A Survery of the Voronoi Diagram

1 Introduction

Voronoi diagrams are one of the most well studied structures in all of computational geometry. The Voronoi diagram takes a set of points and divides the plane (or higher dimensional space) into regions such that each region contains all points which are nearer to some source point than any other, as defined by some distance function. In this paper we describe the Voronoi diagram and some of its properties, give some algorithms to compute it efficiently, and discuss some of the variations on the diagram itself. We also explore the connection between the Voronoi diagram and the Delaunay Triangulation, as well as some applications of the Voronoi diagram outside.

2 The Standard Voronoi Diagram

The most common type of Voronoi diagram lies on the two dimensional plane, $\mathbb{R}^2$, and uses the Euclidean distance function to determine proximity. Thus, the regions which make up the Voronoi diagram effectively partition the plane. The discussion in this section will regard this class of Voronoi diagram, though many of the concepts introduced here remain in a similar form in higher dimensions.

2.1 Definition

We begin by specifying the general definition of a Voronoi diagram. Though there exist variants on the Voronoi diagram for which the following definition will not exactly hold, all Voronoi diagrams follow a similar definition, and can alter or augment the following description to fit their problem space. Let S be a set of n points in the plane, in general position. These points are the 'sites' or 'generators' for the Voronoi diagram. In this paper, we will refer to these points as generators for simplicity. The assumption that the generators are in general position means that there are no three collinear points, nor are there any four co-circular points. Many algorithms make this assumption, with the claim that handling these cases is done rather simply. Now, for any two distinct generators, p and q in S, have that the *dominance* of p over q, $dom(p, q)$ is the portion of the plane which is at least as close to p as it is to q. That is, the Euclidean distance from each point in $dom(p, q)$ to p is no greater than the distance of each point to q, respectively. Formally, we have the following:

$$dom(p, q) = \{x \in \mathbb{R}^2 \mid d(x, p) \leq d(x, q)\} \text{ [6]}$$

Where d represents the Euclidean distance function. This dominance relationship allows us to define the *region* of the plane for which each point in the region of a particular generator p, $reg(p)$, is closer to p than to any other generator in S. $dom(p, q)$ is a closed half-plane defined by the perpendicular bisector of the line segment $\overline{pq}$. $reg(p)$, then, is defined as the subset of the plane of which p has dominance over every other generator in S. This subset of the plane is also known as the Voronoi cell of p. That is to say that $reg(p)$ represents the intersection of the dominance of p over each other generator. Formally, we have:

$$reg(p) = \bigcap_{x \in S - \{p\}} dom(p, x) \text{ [6]}$$

Since each component of $reg(p)$ is a closed half-plane, the resulting region must be a convex polygon. Specifically, this polygon will have no more than $n-1$ edges and vertices, since there are n generators in S, one of which is p. The regions of all generators in S form a partition of the plane, with overlap on the edges and vertices of each polygon. Note that every point on each edge of a polygon is equidistant from at least two generators, and each vertex is similarly equidistant from at least three generators. Because of this, the regions of each generator line up edge to edge and vertex to vertex. Thus, the entire plane is exactly divided by the collection of edges and vertices in all the regions of the generators of S. This partition of the plane is called the Voronoi diagram of S, $Vor(S)$.

2.2 Properties

We typically refer to the Voronoi diagram of a finite point set as the collection of three subsets of the plane: faces, edges, and vertices. The union of these three subsets is the plane itself. That is, each point in the plane is either on a face, on an edge, or defines a vertex. Each face in $Vor(S)$ is essentially the region of a generator of S, excluding the border. This is defined the same way as above, but using a strictly less than comparator in the dominance relation. Each edge is the border between two faces, excluding the vertices at its endpoints, if they exist. Some edges in the Voronoi diagram may have one or zero vertices at their endpoints. This implies that the polygon which contains such an edge is semi-infinite, as one of its edges is unbounded. Specifically, the polygons which are associated with generators that lie on the convex hull of S are unbounded. Any generator g which defines a vertex of the convex hull cannot have a bounded polygon, since there can be no generator in S which has dominance over g in portion of the plane which lies away from the convex hull, as shown in the figure below.

