MCS 481 / David Dumas / Spring 2012

Project 3: Voronoi and Delaunay

Due Friday, April 4

0. Overview

For this project you have two options:

- Implement Voronoi decomposition / Delaunay triangulation
- Go "off the rails" and propose your own experimental or coding project.

The options are described in detail below.

1. Coding option - Voronoi decomposition / Delaunay triangulation

Problem specification. Write a program that computes the Voronoi decomposition of the plane determined by three or more point sites that do not all lie on a single line, and which prints a list of Voronoi vertices and edges to standard output (according to the precise format specified below).

Since the Voronoi diagram can be computed easily from the Delaunay graph, one approach to completing the project would be to implement Delaunay triangulation and then process the result in order to determine the Voronoi vertices and edges.

Due to the relative complexity of efficient algorithms for Voronoi decomposition or Delaunay triangulation, full credit will be given for a submission that is either:

- Efficient — time complexity $O(n \log n)$ with linear storage, OR
- Robust — correctly handles degenerate input.

Thus, for example, it would be sufficient to implement Fortune's plane sweep algorithm in a way that handles non-degenerate input. In this setting "degenerate input" refers to any of the following:

- Two event points have the same x or y coordinate
- Three sites are collinear
- Four sites are concircular

The evaluation of your project will include testing your program on small sets of sites for verification of the output, and on large sets to confirm that your program can handle them.

Input. Read a list of sites from `stdin`. Each line will contain the coordinates x y of one site, separated by whitespace characters (e.g. space, tab). The list of sites may be followed by any number of lines containing only whitespace characters, which must be ignored.

Output. Write a description of the Voronoi diagram to `stdout`; use the following general form for the output:

Vertex list
⟨blank line⟩
Edge list
⟨blank line or EOF⟩
Optional extra information

Project description v1.0 (March 11, 2014)

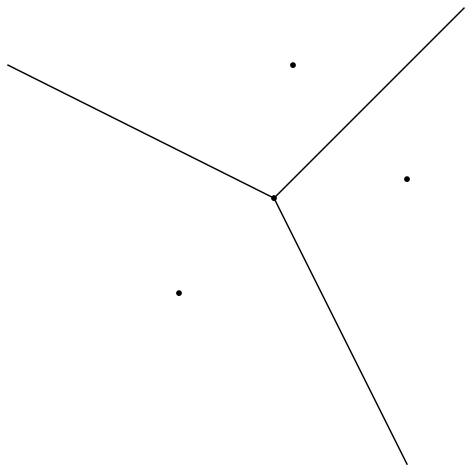

FIGURE 1. Voronoi diagram for the 3-site example.

Here the vertex list has one vertex per line, with coordinates separated by whitespace characters. The position of a vertex in this list determines its zero-based index (meaning 0=first vertex in list, 1=second, etc.), though this index is not part of the output.

Each line of the edge list contains a pair of zero-based indices of vertices that are connected by a line segment in the Voronoi diagram. It does not matter whether the list consists of oriented half-edges or unoriented edges, as long as each pair of vertices that are joined by a Voronoi edge appears in the list (in some order) and no other pairs appear.

The half-infinite edges require special handling. In order to represent them, we allow vertices *at infinity*. Unlike finite vertices, which are specified by a pair of coordinates (x, y), a vertex at infinity is specified by a single coordinate t in the interval $[0, 1)$. If an edge e has this vertex at infinity as an endpoint, then the edge is half-infinite and the clockwise angle from the positive x-axis to e is $2\pi t$. Thus, for example, the negative y-axis has endpoints $(0, 0)$ and 0.75.

Note that in the vertex list, the finite vertices can be distinguished from vertices at infinity because the latter will have only one coordinate.

If the Voronoi diagram has several parallel half-infinite edges, there will be several vertices at infinity with the same angle. These should be listed separately, rather than being combined into a single vertex record. That is, there should always be a bijection from the set of half-infinite edges to the set of vertices at infinity.

Face information is not required in the output of your program, but it might be helpful for debugging. If the edge list is followed by a blank line then your program is allowed to write arbitrary additional information before exiting.

