. For example, by default, the arc_labels are not shown,
        #   but below we can enable the arc labels. Other parameters can be
        #   found in the visualizer's plot_all() method.
        settings=dict(arc_labels=True, site_labels=True),

        # Callback that saves the figure every step
        # If no callback is provided, it will simply display the figure in
        # a matplotlib window
        callback=lambda observer, figure: figure.savefig(
            f"output/voronoi/{observer.n_messages:02d}.png"
        ),

        visualize_steps=True             # Default = True
        visualize_before_clipping=True   # Default = False
        visualize_result=True            # Default = True
    )
)

# Create the output directory if it doesn't exist
if not os.path.exists("output"):
    os.mkdir("output")

if not os.path.exists("output/voronoi/"):
    os.mkdir("output/voronoi/")

# Create the Voronoi diagram
v.create_diagram(points=points)
```

Slideshow of images in *output/voronoi/*:

1.3 Public classes

1.3.1 Algorithm

class foronoi.algorithm.**Algorithm**(*bounding_poly: Optional[foronoi.graph.polygon.Polygon] = None*, *remove_zero_length_edges=True*)

A Python implementation of Fortune's algorithm based on the description of "Computational Geometry: Algorithms and Applications" by de Berg et al.

Parameters

- **bounding_poly** (Polygon) – The bounding box or bounding polygon around the voronoi diagram
- **remove_zero_length_edges** (*bool*) – Removes zero length edges and combines vertices with the same location into one

bounding_poly
The bounding box (or polygon) around the edge

Type *Polygon*

event_queue
Event queue for upcoming site and circle events

Type PriorityQueue

status_tree
The status structure is a data structure that stores the relevant situation at the current position of the sweep line. This attribute points to the root of the balanced binary search tree that functions as a status structure which represents the beach line as a balanced binary search tree.

Type Node

sweep_line
The y-coordinate

Type Decimal

arcs
List of arcs

Type list(*foronoi.nodes.Arc*)

sites
List of points

Type list(*foronoi.graph.Point*)

vertices
List of vertices

Type list(*foronoi.graph.Vertex*)

clean_up_zero_length_edges()
Removes zero length edges and vertices with the same coordinate that are produced when two site-events happen at the same time.

create_diagram(*points: list*)
Create the Voronoi diagram.

The overall structure of the algorithm is as follows.

1. Initialize the event queue *event_queue* with all site events, initialize an empty status structure *status_tree* and an empty doubly-connected edge list *D*.
2. **while** *event_queue* is not empty.
3. **do** Remove the event with largest *y*-coordinate from *event_queue*.
4. **if** the event is a site event, occurring at site *point*
5. **then** *handle_site_event()*
6. **else** *handle_circle_event()*
7. The internal nodes still present in *status_tree* correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box (or polygon) that contains all vertices of bounding box by updating the doubly-connected edge list appropriately.
8. **If** *remove_zero_length_edges* is true.

9. Call *clean_up_zero_length_edges()* which removes zero length edges and combines vertices with the same location into one.

Parameters **points** (*list(Point)*) – A set of point sites in the plane.

Returns

Return type Output. The Voronoi diagram *Vor(P)* given inside a bounding box in a doublyconnected edge list *D*.

handle_circle_event(*event:* foronoi.events.circle_event.CircleEvent)

Handle a circle event.

