the big paper, if you will

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Isabella F. Jorissen

May 2016

Approved for the Division
(Mathematics)

James Fix

Acknowledgements

I want to thank a few (lots, really) people.

Preface

[I'm going to tell you a story about a math problem. There will be some words (don't worry about their meaning too much), some concepts (the words will help some), and a few pretty diagrams (to pull it all together). This is a story about a lot of things, depending on your perspective. In a way, it's a story about figuring out where you ought to go based on where you are. At least, that one of the ways I like to think about it. I've thrown in a little High Performance Computing for good measure, since - if you're anything like me - you'll want to know where your destination is as quickly as possible.

This math problem has a lot of names, and has been discovered, re-discovered, named, re-named, used, and re-used in more disciplines than I'm willing to count or list. I'd argue that the "conceptual" "mathematical" structure itself "belongs" to the realm of Computational Geometry, where it's referred to as the Voronoi Diagram. It has many aliases: Dirichlet Tesselation, Thiessen Polygons, Plant Polygons, and Wigner-Seitz Cell, among others.

In the names of Tradition and Clarity, I've done my best to use exoteric language whenever possible, and to indulge in mathematical notation where appropriate. You're welcome to treat either as an endless stream of notation. My only advice to you, in the words of a Reed math professor I once had: "remain calm."]

Figure 1: check this ish out... now put it in illustrator to make it pretty n such

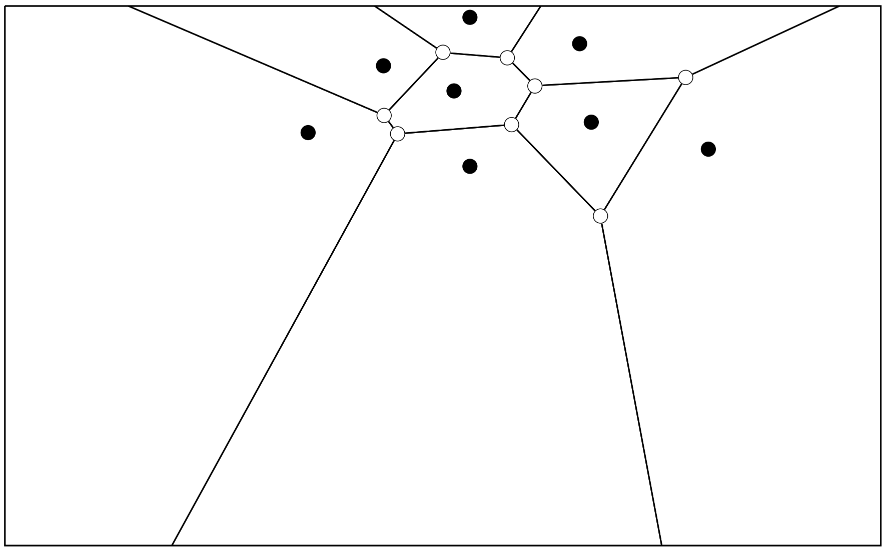

Table of Contents

List of Algorithms

Abstract

[more specifics][math specifics] [this is a math problem, this is a computational geometry problem, this is a spotlight on a common problem, this is related to a lot of other cool problems, this is a problem about space, this is an apology (explanatory), this is a survey of approaches to solving this problem as efficiently and elegantly as possible, this is building off of current implementations to examine the benefits of a) multicore / threaded and / or b) gpu accelerated implementations, this is about re-learning C, this is about wrangling threads and hating CUDA]

Introduction

[the classic post office problem ...][insert something better here] Suppose you are hoping to send a piece of mail via post office. In this day and age, it's uncommon to send a piece of mail, and you might ask yourself "Where is the nearest post office?"

But why is the post office *there*?

