Point Location in Voronoi Diagrams

Anders Høst Kjærgaard and Hildur Uffe Flemberg
{ahkj, hufl}@itu.dk

IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

Abstract. Voronoi diagrams are used in a variety of contexts in mathematics, biology and computing. We are interested in analysing the composition of the voronoi diagram to say something about the underlying data that the diagram represents. In this paper we descibe a solution that generates a voronoi diagram from an arbitrary set of data points and returns a data structure for performing point location i O(log n) time in the generated diagram. The solution makes use of an existing open-source implementation of Fortunes algorithm for generating voronoi diagrams, which is merged with our own implementation of a trapezoidal map for performing point location. Querying the trapezoidal map data structure will return the site of the voronoi cell in which the query point lies. As a concrete example of the usefulness of this, we discuss how the algorithm could be used for answering point location queries on a data set of fast-food restaurants. The idea is that one could seek to answer questions like which restaurants would cover or attract the most customers, or, in general, which restaturant is closest to a given point of interest. Secondly, we provide an analysis of the expected number of customers that will end in the biggest voronoi cell when distributed randomly across the fast-food restaurant diagram. We find that such information querying on the target data set, could be of potential interest when dealing with location planning of new restaurants, or analysing customer behavior.

Keywords: algorithms, computational geometry, point location, software development

1 Introduction

Voronoi diagrams have been know for... In YYYY Fortune invented the algorithm for ... They are interesting because this and that. Get examples from `http://voronoi.com/wiki/index.php?title=Voronoi_Applications` We use restaurants, defined as sites.

Various interesting questions could be asked about the diagram, that would also pose of interest in algorithm design. In which cell does a given point reside? Which cell is the largest? The questions are not jsut interest for the algorithmic challange, but also due to how we can interpret the answer to such questions in relation to the sites that composit the diagram. E.g. we could assume that if a given

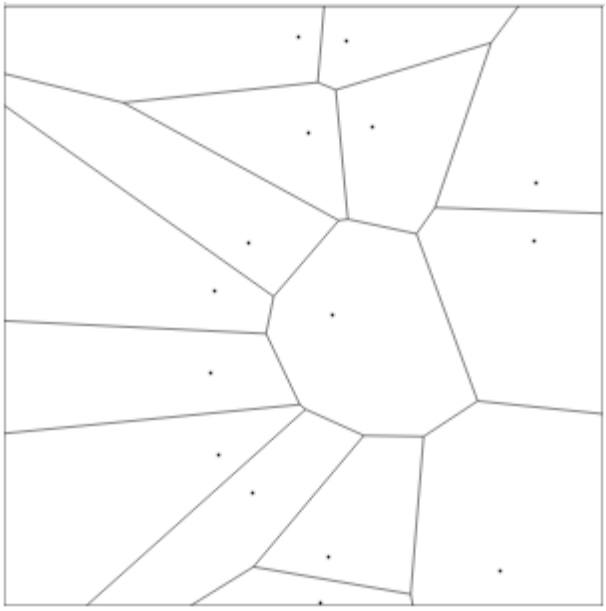

Fig. 1. A Voronoi Diagram with 15 sites

2 Background

Basic description of voronoi diagrams and the algortihm. Refer to the book, but explain running time of Fortune vs slap approach and single shot. Describe circle events, how they end up and get pulled from the event queue and why they are important - segments are generated and we want to add them to the map when taht happens.

Trapezoidal map. Maybe a bit more fine grained explanation. Refer to book for full detail, a few visual examples.

3 Problem Statement

As mentioned in section, there are multiple useful applications of point location in voronoi diagrams. In the remainder of this report we describe a solution that facilitates this kind of point location using techniques from two algorithms that are known in advance – one that generates voronoi diagrams and one that performs point location in any planar subdivision.

3.1 Desirable Features

- The complexity should be unchanged, i.e. not bigger than any of the existing algorithms
- The solution must be generic, i.e. not targeted a specific business area or domain

3.2 Assumptions

– We require that no two sites of the input share the same x-coordinate as this is a simplifying assumption made by the original point location algorithm. This assumption means that every trapezoid can be uniquely represented in terms of their left point, right point, bottom segment and top segment. Furthermore, every trapezoid can have at most four neighbours. It is clear that this assumption is unreasonable when working with real-world data. How to deal with input where such "degenerate" cases occur is described in [1]

4 Solution

The solution is two fold and described in the following.

4.1 Point location in Voronoi

While there are existing algorithms for the two problems with good and optimal running times, an analysis of the two algoritms makes it clear that, for the scope of this project, it would not make much sense to look for an improvement or more elegant way of acomplishing what the two algorithm do. By realizing that the trapezoidal map simply reuires a random edge at the time until all edges have been traversed, makes it somewhat trivial to see that the running output of the voronoi algorithm can be used as input for the trapezoidal map algorithm. Seen in another way, while merging the two algorithm might be a learningful experience, it would in the general case make sense to just keep the two algorithms seperated and rund them in sequense, as the complexity and result is exactly the same. On the down side one could argue that merging them is just a minus as it simply makes the implementation harder to understand. In fact, by merging them we do save some nlognn computations, but unless we come up with some extreme case, we see this as a premature optimization. However, we find the task of combining the two of sufficient academic interest to have carried it through and provided an implementation. describe voronoi and how one needs to modify to the creation of segments upon circle events, maybe discuss why this is actually a final segment, refer to DCEL data structure and half edges not knowing their other ends. Describe the open source solution that we use, problems with it, and bridge it to our trapezoidal map implementation. Roud off with nice features of our trapezoidal map implementation. And show the simplicity in getting the tree DS from it for querying.

4.2 Expected Customers

5 Evaluation

Test with a few screen shots showing that algo actually found the right cell.

6 Threats to Validity

Buggy software til generering af Voronoi.

7 Plan For Future Work

7.1 Future Work

Create a clean and generic voronoi implementation in .NET 4.x Find best point for planar location. Not trivial, would probably be Hawaii, but that would be winning a lot of oceanic area.

7.2 Something else?

8 Conclusion

We have shown the design of two algorithms in the field of computational geometry. We found that altering the the Fortune algorithm to return a fast search data structure of a trapezoidal map for point location was indeed feasible, and showed that by querying random points at the datastructure we could reason about which cell is the biggest. The algorithms presented are implemented in a generic fashion and it should be easy to see the solution used on top of another any other data set.

Acknowledgements We would like to thank Rasmus Pagh for his lectures and his on the spot guidance that has been essential for moving this project in the right direction.

References

[1] O. Schwarzkopf, M. Overmars, M. van Kreveld, and M. de Berg. *Computational Geometry, Algorithms and Applications*. Springer, 1997.