1. Delete the leaf that represents the disappearing arc from *status_tree*. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on *status_tree* if necessary. Delete all circle events involving from *event_queue*; these can be found using the pointers from the predecessor and the successor of in *status_tree*. (The circle event where is the middle arc is currently being handled, and has already been deleted from *event_queue*.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list `D` storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into *event_queue*. and set pointers between the new circle event in *event_queue* and the corresponding leaf of *status_tree*. Do the same for the triple where the former right neighbor is the middle arc.

Parameters **event** –

handle_site_event(*event:* foronoi.events.site_event.SiteEvent)

Handle a site event.

1. Let `point_i = event.point`. If *status_tree* is empty, insert `point_i` into it (so that *status_tree* consists of a single leaf storing `point_i`) and return. Otherwise, continue with steps 2– 5.
2. Search in *status_tree* for the arc vertically above `point_i`. If the leaf representing has a pointer to a circle event in *event_queue*, then this circle event is a false alarm and it must be deleted from *status_tree*.
3. Replace the leaf of *status_tree* that represents with a subtree having three leaves. The middle leaf stores the new site `point_i` and the other two leaves store the site `point_j` that was originally stored with . Store the breakpoints (`point_j`, `point_i`) and (`point_i`, `point_j`) representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on *status_tree* if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating the faces for `point_i` and `point_j`, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for pi is the left arc to see if the breakpoints converge. If so, insert the circle event into *status_tree* and add pointers between the node in *status_tree* and the node in *event_queue*. Do the same for the triple where the new arc is the right arc.

Parameters **event** (`SiteEvent`) – The site event to handle.

initialize(*points*)
Initialize the event queue *event_queue* with all site events.

Parameters **points** (*list(Point)*) – The list of cell points to initialize

Returns **event_queue** – Event queue for upcoming site and circle events

Return type PriorityQueue

1.3.2 BoundingBox

class foronoi.graph.bounding_box.**BoundingBox**(*left_x*, *right_x*, *bottom_y*, *top_y*)
Convenience method to create a bounding box. Extends *foronoi.graph.Polygon*.

Parameters

- **left_x** (*float*) – The x-coordinate of the left border
- **right_x** (*float*) – The x-coordinate of the right border
- **bottom_y** (*float*) – The y-coordinate of the bottom border
- **top_y** (*float*) – The y-coordinate of the top border

1.3.3 Coordinate

class foronoi.graph.**Coordinate**(*x=None*, *y=None*)
A point in 2D space

Parameters

- **x** (*float*) – The x-coordinate
- **y** (*float*) – The y-coordinate

property x
Get the x-coordinate as float

Returns **x** – The x-coordinate

Return type float

property xd
Get the x-coordinate as Decimal

Returns **x** – The x-coordinate

Return type Decimal

property xy
Get a (x, y) tuple

Parameters **xy** (*(float, float)*) – A tuple of the (x, y)-coordinate

property y
Get the y-coordinate as float

Returns **y** – The y-coordinate

Return type float

property yd
Get the y-coordinate as Decimal

Returns **y** – The y-coordinate

Return type Decimal

1.3.4 HalfEdge

class foronoi.graph.**HalfEdge**(*incident_point*, *twin=None*, *origin=None*)

Edges are normally treated as undirected and shared between faces. However, for some tasks (such as simplifying or cleaning geometry) it is useful to view faces as each having their own edges. You can think of this as splitting each shared undirected edge along its length into two half edges. (Boundary edges of course will only have one "half-edge".) Each half-edge is directed (it has a start vertex and an end vertex).

The half-edge properties let you quickly find a half-edge's source and destination vertex, the next half-edge, get the other half-edge from the same edge, find all half-edges sharing a given point, and other manipulations.

Examples

Get the half-edge's source