To answer that question, assume the perspective of a city planner whose task is to determine where the next post office should located. The city planner might work under the assumption that people go to the post office closest to where they live. Roughly, the planner determines the areas of service for each post office, which consists of dividing the city into regions whose occupants are closer to the post office in that region than any other office. The area of service, for a particular post office, corresponds directly with the Voronoi cell of that site (the post office). We call this cell, or a region.[1]

0.1 Formalizing the Post Office Problem

[The Voronoi diagram is more than a pretty picture; it is a versatile geometric structure!] [Disclaimer about nice point sets and diagrams with non degenerate cases?]The diagram is the subdivision induced under the Voronoi assignment model, which consists in assigning every point in the plane to the site nearest to it.

To construct a Voronoi diagram, we need a set of sites, P, and way to measure the distance between two sites, p and q, denoted dist(p,q). For the purposes of this thesis, we consider the Voronoi diagram in the plane, sites are defined as points, and the distance between any two points is the Euclidian distance.

Let $P := \{p_1, p_2, ..p_n\}$ a set of n distinct points – each one constituting a site – in the plane. The Voronoi diagram of P the subdivision of the plane into n regions where for each region V_{p_i} in the set of regions $V := \{V_{p_1}, V_{p_2}, ..V_{p_n}\}$, V_{p_i} is the region associated with site p_i. Further, the region V_{p_i} is the set of points in the plane such that each point in V_{p_i} is closer to p_i than it is to any other site in P. Formally, given a plane S, a region or cell in a Voronoi diagram is the set of points in the plane which satisfy:

$$V_{p_i} := \{\, q \in S \mid dist(q, p_i) < dist(q, p_j) \text{ for all } \ p_j \in P \ \text{ where } j \neq i\}.$$

[1]If the aforementioned example doesn't resonate with you, I'd encourage you to peruse appendix A, where I've included a plethora of other examples with the intent of motivating the problem from different perspectives, disciplines, and industries.

For our purposes, $S = \Re^2$, the Euclidian plane in two dimensions, and $dist(p, q)$ is the Euclidian measure of distance in $\Re^2$.

We can also think of the region V_{p_i} as the (possibly open) convex polygon which is the result of intersection of $n-1$ half-planes. Let's unpack that a little bit. Given two sites p_i and p_j, imagine the line, l connecting them. [graph] At the midpoint, m, of l, imagine a line b, purpendicular to l and passing through m. Note that b is the purpendicular bisector of l. Then b splits the plane into two half-planes. Define the half-plane containing p_i with $h(p_i, p_j)$; similarly, define the half-plane containing p_j with $h(p_i, p_j)$. To connect this notion back to our first description of a Voronoi region, note that all of the points in $h(p_i, p_j)$ will *not* be in p_j's region, since every point in $h(p_i, p_j)$ is closer to p_i than it is to p_j. Define set of half-planes containing p_i as $H_{p_i} := \{\, h(p_i, p_j) \text{ for all } p_j \in P \text{ where } j \neq i\}$. There are $n-1$ half-planes in H_{p_i}. Consider the intersection of any two of these half-planes, $h(p_i, p_j)$ and $h(p_i, p_k)$. The set of points lying in the intersection of $h(p_i, p_j)$ and $h(p_i, p_k)$ are all of the points which are closer to p_i than they are to p_j or p_k. If we performed this intersection over all $n-1$ half-planes, we would have described the Voronoi region of p_i exactly. So we can also describe the region as the intersection of a set of half-planes:

$$V_{p_i} = \bigcap_{h \in H_{p_i}}$$

Because the half-plane is a convex set[2] (or region), and the intersection of convex regions is also convex, we can say with confidence, nay, with certainty, that the Voronoi region is convex. In section 1.1 we describe an algorithm to construct the Voronoi diagram using an approach which relies heavily on intersection of half-planes and convex regions.