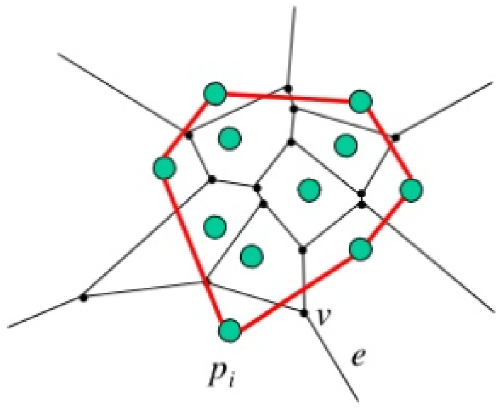

Figure 1: An overlay of the convex hull on a section of the Voronoi diagram, illustrating the unboundedness of polygons associated with generators on the convex hull [29]

Another important property of Voronoi diagrams that is useful in both storing and computing the diagrams is the linearity of their size. Although there are $\binom{n}{2} = O(n^2)$ possible edges on a set of n points, we have two facts based on the construction of the Voronoi diagram which gives a linear upper bound to the number of edges and vertices in $Vor(S)$. First, since $Vor(S)$ defines a partition of the plane, Euler's relation $n + v - e = 2$ holds, with n equal to the number of generators of S, and therefore equal to the number of faces of $Vor(S)$, and v, e equal to the number of vertices and edges in $Vor(S)$ respectively. Secondly, we know that each edge has at most two vertices, and each vertex is shared by at least three edges. Therefore, we gain the additional relation $2e \geq 3v$.

Combining these two relations gives us that $e \leq 3n - 6$ and $v \leq 2n - 4$. Since the Voronoi diagram is defined by the collection of faces, edges, and vertices, and since all of these quantities are linear with respect to the number of generators, n, the size of a Voronoi diagram is also linear.

2.3 Algorithms

Early algorithms for computing Voronoi diagrams involve computing the diagram incrementally, by adding one generator at a time, updating the regions of each existing generator at each step. These algorithms are fairly simple, but have a runtime bound of $O(n^2)$. The first algorithm which obtained the optimal time bound for computing the Voronoi diagram of $O(n \log(n))$ was published in 1975 by Shamos and Hoey [32]. This bound is optimal since the problem of sorting n real numbers can be reduced to the problem of computing a Voronoi diagram on $n + 1$ points easily. Given reals $x_1, x_2, \cdots, x_n$, simply compute the Voronoi diagram of the points $(x_1, 0), (x_2, 0), \cdots, (x_n, 0), (0, \infty)$. Then, a simple trace of the edges of the polygon associated with the generator at $(0, \infty)$ will provide the sorted order of the original set of reals. Thus, $O(n \log(n))$ is the optimal time bound for computing a Voronoi diagram on n points in the plane.

2.3.1 Divide and Conquer

Given the time bound of $O(n \log(n))$, it is natural to guess that there might be a divide and conquer solution to the problem of constructing a Voronoi diagram on n points in the plane. In fact, the original solution proposed by Shamos and Hoey [32] is just that. The algorithm proceeds in a fairly conventional manner. First, the points of set S are presorted into a list by x-coordinate. Then, the points are split into two groups by a vertical line, with half of the points in each group, say S_l and S_r to indicate the left and right group respectively. Now, in traditional divide and conquer fashion, the diagram for each of these two subsets is computed recursively, with a base case of $n \leq 3$ used to compute the diagram by brute force. The final step is by far the most involved and interesting, again typical of a divide and conquer algorithm. The 'conquer' phase involves merging the diagrams computed for the subsets into one final diagram. Since the two diagrams being merged are indeed Voronoi diagrams, they must partition the plane. Thus, any point x in $\mathbb{R}^2$ must lie in the region of some generator p, $reg_{S_l}(p)$ [1] where $p \in S_l$, as well as the region of some generator q, $reg_{S_r}(q)$ where $q \in S_r$. In the complete diagram, $Vor(S)$, it must be that $x \in reg_S(p)$ if $d(x, p) < d(x, q)$, and similarly for $reg_S(q)$. So, we need to trim the regions of p and q along the perpendicular bisector of these two generators to obtain their appropriate regions in $Vor(S)$. This involves adding additional edges along these bisectors to form a component edge of the *merge chain*. Performing this process for each relevant pair of generators p, q completes the merge process. Figure 2 below illustrates the effects of merging two diagrams into one.
Now, it is important that this merge step take only linear time in the number of generators of S, in order to obtain the optimal time bound. One property of the merge chain allows us to do this, namely that it is y-monotone. Since the merge chain divides the regions of generators in opposite sets S_l and S_r, it must be that each horizontal line in the plane intersects the merge chain in only one place. If it did not, then some generator in S_r must be to the left of a generator in S_l, which would cause the merge chain to go back on itself (in the y-direction). This contradicts the fact that these two sets are separable by a vertical line, so the merge chain is y-monotone. Now, this directly implies that the merge chain is connected and unbounded. So we can form a series of edges, one at a time, to make the merge chain. We start with the unbounded edge corresponding to some generator on the convex hull of S. As stated above, the regions for these generators are unbounded,