```
>>> edge.origin
```

Get the half-edge's destination

```
>>> edge.target # or edge.twin.origin
```

Get the previous and next half-edge

```
>>> edge.prev
>>> edge.next
```

Get the other half-edge from the same edge

```
>>> edge.twin
```

Find all half-edges sharing a given point

```
>>> edge.origin.connected_edges
```

Parameters

- **incident_point** (Point) – The cell point of which this edge is the border
- **twin** (HalfEdge) – The other half-edge from the same edge
- **origin** (Breakpoint *or* Vertex) – The origin of the half edge. Can be a Breakpoint or a Vertex during construction, and only Vertex when the diagram is finished.

origin

Pointer to the origin. Can be breakpoint or vertex.

Type Breakpoint or *Vertex*

next

Pointer to the next edge

Type *HalfEdge*

prev
Pointer to the previous edge

Type *HalfEdge*

delete()
Delete this half edge by pointing the previous edge to the next, and removing it from the origin's connected edges list.

get_origin(*y=None*, *max_y=None*)
Get the coordinates of the edge's origin. During construction of the Voronoi diagram, the origin can be a vertex, which has a fixed location, or a breakpoint, which is a breakpoint between two moving arcs. In the latter case, we need to calculate the position based on the *y*-coordinate of the sweep line.

Parameters

- **y** (*Decimal*) – The y-coordinate of the sweep line.
- **max_y** – Bounding box top for clipping infinitely highly positioned breakpoints.

Returns **origin**

Return type *Coordinate*

set_next(*next*)
Update the *next*-property for this edge and set the *prev*-property on the *next*-edge to the current edge.

Parameters **next** (HalfEdge) – The next edge

property target
The twin's origin.

Returns **vertex**

Return type *Vertex*

property twin
Get the other half-edge from the same edge

Returns **twin**

Return type *HalfEdge*

1.3.5 Point

class foronoi.graph.**Point**(*x=None*, *y=None*, *name=None*, *first_edge=None*)
A cell point a.k.a. a site. Extends the *Coordinate* class.

Examples

Site operations

```
>>> size: float = site.area()                  # The area of the cell
>>> borders: List[HalfEdge] = site.borders()   # Borders around this cell point
>>> vertices: List[Vertex] = site.vertices()   # Vertices around this cell point
>>> site_x: float = site.x                     # X-coordinate of the site
>>> site_xy: [float, float] = site.xy          # (x, y)-coordinates of the site
>>> first_edge: HalfEdge = site.first_edge     # First edge of the site's border
```

Parameters

- **x** (*Decimal*) – The x-coordinate of the point
- **y** (*Decimal*) – They y-coordinate of the point
- **metadata** (*dict*) – Optional metadata stored in a dictionary
- **name** (*str*) – A name to easily identify this point
- **first_edge** (HalfEdge) – Pointer to the first edge

name
A name to easily identify this point

Type str

first_edge
Pointer to the first edge

Type *HalfEdge*

area(*digits=None*)
Calculate the cell size of the cell that this point is the cell point of. Under the hood, the shoelace algorithm is used.

Parameters **digits** (*int*) – The number of digits to round to

Returns **area** – The area of the cell

Return type float

borders()
Get a list of all the borders that surround this cell point.

Returns **edges** – The list of borders, or None if not all borders are present (when the voronoi diagram is under construction)

Return type list(*HalfEdge*) or None

vertices()
Get a list of all the vertices that surround this cell point.

Returns **vertices** – The list of vertices, or None if not all borders are present (when the voronoi diagram is under construction)

Return type list(*Vertex*) or None

1.3.6 Polygon

class foronoi.graph.**Polygon**(*tuples*)
A bounding polygon that will clip the edges and fit around the Voronoi diagram.

Parameters **tuples** (*list[(float, float)]*) – x,y-coordinates of the polygon's vertices

finish_edges(*edges*, ***kwargs*)
Clip the edges to the bounding box/polygon, and remove edges and vertices that are fully outside. Inserts vertices at the clipped edges' endings.

Parameters **edges** (*list(HalfEdge)*) – A list of edges in the Voronoi diagram. Every edge should be presented only by one half edge.

Returns **clipped_edges** – A list of clipped edges

Return type list(*HalfEdge*)

finish_polygon(*edges*, *existing_vertices*, *points*)
Creates half-edges on the bounding polygon that link with Voronoi diagram's half-edges and existing vertices.

Parameters

- **edges** (*list (HalfEdge)*) – The list of clipped edges from the Voronoi diagram
- **existing_vertices** (*set (Vertex)*) – The list of vertices that already exists in the clipped Voronoi diagram, and vertices
- **points** (*set (Point)*) – The list of cell points

Returns

- **edges** (*list(HalfEdge)*) – The list of all edges including the bounding polygon's edges
- **vertices** (*list(Vertex)*) – The list of all vertices including the

inside(*point*)
Tests whether a point is inside a polygon. Based on the Javascript implementation from https://github.com/substack/point-in-polygon

Parameters **point** (Point) – The point for which to check if it it is inside the polygon

Returns **inside** – Whether the point is inside or not

Return type bool

1.3.7 Vertex

class foronoi.graph.**Vertex**(*x*, *y*, *connected_edges=None*)
A vertex is a fixed cross point between borders. Extends the *Coordinate* class.

Examples

Vertex operations