Every Voronoi region can be thought of as a (possibly open) convex polygon whose edges are the bounding lines of the convex regions and whose vertices are the points of intersection between the bounding lines. Note that some of the edges will be line segments, while others will extend out towards infinity. We will refer to the latter type of edge as a *half infinite edge*. Since the region was the result of $n-1$ half-plane intersections, we know that the region has, at most, $n-1$ edges. The Voronoi diagram of a set of sites P, $Vor(P)$, is the collection of all of these regions [at then end, though, in the simplest sense, is the handful of vertices and a bundle of edges resulting from the union of these regions.] [3]

We know that $Vor(P)$ will have n regions; and we can put an upper bound on the number of edges and vertices it has by recognizing that a Voronoi diagram in $\Re^2$ is nearly a planar graph. We make the Voronoi diagram of P into a planar graph by introducing a vertex "at infinity". Then, we assign the vertex at infinity to the half infinite edges (that is to say, all edges without two endpoints). [Diagram, before and

[2]A set is convex if the straight line connecting any two points in the set is also in the region.

[3]If I may draw your attention to the set of vertices and edges resulting from the union of regions, you might note that there are bound to be duplicates. For example, two adjacent regions will share a bounding edge and two vertices (the endpoints of the shared edge). The presence of duplicates in this set might indicate that constructing the Voronoi diagram by taking the union of the individually computed regions might not be the most space or time conscious approach.

after] *Euler's Formula* states that any planar graph with v vertices, e edges, and f faces (regions, for our purposes) the following is true: $v - e + f = 2$. Applying this to $Vor(P)$ with an infinite vertex, which has n regions, v vertices, and e edges we have:

$$(v + 1) - e + n = 2.$$

Since every edge has exactly two endpoints and every (and every vertex is of degree at least 3), we can say that $2e \geq 3(v+1)$. Together with Euler'f formula, this implies that for a $n \geq 3$, the number of vertices is at most $2n - 5$ and the number of edges is at most $3n - 6$.

0.2 The Voronoi Diagram and its Dual

[define triangulations, convex hull] [define dual, the delaunay diagram] [As long as the voronoi diagram doesn't have any degeneracies (all vertices are of degree 3) the delaunay graph will be a triangulation] Imagine that we had a Voronoi diagram on P, $Vor(P)$. Choose a site $p_i \in P$ and, using $Vor(P)$ as a reference, draw a line connecting p_i to the sites of the regions adjacent to it. [DIAGRAM] Do this for all the sites [DIAGRAM]. If we assume that $Vor(P)$ contains no degeneracies, then the resulting diagram is a Delaunay Trangulation. [This is true:] Define the convex hull of P to be the smallest convex region containing P [DIAGRAM of convex hull of the set of points]. It is the intersection of all of the convex sets containing P. The boundary of the Delaunay triangulation of P is the convex hull of P.

0.3 Properties of Voronoi and Delaunay Diagrams

they are just the cat's pajamas they are the bees knees.

0.4 Applications of Voronoi and Delaunay Diagrams

Generalizing voronoi diagrams (nearness, sites as objects other than points) [Problems with $\mathcal{O}(n)$ reduction to delaunay, voronoi]

0.5 Statement of algorithm goals (High Performance Computing)

[I'm not one for goal setting, but here goes...] In this thesis, we wish to bear witness to the effects of modifying the two aspects of two different approaches to constructing the voronoi diagram (or the delaunay triangulation). Further, we ask the following question, which will be further motivated by the section on divide and conquer algorithms: "given two valid delaunay triangulations (or voronoi diagrams), merge them". So we're trying to merge them in parallel so we can see the results and maybe say something about that. We're also thinking about the question "is there a way to *order* the randomized incremental approach in a way that allows us to have multiple workers attacking the problem while also minimizing the amount of conflict?" And we'll see more information on that problem in the section on randomized incremental and also the section on progress with that endeavor.]