Reference implementation. A C++ program that uses CGAL to compute the Voronoi diagram and print it in the format described above is included with this project description.

Examples. Note that the output of the program is not uniquely determined by the specifications above because a given Voronoi diagram could be described in many ways (e.g. re-ordering the vertex and edge lists). The following examples show conforming output for two Voronoi diagrams.

(1) Sample input for 3 sites:

```
0 1
1 0
-1 -1
```

Sample output for 3 sites:

```
-0.166667 -0.166667
0.125
0.426208
0.823792

0 1
0 2
0 3
```

There are three edges, each is half-infinite. The diagram is shown in Figure 1.

(2) Sample input for 5 sites:

```
0 1
-2 6
2 -6
4 4
-4 -4
```

Sample output for 5 sites:

```
-5.8333 1.5667
-0.2895 -2.8684
0.7308 4.1923
4.9118 -1.3824
0.1988
0.4686
0.9686
0.6988

0 1
0 2
0 5
1 3
1 7
2 3
2 4
3 6
```

There are eight edges, four are half-infinite. There is one bounded cell corresponding to the site at the origin. Its vertices have indices $0, 1, 3, 2$. The diagram is shown in Figure 2.

Languages and libraries. The same programming language and library policies apply as in previous projects, with one important exception:

> For this project you may use an external library that implements a doubly-connected edge list (DCEL). However, your code should only communicate with the library through the use of basic $O(1)$ operations on the DCEL as described in our textbook. Since I need to be able to compile your code, please contact me for approval of the specific library and version you intend to use.

Also remember to contact me for language approval, if applicable.

What to submit. Write a report on your implementation process. Start by briefly describing the problem and the algorithm. Then focus on the design decisions that were involved in your implementation.

Create test cases for your code that demonstrate its correctness for various types of input. Display these test configurations and their Voronoi diagrams in the report.

Submit the report, source code, and test cases by email (`ddumas@math.uic.edu`), following the standards for coding option and source code submissions from the description of Project 1.

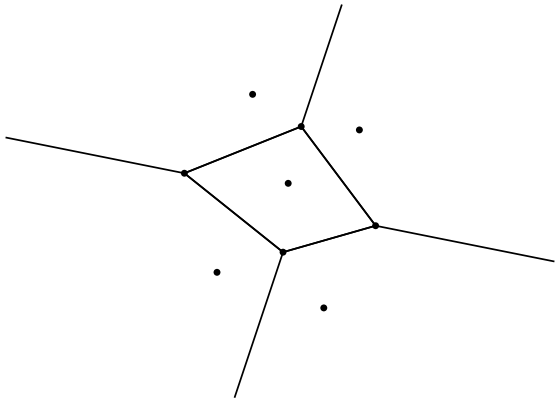

FIGURE 2. Voronoi diagram for the 5-site example.

2. Off the rails

Design your own coding, experimental, or theoretical project and send me a project proposal (by email to `ddumas@math.uic.edu`). Your proposal should be a few paragraphs in length, roughly similar to the kind of description given above for the Voronoi diagram coding option. Your description **must** indicate:

- What final products you will submit for evaluation
- What references you will use (to show you have some idea where to start)
- What algorithms you will implement (if coding) or study (if doing experiments or theory)

Furthermore, if choosing an experimental option, indicate what implementation(s) you will study and what questions you intend to answer about them.

The proposal is due by **Monday, March 17**. I will read your proposal and either approve it or suggest some changes. I will also indicate how your project would be evaluated. Once you receive approval and are satisfied with the proposed evaluation standards, you can begin working.

Sample project ideas.

- Voronoi diagrams can be defined for other ways of measuring distance in the plane (e.g. the "taxicab" distance $d(p, q) = |p_x - q_x| + |p_y - q_y|$). Find out what is known about these generalized Voronoi diagrams (theoretically, algorithmically) and write an expository report.
- Compare CGAL's various point location strategies using timing experiments and analyze the results.
- Write a program to compute the medial axis transform of a simple polygon. (This is a special case of the Voronoi diagram that has applications in shape recognition.)