```
>>> connected_edges: List[HalfEdge] = vertex.connected_edges  # All connected␣
↪edges
>>> vertex_x: float = vertex.x                                # x-coordinate
>>> vertex_xy: [float, float] = vertex.xy                     # (x, y)-coordinates
```

Parameters

- **x** (*Decimal*) – x-coordinate
- **y** (*Decimal*) – y-coordinate
- **connected_edges** (list(*HalfEdge*)) – List of edges connected to this vertex.

connected_edges
List of edges connected to this vertex.

Type list(*HalfEdge*)

1.3.8 Visualizer

class foronoi.visualization.visualizer.**Visualizer**(*voronoi*, *canvas_offset=1*, *figsize=(8, 8)*)

A visualizer for your voronoi diagram.

Examples

Quickly plot individual components of the graph.

```
>>> vis = Visualizer(voronoi, canvas_offset=1)
>>> vis.plot_sites(show_labels=True)
>>> vis.plot_edges(show_labels=False)
>>> vis.plot_vertices()
>>> vis.plot_border_to_site()
>>> vis.show()
```

Chaining commands

```
>>> Visualizer(voronoi, 1).plot_sites().plot_edges().plot_vertices().show()
```

Plot all components that are useful to visualize during construction of the diagram

```
>>> from foronoi.visualization import Presets
>>> Visualizer(voronoi, 1).plot_all(**Presets.construction)
```

Plot all components that are useful to visualize when the diagram is constructed