Constructing the Voronoi Diagram

Algorithms to construct the Voronoi diagram fall into three rough categories: incremental approaches, divide and conquer approaches, and sweepline approaches. The worst case time compexity approaches for each of these is $\mathcal{O}(n^2)$, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n \log n)$ respectively. [Probably some more info about space complexity, average case, expected case]

Since the problem of sorting numbers can be transformed into the problem of computing a Voronoi diagram in linear $\mathcal{O}(n)$ time, the Voronoi diagram can be constructed in optimal time $\theta(n \log n)$. (Cool fact: Asymptotically, the complexity of the constructing the diagram is no more difficult than the task of finding a region.)

A note on storage: There are many different ways to store a Voronoi diagram. Algorithms will occasionally suggest data structures which complement the algorithm approach and are space efficient. However, it is often at the discretion of the programmer to choose the structures most appropriate to their application's needs. In sum, these choices can range from devilishly clever [ref Guibas and Stolfi] to painfully ham-fisted. [I won't dwell on them in this chapter in any detail, but I may mention possibilities, especially if they appear in Shewchuck's implementation.] This much is true for any algorithm one might enounter: given a set of n unique sites, $P := \{p_1, p_2, ..p_n\}$, the output will be a handful of Voronoi vertices, V, and a bundle of Voronoi edges, E.

1.1 Intersection of Half-Planes: A Naïve Approach

The structure of the planar Voronoi diagram lends itself to relatively simple algorithm for its construction. Intuitively, the Voronoi diagram of a set of sites P, $Vor(P)$, can be found by computing each region V_{p_i}. If $|P| = n$, then no region can have more than $n - 1$ edges. Each region, then takes $\mathcal{O}(n \log n)$ time. [summarize this alg?]. A region needs to be computed for each of the n sites, so computing the Voronoi diagram with this approach would take $\mathcal{O}(n^2 \log n)$ time.

1.1.1 Constructing a Voronoi Region

Each Voronoi region is composed of edges and vertices. The vertices of the region are a subset of the Voronoi vertices. How can we determine the edges of a given region? Consider an arbitrary site p_i P (where $P := \{p_1, p_2, ..p_n\}$ is a set of n distinct points in $\Re^2$.) We've seen that the Voronoi region is a convex polygonal region. Futhermore, if $|P| = n$, then no region can have more than $n-1$ edges. Recall that an equivalent way of describing a Voronoi region is as the intersection of a set of half-planes. We can construct the region for a specific site by taking the intersection of this set of half-planes.

1.2 Shamos Algorithm: A Divide and Conquer Approach

[history... one of the first algs?]

1.2.1 Overview

[The Divide and Conquer algorithm is actually a really cool algorithm that splits up the voronoi sites (via a recursive function) until they are of trivial size, and then computes the voronoi for the trivial region and then merges the two (adjacent) regions. it's the merge part that's hard. so spend a good amount of time explaining that. Now talk about how shewchuck and his approach and also how this one of the main portions of the code we're gonna modify]

1.2.2 Divide and Conquer Algorithm Pseudocode

algorithm 1 and its supporting procedures, algorithm 4 and algorithm 5, provide a pseudocode implementation of Fortune's algorithm. The vertex and edge structure for $Vor(P)$ will use a classic doubly connected edge list (DCEL) [do i need to explain this] and we will use a balanced binary search tree (BBST) to maintain the beachfront, β. The event queue, Q, is a plain old priority queue.

Algorithm 1 Divide and Conquer Voronoi Diagram

```
1: procedure VoronoiDivConq($P$)                              ▷ Construct $Vor(P)$
2:     $P_L, P_R, \leftarrow$ PartitionSites(P)        ▷ Partition $P$ by median x-coordinate
3:     $Vor(P_L) \leftarrow$VoronoiDivConq($P_L$)                         ▷ Recursive call
       $Vor(P_R) \leftarrow$VoronoiDivConq($P_R$)
4:     VoronoiMerge($Vor(P_L)$, $Vor(P_R)$)                      ▷ Merge sub-diagrams
5: end procedure
```