[1] Here, $reg_{S_l}(p)$ refers to the region of generator p with respect to the point set S_l

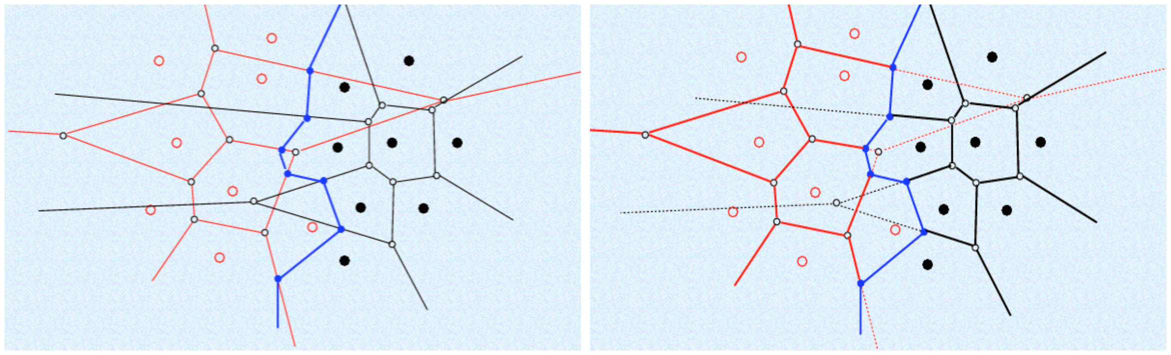
Figure 2: The left diagram shows $Vor(S_l)$ in red, $Vor(S_r)$ in black, and the merge chain in blue. The right diagram shows $Vor(S)$ using solid lines, and uses dotted lines to indicate the portions of edges which were trimmed by the merge. [25]

and so have unbounded edges. So, we simply find a pair of generators, p, q such that $p \in S_l$ and $q \in S_r$, and p, q are neighboring points on the convex hull of S, which takes $O(n)$ time using the tangents to the convex hulls of the subsets. Adding the remaining edges also takes linear time, since we need only trace the previous edge until it intersects with the edge which would separate the regions of p and q. Since each edge we trace is either ignored or shortened into an edge of $Vor(S)$, we have a linear time bound on the merge operation. [6] Now, the recurrence relation for this algorithm is $T(n) = 2T(\frac{n}{2}) + O(n)$, where $T(n)$ is the time complexity of computing a Voronoi diagram of size n. This relation is solved by the Master Theorem to give $T(n) = O(n\log(n))$.

2.3.2 Fortune's Algorithm

Fortune's Algorithm presents an optimal solution to the problem of constructing a Voronoi diagram using a line-sweep (hyperplane-sweep in higher dimensions) algorithm. This algorithm uses a priority queue, Q to hold initially all the generators of S, ordered by non-decreasing y-coordinate, and later to also hold intersection points of region boundaries. This priority queue realizes the sweepline, since the algorithm begins at the lowest generator $p \in S$, and proceeds upwards, taking generators and region boundary intersections as event points. It also uses a list L which initially contains the region of the generator with the lowest y-coordinate, p, and later contains regions and their boundaries to other regions. Fortune's algorithm also makes use of a mapping of the plane he calls '*'. [16] It is this mapping, which maps bisectors into hyperbolas, and vertical lines to vertical lines, that is crucial for the algorithm to work. The algorithm makes extensive use of this transformation during the sweep, adding points into Q based on the properties of transformed bisectors. At each step, depending on whether the minimum point in Q is a generator or a region intersection, various other points are added into and removed from both Q. It also updates the list L with relevant regions and bisectors. Analysis of Fortune's algorithm gives the optimal time bound of $O(n\log(n))$, and also maintains the optimal $O(n)$ spatial requirement. Compared to the divide and conquer method, the algorithm itself is slightly simpler, since the complications involved in the merge step are not present here. However, there is also a significant degree of mathematical support required in this algorithm, mainly in the mapping, '*', and it's prevalence in the core functional steps of the algorithm.