```
>>> Visualizer(voronoi, 1).plot_all()
```

Parameters

- **voronoi** (*Voronoi*) – The voronoi object
- **canvas_offset** (*float*) – The space around the bounding object
- **figsize** (*float, float*) – Width, height in inches

get_canvas()

Retrieve the figure.

Returns Figure

Return type matplotlib.figure.Figure

plot_all(*polygon=False*, *edges=True*, *vertices=True*, *sites=True*, *outgoing_edges=False*, *border_to_site=False*, *scale=1*, *edge_labels=False*, *site_labels=False*, *triangles=False*, *arcs=False*, *sweep_line=False*, *events=False*, *arc_labels=False*, *beach_line=False*)

Convenience method that calls other methods to display parts of the diagram.

Parameters

- **polygon** (*bool*) – Display the polygon outline. *Only useful during construction.*
- **edges** (*bool*) – Display the borders of the cells.
- **vertices** (*bool*) – Display the intersections of the edges.
- **sites** (*bool*) – Display the cell points (a.k.a. sites)

- **outgoing_edges** (*bool*) – Show arrows of length *scale* in the direction of the outgoing edges for each vertex.
- **border_to_site** (*bool*) – Indicate with dashed line to which site a border belongs. The site's first edge is colored green.
- **scale** (*float*) – Used to set the length of the *outgoing_edges*.
- **edge_labels** (*bool*) – Display edge labels of format "*A/B*", where the edge is *A*'s border and the edge's twin is *B*'s border.
- **site_labels** (*bool*) – Display the labels of the cell points, of format "*P#*", where *#* is the `n`th point from top to bottom.
- **triangles** (*bool*) – Display the triangle of the 3 points responsible for causing a circle event. *Only useful during construction.*
- **arcs** (*bool*) – Display each arc for each point. Only used if *beach_line* is also *True*. *Only useful during construction.*
- **sweep_line** (*bool*) – Display the sweep line. *Only useful during construction.*
- **events** (*bool*) – Display circles for circle events. *Only useful during construction.*
- **arc_labels** (*bool*) – Display labels on the arcs. *Only useful during construction.*
- **beach_line** (*bool*) – Display the beach line. *Only useful during construction.*

Returns self

Return type *Visualizer*

plot_arcs(*arcs=None*, *sweep_line=None*, *plot_arcs=False*, *show_labels=True*)

Display each arc for each point. Only used if *beach_line* is also *True*. *Only useful during construction.*

Parameters

- **arcs** (list(*Arc*)) –
- **sweep_line** (*Decimal*) – The y-coordinate of the sweep line, used to calculate the positions of the arcs. By default, the *voronoi*'s sweep_line will be used.
- **plot_arcs** (*bool*) – Display each arc for each point
- **show_labels** (*bool*) – Display labels on the arcs.

Returns self

Return type *Visualizer*

plot_border_to_site(*edges=None*, *sweep_line=None*)

Indicate with dashed line to which site a border belongs. The site's first edge is colored green.

Parameters

- **edges** (list(*foronoi.graph.HalfEdge*), optional) – The edges to display. By default, the *voronoi*'s edges will be used.
- **sweep_line** (*Decimal*) – The y-coordinate of the sweep line, used to calculate the positions of unfinished edges. By default, the *voronoi*'s sweep_line will be used.

Returns self

Return type *Visualizer*

plot_edges(*edges=None*, *sweep_line=None*, *show_labels=True*, *color='#636e72'*, ***kwargs*)

Display the borders of the cells.

Parameters

- **edges** (list(*foronoi.graph.HalfEdge*), optional) – The edges to display. By default, the *voronoi*'s edges will be used.
- **sweep_line** (*Decimal*) – The y-coordinate of the sweep line, used to calculate the positions of unfinished edges. By default, the *voronoi*'s sweep_line will be used.
- **show_labels** (*bool*) – Display edge labels of format "*A/B*", where the edge is *A*'s border and the edge's twin is *B*'s border.
- **color** (*str*) – Color of the sites in hex format (e.g. "#636e72").

Returns self

Return type *Visualizer*

plot_event(*event=None*, *triangles=False*)
Display circles for circle events. *Only useful during construction.*

Parameters

- **event** (*Event*) – A circle event. Other events will be ignored.
- **triangles** (*bool*) – Display the triangle of the 3 points responsible for causing a circle event.

Returns self

Return type *Visualizer*

plot_outgoing_edges(*vertices=None*, *scale=0.5*, ***kwargs*)
Show arrows of length *scale* in the direction of the outgoing edges for each vertex.

Parameters

- **vertices** (list(*foronoi.graph.Vertex*), optional) – The vertices for which to display the outgoing edges. By default, the *voronoi*'s vertices will be used.
- **scale** (*float*) – Used to set the length of the *outgoing_edges*.
- **kwargs** – Optional arguments that are passed to arrowprops

Returns self

Return type *Visualizer*

plot_polygon()
Display the polygon outline. *Only useful during construction.*

Returns self

Return type *Visualizer*