Algorithm 2 Divide and Conquer Voronoi Diagram: Merge Voronoi

```
1: procedure VoronoiMerge($Vor(P_L)$, $Vor(P_R)$)
2: end procedure
```

1.2.3 Analysis of Space and Time Complexity

1.3 Fortune's Algorithm: A Sweepline Approach

[in the 1980's] Steven Fortune proposed a sweepline algorithm with asymptotic running time $\mathcal{O}(n \log n)$ and using $\mathcal{O}(n)$ space. [YOU NEED TO REVIEW THIS STUFF] Fortune's algorithm makes use of a sweepline, a horizontal line beginning at the top of the plane and descending to the bottom of the plane. The general strategy of the algorithm is to use a line to scan the plane, updating information about the structure of the diagram whenever the sweepline intersects a point of interest. Some of these points of interest are known in advance, and others arise during the construction of the diagram. When the sweepline has intersected each of these points of interest, a little tidying up gives the Voronoi diagram. With a bit of planning and the clever choice of a quad-edge data structure, the task of constructing the Voronoi diagram with this approach immediately yields its dual, the Delaunay triangulation.

1.3.1 Conceptual Overview

[Trim the fat here] Fortune's algorithm uses a few key concepts and structures to support the construction of the Voronoi diagram. To gain an understanding of how to algorithm works, it's easiest to apply it to a set of sites. Let $P := \{p_1, p_2, ..p_n\}$ be the a set of n distinct points in $\Re^2$. Then, define the sweepline s to be a horizontal line in $\Re^2$. [side note: Conceptually, s begins at the "top" of the plane, but in practice, it is sufficient that s be initialized with a y-coordinate greater than the site with the largest y-coordinate. If you include this, at least mention that a circle event might occur above the highest site event, and how to handle that]

An astute reader might wonder how exactly we construct the Voronoi diagram with nothing but a set of sites and a vague notion of a line that falls down the plane. After all, there's no obvious method by which the intersection of the sweepline with a point of interest might yield a Voronoi vertex or edge. Further, it would seem that even if we could provide such a relationship would be subject to change as soon as the sweepline encountered the next point of interest.

Herein lies the elegance of Fortune's algorithm. Consider the portion of the plane that the sweepline has already encountered, this portion constitutes a closed half-plane, s^+. If a site exists in s^+, then any point q in s^+ which is closer to that site than it is to s will not be assigned to a site below s. When the sweepline s encounters a site p_i we call this a *site event*. We consider the locus of points which are closer to p_i than to s. The set of points equidistant to s and p_i form a parabolic arc. This arc, denoted β_{p_i}, is added to a a sequence of parabolic arcs called the *beachfront*. We will refer to the beachfront as $\beta(P)$. The crucial observation is that the points above $\beta(P)$ will be assigned to a site already encountered by s.

[Make sure there's a clear difference between the notation used for the parabola and the notation used for an arc]. To understand how the beachfront plays a role in constructing the Voronoi diagram, let's consider times where the structure of the beachfront changes. New arcs are added to the beachfront whenever a site event occurs. At the moment where the new arc, β_{p_i}, is inserted into the beachfront, it is a simple vertical line (or a parabola of no width). If there is an arc above β_{p_i}, β_{p_i} will only intersect it at one point since the beachfront is x-monotone. To insert β_{p_i} into the beachfront, we split the arc above it into two parabolic arcs and insert β_{p_i} in between the split arcs. We record the points where β_{p_i} intersects with the arcs adjacent to it, and we refer to these points as *breakpoints*[4] As s moves downward, the breakpoints of each arc in the beachfront $\beta(P)$ change since the parabola that governs the arc widens. Another way to understand this is to recognize that that parabola whose corresponding arc is β_{p_i} has as its focus the site p_i and s as its directrix. If we were to draw lines that tracked the movement of the breakpoints for the beachfront as the sweepline descended, they would each trace out the Voronoi edges. So we know that arcs get added to the beachfront whenever a site event occurs, in fact, the *only* time an arc gets added to the beachfront is when a site event occurs. [And we can prove that!]