2.3.3 Linear Time

While the optimal time bound for computing a Voronoi diagram on a set S of n points in general position is $O(n\log(n))$, a faster bound can be obtained with a special case on the input set S. Aggarwall et al. [3] demonstrate a linear time algorithm for computing the Voronoi diagram if the points of S are the vertices of a convex polygon, given in, say, counterclockwise order. They use the lifting map described by Guibas and Stolfi [19], which maps a point (x, y) onto the paraboloid $(x, y, x^2 + y^2)$, and thereby simplify the problem. The process begins by computing the convex hull of the 'lifted' points in space, which correspond to the generators on the plane. Specifically, the algorithm they provide computes the upper hull of the lifted points, while the lower hull is used for the Voronoi diagram construction. The convex hull algorithm itself has other applications as well, since it reduces the $O(n\log(n))$ bound for that problem to a linear one as well, for the same special case in which the points form a convex polygon. Once the lower hull is obtained, the key observation is that the lower hull of the lifted points in space maps back to the Delaunay Triangulation (discussed in section 4) of S. Using the duality of the Delaunay Triangulation with the Voronoi diagram, we can construct the Voronoi diagram from the Delaunay Triangulation in linear time. Since the computation of the convex hull, the mapping back to the plane, and the transformation into the Voronoi diagram are all linear operations, it follows that the Voronoi diagram of n points which define the vertices of a convex polygon can be computed in linear time. Another algorithm presented by Ohya et al. [26] gives a version of the incremental algorithm which, while it has a worst case performance of $O(n^2)$, has an expected runtime of $O(n)$ by performing local changes to the diagram during execution. The crux of this algorithm is to find a way to order the generators such that each step only requires constant updates to the diagram, on average. The worst case for this algorithm only occurs when each generator added in an incremental step affects all existing Voronoi regions. Ohya et al. state that, though the worst case runtime is indeed quadratic, the algorithm runs in about $O(n^{\frac{3}{2}})$

3 Variants

We can think of the ways to alter this vision of a Voronoi diagrams in several fundamental ways. We assumed that our Voronoi diagram was for point generators in the plane with the Euclidean norm used to define the dominance of p of q. That already suggests ways to alter the Voronoi Diagram Problem: What if the generators are not points, what if the generators do not lie in the plane, and what if we aren't using the Euclidean norm? Another less obvious variant involves changing what each cell represents. Instead of the points in $reg(p)$ being those points whose strictly nearest generator is p, consider a similar Voronoi region as the set of points who share their nearest k neighbors. This is the called the k-th order Voronoi diagram. For each of these four cases we will discuss some important properties and describe how to construct this altered Voronoi Diagram as quickly as possible.

3.1 Different objects as generators

Traditionally, we are given n points as generators for a Voronoi diagram, but we can consider instances where the generators are more complex shapes. The distance between a point p and an object O is $min_{q\in O}(d(p, q))$ Yap gives us an optimal solution to find the Voronoi diagram of n circular segments(i.e. arcs of a circle) and line segments in a $O(n\log n)$ time [33]. This algorithm is based on the divide and conquer algorithm. Fortune's algorithm can also be used to solve the line segment diagram problem in the same time and space with a sweepline algorithm [16]. We assume that lines may intersect but not at endpoints. An important note is that a bisector between two line segments of circular segments are not guranteed to be a single straight line but instead can be made of hyperbolas, parabolas, and straight line components. The total complexity of a diagram in this setting is $O(n)$.