plot_sites(*points=None*, *show_labels=True*, *color='#bdc3c7'*, *zorder=10*)
Display the cell points (a.k.a. sites).

Parameters

- **points** (list(*foronoi.graph.Point*), optional) – The vertices to display. By default, the *voronoi*'s vertices will be used.
- **show_labels** (*bool*) – Display the labels of the cell points, of format "*P#*", where # is the `n`th point from top to bottom.
- **color** (*str*) – Color of the sites in hex format (e.g. "#bdc3c7").
- **zorder** (*int*) – Higher order will be shown on top of a lower layer.

Returns self

Return type *Visualizer*

plot_sweep_line(*sweep_line=None*)
Plot the sweep line.

Parameters **sweep_line** (*Decimal*) – The y-coordinate of the sweep line. By default, the *voronoi*'s sweep_line will be used.

Returns self

Return type *Visualizer*

plot_vertices(*vertices=None*, ***kwargs*)
Display the intersections of the edges.

Parameters **vertices** (list(*foronoi.graph.Vertex*), optional) – The vertices to display. By default, the *voronoi*'s vertices will be used.

Returns self

Return type *Visualizer*

show(*block=True*, ***kwargs*)
Display all open figures.

Parameters **block** (*bool, optional*) – If *True* block and run the GUI main loop until all windows are closed.

If *False* ensure that all windows are displayed and return immediately. In this case, you are responsible for ensuring that the event loop is running to have responsive figures.

Returns self

Return type *Visualizer*

1.3.9 Voronoi

foronoi.**Voronoi**
alias of *foronoi.algorithm.Algorithm*

1.4 Private classes

1.4.1 Arc

class foronoi.nodes.**Arc**(*origin: foronoi.graph.coordinate.Coordinate*, *circle_event=None*)
Each leaf of beach line, representing an arc , stores one pointer to a node in the event queue, namely, the node that represents the circle event in which will disappear. This pointer is None if no circle event exists where will disappear, or this circle event has not been detected yet.

Parameters

- **origin** (Point) – The point that caused the arc
- **circle_event** (CircleEvent) – The pointer to the circle event in which the arc will disappear

origin
The point that caused the arc

Type *Point*

circle_event
The pointer to the circle event in which the arc will disappear

Type *CircleEvent*

get_plot(*x*, *sweep_line*)
Computes all *y*-coordinates for given *x*-coordinates and the sweep line's *y*-coordinate.

Parameters

- **x** (*np.array*) – The input x-coordinates
- **sweep_line** (*Decimal, float*) – The y-coordinate of the sweep line

Returns **y** – A list of y-values

Return type number, array-like

1.4.2 Breakpoint

class foronoi.nodes.**Breakpoint**(*breakpoint: tuple*, *edge=None*)
A breakpoint between two arcs.

The breakpoint is stored by an ordered tuple of sites (p_i, p_j) where p_i defines the parabola left of the breakpoint and p_j defines the parabola to the right. Furthermore, the internal node v has a pointer to the half edge in the doubly connected edge list of the Voronoi diagram. More precisely, v has a pointer to one of the half-edges of the edge being traced out by the breakpoint represented by v.

Parameters **breakpoint** ((Point, Point)) – A point where two arcs intersect, represented as a tuple of the two site points that the arcs refer to

does_intersect()
A guard that handles the edge-case where two arcs were initialized at the same time due to their sites having the same y-coordinate. This guard makes sure that the left arc intersects once with the right arc and not the other way around.

Returns **intersects** – Returns false when p_i and p_j have the same y-coordinate and p_j is situated left of *p_i*.

Return type bool

get_intersection(*l*, *max_y=None*)
Calculate the coordinates of the intersection Modified from https://www.cs.hmc.edu/~mbrubeck/voronoi.html

Parameters

- **l** (*float*) – The y-coordinate of the sweep line
- **max_y** (*float*) – The top of the bounding box/polygon for clipping infinite breakpoints

Returns **coordinate** – The current coordinates of the breakpoint

Return type *Coordinate*

1.4.3 CircleEvent

class foronoi.events.circle_event.**CircleEvent**(*center: foronoi.graph.coordinate.Coordinate*, *radius: decimal.Decimal*, *arc_node: foronoi.nodes.leaf_node.LeafNode*, *point_triple=None*, *arc_triple=None*)

A circle event.

Parameters

- **center** (*Coordinate*) – The center coordinate of the circle (where the new vertex will appear)
- **radius** (*Decimal*) – The radius of the circle
- **arc_node** (*LeafNode*) – Pointer to the node in the beach line tree that holds the arc that will disappear
- **point_triple** ((*Point, Point, Point*)) – The triple of points that caused the event
- **arc_triple** ((*Arc, Arc, Arc*)) – The triple of arcs related to the points

static create_circle(*a*, *b*, *c*)

Create a circle from three coordinates.

Parameters

- **a** (*Coordinate*) –
- **b** (*Coordinate*) –
- **c** (*Coordinate*) –

Returns