The second way the beachfront's structure changes is when an arc disappears. The disappearance of an arc coincides with the second type of event, called a *circle event*. We'll define the notion of a circle event by studying a small portion of the beachfront. Consider the arcs α, δ, and γ defined by three sites p_i, p_j, and p_k. Define the circle, C with the three sites p_i, p_j, p_k. [You should probably remind the reader why α and γ are not arcs from the same site, also why the circle contains no sites] Though the reasons aren't immediately obvious, we want to keep track of two points: the center of the circle, q, and the lowest point of the circle, l. If δ is the arc that is shrinking, when does it disappear? Recall that the center of the circle, q, is equidistant from p_i, p_j, p_k and l. When the sweepline, s, reaches l, the arcs correspoinding to p_i and p_k, α and γ, respectively, intersect at q. [Do I need to explain in more detail why, or can I *just* include a diagram]. At this point, the right breakpoint of α and the left breakpoint of γ become equal to the left and right breakpoints of δ, δ disappears. The edges traced out by the breakpoints of δ meet at q and q becomes a Voronoi vertex.

To summarize, a site event occurs when the sweepline intersects with a site (i.e they both have the same y-coordinate) and a circle event occurs when the sweepline intersects with the lowest point (that is to say, the point with the minimum y-coordinate) of a circle. Arcs are added to the beachfront when site events occur, and removed when circle events occur. The effect of an adding an arc is the growth of a new edge, and the effect of removing an arc is the addition of a vertex.

It's important to note that in this example, we assumed that p_i, p_j, p_k were non-collinear sites, since the circle defined by three collinear points would be a circle of infinite radius (that is to say, a line). Moreover, we glossed over the possibility that C contained any other sites. How would any number of sites on the interior of C affect the circle event itself?

To answer this question, let's first make a brief digression and introduce the second

[4]Note that this means that the right breakpoint of β_{p_i} is the same as the left breakpoint of the arc which is adjacent on the right. Similarly, the right breakpoint of the arc adjacent to β_{p_i} on the left is equal to β_{p_i}'s left breakpoint.

structure that Fortune's algorithm employs: the *event queue*. Armed with an understanding of how site events and circle events effect the structure of the beachfront, we can consider ourselves sufficiently motivated to define a structure that keeps track of these kinds of events. We know that the sweepline travels downward, handling site and circle events as it encounters them. In fact, we know a lot about the site events; since we know all the sites before the sweepline begins, we can simply order the sites by decreasing y-coordinate and initialize the event queue with that infomation. Unlike site events, we don't know exactly when or how many circle events will occur[5], so they complicate the problem insofar as they punctuate the otherwise predictable order of site events. What we do know is that we'd like to maintain an event queue, which consists of—and is ordered by—all of the site events and the known circle events. Circle events are added to event queue for every triple of consecutive arcs on the beachfront. So when a site event triggers a new arc on the beachfront, we need to check a couple of things, namely, whether or not the new arc a) invalidates one or more circle events already in the event queue; or b) creates one or more circle events. In this manner, handling a site event often has a cascading effect, wherein the event itself doesn't create a circle event, but it might create a situation where one or more circle events would need to added or removed from the event queue. Similarly, handling a circle event can also lead to the creation or removal of one or more circle events, since the disappearance of an arc can lead to new combinations of triples on the beachfront. Note that a circle event exists in the event queue if and only if it satisfies a) the property that the circle defined by the sites corresponding to the triple of arcs intersects the sweepline; and b) hasn't already been deleted from the event queue.

