Overview

This software is written by Python (3.6).

Introduction

This project is to compute the Voronoi diagram (https://en.wikipedia.org/wiki/Voronoi_diagram) of given set of points. The Fortune's algorithm, which can compute the Voronoi diagram in O(nlog n) time,` is used. For convenience, the algorithm is listed as follows from the book [1].

Algorithm VORONOIDIAGRAM(P)
Input. A set $P := \{p_1, \ldots, p_n\}$ of point sites in the plane.
Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

1. Initialize the event queue $\mathcal{Q}$ with all site events, initialize an empty status structure $\mathcal{T}$ and an empty doubly-connected edge list $\mathcal{D}$.
2. **while** $\mathcal{Q}$ is not empty
3. **do** Remove the event with largest y-coordinate from $\mathcal{Q}$.
4. **if** the event is a site event, occurring at site p_i
5. **then** HANDLESITEEVENT(p_i)
6. **else** HANDLECIRCLEEVENT(γ), where γ is the leaf of $\mathcal{T}$ representing the arc that will disappear
7. The internal nodes still present in $\mathcal{T}$ correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding box by updating the doubly-connected edge list appropriately.
8. Traverse the half-edges of the doubly-connected edge list to add the cell records and the pointers to and from them.

HANDLESITEEVENT(p_i)

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

HANDLECIRCLEEVENT(γ)

1. Delete the leaf γ that represents the disappearing arc α from $\mathcal{T}$. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary. Delete all circle events involving α from $\mathcal{Q}$; these can be found using the pointers from the predecessor and the successor of γ in $\mathcal{T}$. (The circle event where α is the middle arc is currently being handled, and has already been deleted from $\mathcal{Q}$.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list $\mathcal{D}$ 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 α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into $\mathcal{Q}$. and set pointers between the new circle event in $\mathcal{Q}$ and the corresponding leaf of $\mathcal{T}$. Do the same for the triple where the former right neighbor is the middle arc.

Software features

1. Computation of the Voronoi diagram.

2. Compute the Delaunay triangulation.

3. Animation of the computation process of Voronoi diagram.

Details of the codes

Since the software is based on visual studio, some files are redundant. The essence of the software lies in the following file:

1. animatevoronoi.py: main entrance to show the animation of the computation process.

2. compute_voronoi.py: main entrance to compute the Voronoi diagram

3. handle_event.py: corresponding to HANDLESITEEVENT and HANDLECIRCLEEVENT

4. queue_swx.py: data structure---queue.

5. quicksort_swx.py: quicksort of the nodes in the tree

6. status_swx.py: maintenance of the tree.

7: DCEL.py: data structure DCEL (Doubly connected edge list).

Remarks

1. The video shows how the software works

2. The file sites.txt contains the coordinates of points of the sites, of which the Voroni diagram is to be computed.

3. It seems that the project is very easy at first glance since someone has designed an algorithm. However, it is extremely complicated if you start from scratch without any other external libraries. In particular, to achieve the running time of O(nlog n) every part has to be optimized like the searching nodes on the tree and the selection of tree type. This can also be seen from the length of codes: more than 2000 lines of codes are needed.

Reference

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. Computational Geometry: Algorithms and Applications (3rd ed. ed.). TELOS, Santa Clara, CA, USA.