- **x** (*Decimal*) – The x-coordinate of the center of the circle
- **y** (*Decimal*) – The y-coordinate of the center of the circle
- **radius** (*Decimal*) – The radius of the circle

static create_circle_event(*left_node: foronoi.nodes.leaf_node.LeafNode*, *middle_node: foronoi.nodes.leaf_node.LeafNode*, *right_node: foronoi.nodes.leaf_node.LeafNode*, *sweep_line*) → *foronoi.events.circle_event.CircleEvent*

Checks if the breakpoints converge, and inserts circle event if required.

Parameters

- **left_node** (*LeafNode*) – The node that represents the arc on the left
- **middle_node** (*LeafNode*) – The node that represents the arc in the middle
- **right_node** (*LeafNode*) – The node that represents the arc on the right
- **sweep_line** (*Decimal*) – The y-coordinate of the sweep line

Returns **circleEvent** – The circle event or None if no circle event needs to be inserted

Return type *CircleEvent* or None

remove()

Mark this circle event as a false alarm.

Returns **self**

Return type *CircleEvent*

property xd
The x-coordinate (in Decimal format) of the center of the circle, which functions as the secondary priority of this event.

Returns x

Return type Decimal

property yd
The y-coordinate (in Decimal format) of the bottom of the circle, which functions as the primary priority of this event.

Returns y

Return type Decimal

1.4.4 SiteEvent

class foronoi.events.site_event.**SiteEvent**(*point: foronoi.graph.point.Point*)
A site event.

Parameters **point** (Point) – The point that causes the site event.

property xd
The x-coordinate (in Decimal format) of the point, which functions as the secondary priority of this event.

Returns x

Return type Decimal

property yd
The y-coordinate (in Decimal format) of the point, which functions as the primary priority of this event.

Returns y

Return type Decimal

Note: More to be added soon!

1.5 Observers

1.5.1 DebugObserver

class foronoi.**DebugObserver**(*callback=None*)
Listens to debug messages.

Parameters **callback** (*function*) – By default, the DebugObserver prints the debug message. When a callback function is given, it will pass the debug message as string to the callback function.

update(*subject:* foronoi.observers.subject.Subject, *message:* foronoi.observers.message.Message, ***kwargs*)
Send the updated state of the algorithm to the VoronoiObserver.

Parameters

- **subject** (Algorithm) – The algorithm to observe
- **message** (Message) – The message type

- **kwargs** (*dict*) – Keyword arguments that include a payload-parameter of type str

1.5.2 Message

class foronoi.observers.message.**Message**(*value*)
Enum class for message types.

STEP_FINISHED
Indicates that the algorithm processed one event

SWEEP_FINISHED
Indicates that the sweep line algorithm is finished

VORONOI_FINISHED
Indicates that the voronoi diagram is clipped and cleaned

DEBUG
Indicates that this message is a debugging-message

1.5.3 Observer

class foronoi.observers.observer.**Observer**
The Observer interface declares the update method, used by subjects.

abstract update(*subject*, *message:* foronoi.observers.message.Message, ***kwargs*) → None

Parameters

- **subject** (Subject) – The sender of the update
- **message** (Message) – The message type
- **kwargs** (*dict*) – Any additional keyword arguments

1.5.4 Subject

class foronoi.observers.subject.**Subject**
An observable subject that you can attach observers to.

attach_observer(*observer:* foronoi.observers.observer.Observer)
Attach an observer to the subject.

Parameters **observer** (Observer) – An observer to attach to this subject

detach_observer(*observer:* foronoi.observers.observer.Observer)
Detach an observer from the subject.

Parameters **observer** (Observer) – An observer to remove from this subject

get_observers()
Getter for observers

inherit_observers_from(*parent*)
Make this subject inherit observers from a parent. When the child sends an update to the observers, the parent will be passed as the sender.

Parameters **parent** (Subject) – The parent to inherit observers from

notify_observers(*message*, ***kwargs*)
Notify all observers about an event.

Parameters **message** (Message) – The message type

1.5.5 TreeObserver

class foronoi.**TreeObserver**(*visualize_steps=True*, *visualize_result=True*, *text_based=False*, *callback=None*)

Observers the state of the status tree (*foronoi.algorithm.Algorithm.status_tree*) and visualizes the result using GraphViz.

Parameters

- **visualize_steps** (*bool*) – Visualize all individual steps
- **visualize_result** (*bool*) – Visualize the final result
- **text_based** (*bool*) – Visualize the tree using plain text instead of GraphViz
- **callback** (*function*) – By default, the TreeObserver renders and shows the result in a window, or prints the result when *text_based* is true. When a callback function is given, either the GraphViz diagram or the text-string is passed to the callback.

Examples