[Probably don't need this anymore .... Now that we have an event queue to keep track of the order of circle and site events, we can look back at our initial example. Let p_m be a site located somewhere on the interior of our original circle, C. As s crept downward, the site event corresponding to p_m would have been handled before circle event at l. As a result, an arc β would have been inserted somewhere between α, δ, and γ. More specifically, the arc directly above p_m would have been split into two arcs. The only exception to this would be the edge case where p_m's x-coordinate was exactly the same as either of the breakpoints of δ. In that case, the arc wouldn't have split δ; it would have been inserted directly in between δ and the appropriate adjacent arc. If this were the case, the triple of arcs whose corresponding sites had defined the circle would no longer exist, so the circle event would be invalid. In the former scenario, the new arc would have created at least one new triple of arcs, which would have prompted the creation of more circles, and thus cicle events. It's possible to show [do i have to?] that at least one of these new circle events would have been handled before our original circle event...]

[5]Though we can provide an upper bound, see **??**

1.3.2 Data Structures and Algorithm Pseudocode

algorithm 3 and its supporting procedures, algorithm 4 and algorithm 5, provide a pseudocode implementation of Fortune's algorithm. The vertex and edge structure for $Vor(P)$ will use a classic doubly connected edge list (DCEL) [do i need to explain this] and we will use a balanced binary search tree (BBST) to maintain the beachfront, β. The event queue, Q, is a plain old priority queue.

Algorithm 3 Sweepline Voronoi Diagram

```
 1: procedure VoronoiSweepline($P$)                ▷ Construct $Vor(P)$
 2:     $Q \leftarrow P$                 ▷ Initialize event queue, $Q$, with set of sites, $P$
 3:     $\beta \leftarrow \{\}$                  ▷ Initialize beachfront, $B$, as empty
 4:     while $Q$ is not empty do               ▷ There are more events to handle
 5:         $e \leftarrow Q.pop()$                ▷ Pop the next event off the queue
 6:         if $e$ is a site event then
 7:             HandleSiteEvent($e$)                        ▷ e is a site $p_i$
 8:         else
 9:             HandleCircleEvent($e$)          ▷ $e$ is a circle, $c_l$ with low point $l$
10:         end if
11:     end while
12: end procedure
```

Algorithm 4 Sweepline Voronoi Diagram: Handle Site Event

```
 1: procedure HandleSiteEvent($p_i$)
            Something compelling
 2: end procedure
```

Algorithm 5 Sweepline Voronoi Diagram: Handle Circle Event

```
 1: procedure HandleCircleEvent($c_l$)
            now blow their minds, this is cool stuff!
 2: end procedure
```

1.3.3 Final thoughts on Fortune's Algorithm

Though Fortune's algorithm wasn't the algorithm I spent the most time with on this thesis, it remains my favorite algorithm to witness. Fortune's algorithm reveals- more readily than any other I've encountered- the connection between sorting a set of numbers and computing the Voronoi diagram. [the connection] [perhaps a note on degenerate cases] [a note on space and time complexity]

1.4 Randomized Incremental Approach

1.4.1 Overview

1.4.2 Proof of Correctness

1.4.3 Analysis of Space and Time Complexity

Parallel Approaches to Constructing the Voronoi Diagram

2.1 Parallel(?) Merge Hull

[content content content maybe a diagram maybe some code]

2.2 Parallel(?) Randomized Incremental

Appendix A

(The Post Office Problem Re-framed)

[Imagine you are looking at a map of your campus

Imagine you are studying trees. You are lying in the grass and the shade is covering your face. There is another tree about a dozen feet away. we all know exactly how far away it is, but dozen sounds more prosaic. so it's a dozen feet away, and why is that? [area availiable to a tree]

You're walking the blocks of new york city and the wind is cold. Where is the nearest subway entrance? People are walking past you though, really, you're just shuffling. Where is the nearest subway with an N train? Is it a local or express line? Are you hungry, maybe you're hungry. Or maybe your're the one selling hot dogs. yes. hot dogs in new york city. where should you put your hot dog cart?]

References