```
>>> from foronoi import Voronoi, TreeObserver, Polygon
>>> points = [
...     (2.5, 2.5), (4, 7.5), (7.5, 2.5), (6, 7.5), (4, 4), (3, 3), (6, 3)
... ]
>>> poly = Polygon(
...     [(2.5, 10), (5, 10), (10, 5), (10, 2.5), (5, 0), (2.5, 0), (0, 2.5), (0,␣
↪5)]
... )
>>> v = Voronoi(poly)
>>>
>>> # Define callback
>>> def callback(observer, dot):
...     dot.render(f"output/tree/{observer.n_messages:02d}")
>>>
>>> # Attach observer
>>> v.attach_observer(TreeObserver(callback=callback))
>>>
>>> # Start diagram creation
>>> v.create_diagram(points)
```

update(*subject: foronoi.algorithm.Algorithm*, *message: foronoi.observers.message.Message*, ***kwargs*)

Send the updated state of the algorithm to the TreeObserver.

Parameters

- **subject** (Algorithm) – The algorithm to observe
- **message** (Message) – The message type

1.5.6 VoronoiObserver

class foronoi.**VoronoiObserver**(*visualize_steps=True*, *visualize_before_clipping=False*, *visualize_result=True*, *callback=None*, *figsize=(8, 8)*, *canvas_offset=1*, *settings=None*)

Observers the state of the algorithm (*foronoi.algorithm.Algorithm*) and visualizes the result using the Visualizer (*foronoi.visualization.visualizer.Visualizer*).

Parameters

- **visualize_steps** (*bool*) – Visualize all individual steps
- **visualize_before_clipping** (*bool*) – Visualize the result before the edges are clipped
- **visualize_result** (*bool*) – Visualize the final result
- **callback** (*function*) – By default, the VoronoiObserver shows or prints the result when *text_based* is true. When a callback function is given, either the GraphViz diagram or the text-string is passed to the callback.
- **figsize** (*(float, float)*) – Window size in inches
- **canvas_offset** (*float*) – The space around the bounding object
- **settings** (*dict*) – Visualizer settings to override the default presets used by the VoronoiObserver

Examples

```
>>> from foronoi import Voronoi, VoronoiObserver, Polygon
>>> points = [
...     (2.5, 2.5), (4, 7.5), (7.5, 2.5), (6, 7.5), (4, 4), (3, 3), (6, 3)
... ]
>>> poly = Polygon(
...     [(2.5, 10), (5, 10), (10, 5), (10, 2.5), (5, 0), (2.5, 0), (0, 2.5), (0,␣
↪5)]
... )
>>> v = Voronoi(poly)
>>>
>>> # Define callback and settings
>>> def callback(observer, figure):
...     figure.savefig(f"output/voronoi/{observer.n_messages:02d}.png")
>>> settings=dict(arc_labels=True, site_labels=True)
>>>
>>> # Attach observer
>>> v.attach_observer(VoronoiObserver(callback=callback, settings=settings))
>>>
>>> # Start diagram creation
>>> v.create_diagram(points)
```

update(*subject:* foronoi.algorithm.Algorithm, *message:* foronoi.observers.message.Message, ***kwargs*)

Send the updated state of the algorithm to the VoronoiObserver.

Parameters

- **subject** (Algorithm) – The algorithm to observe
- **message** (Message) – The message type

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

N

O

P

R

S

T

U

V

X

Y