3.1.1 Constructions

Yap [33] gives an $O(n \log(n))$ divide and conquer algorithm that works for sets of line segments and curved segments. This algorithm works by taking each of the m endpoints and placing $m + 1$ vertical lines such that each endpoint lies uniquely between two vertical lines. They call the area between two vertical lines a slab. The algorithm works up by merging adjacent slabs in $O(\log(n))$ steps and show that the cost of a merge between two slabs can be done in time $O(k)$ where k is the number of end points belonging to the two slabs being merged or $O(n)$ for 1 merge step giving us our bound.

3.2 Higher Order Diagrams

In the standard Voronoi diagram formulation the set of points in a cell are defined by the fact that they all share the same nearest generator. We can extend this idea so that each Voronoi cell is the set of points that share their k nearest generators. The standard terminology is to call such a diagram an order k or kth order Voronoi diagram. For example, k=1 corresponds to our familiar standard Voronoi Diagram which is also called the nearest point Voronoi diagram or the first order Voronoi diagram. The second order diagram is composed of cells where each point in a cell share have the same nearest two generators, and the n-1st order diagram, also known as the farthest point Voronoi diagram, describes a diagram in which each cell is defined by its $n - 1$ closest generators. Formally speaking, given a set G of generators and a subset H of G with k points then the kth order Voronoi cell is $Vor_k(H, S) = \{x | \forall h \in H, \forall g \in G \setminus H, d(h, x) < d(g, x)\}$ where $d(\cdot, \cdot)$ is the distance function which unless otherwise noted for this section is the L_2 norm. In this section we will also look at further extension where the set of generators is allowed to be composed of line segments.
What are the important structural differences between the first order diagram and the other orders of diagrams? The first key difference is how the complexity of the kth order Voronoi diagram is different than the standard diagram. For the 2d Voronoi diagram the complexity, that is the maximum number of cells, edges, and vertices in the diagram, is $O(n)$ but in the 2D kth order diagram the complexity $O(k(n - k))$ [22]. The complexity for non-intersecting line segments is also $O(k(n - k))$ [28]. Unfortunately when we move to line segments we miss out on an important property: In the kth order Voronoi diagram with lines segments as generators, an individual Voronoi cell may be disjoint [28]. In fact when using lines as the generators, one cell can be divided into $\Theta(n)$ faces in a malicious case.

3.2.1 Constructions

A common trend for dealing with Voronoi diagrams in $\mathbb{R}^n$ is to do a transformation to a different problem usually in $\mathbb{R}^{n+1}$. Edelsbrunner et. al. [15] describe the relation that we will use between the nth dimension and n+1th problem that is easier. Furthermore, there is a well known transformation of a concept called the k-level arrangement of hyperplanes in $\mathbb{R}^{n+1}$ to the Voronoi diagram in $\mathbb{R}^n$ [30] [9] [3]. But what is the kth level arrangement of hyperplanes? Given a set H of n hyperplanes, the kth level of our space is $Level_k(H) = \{p | \text{At most k hyperplanes are strictly below p}\}$. For our problem case of the 2d dimmensional Voronoi, Chan gives a randomized algorithm to find the k-level of a planes in $\mathbb{R}^3$ that runs in expected time $O(n\log(n) + nk\log(k))$ [8] which can be reduced to the same bounds for the Voronoi Diagram. Chan also provides an educational table on the history of this problem which we summarize:

Year	*Bound*	*Source*
1982	$O(nk^2\log(n)$	[22]
1986	$O(n^3)$	[15]
1987	$O(nk\sqrt{n}\log(n))$	[10]
1989	$O(n\log(n) + nk^2$	[3]
1995	$O(n^{1+\epsilon}k)$	[2]
1998	$ExpectedO(n\log^3(n) + nk\log(k))$	[1]
2000	$O(n\log(n) + nk\log(k))$	[8]

3.3 Different distance functions

Yet another variant of a Voronoi Diagram to consider if we replace the Euclidean norm we have been using with any distance function in the plane. Of particular research interest, are the L_1 metric or Manhattan Distance: $d_1(p,q) = ||p_x - q_x| + |p_y - q_y||$ and the L_∞ metric: $d_\infty(p,q) = max(|p_x - q_x|, |p_y - q_y|)$. Chew and Dyrsdale [12] give a particularly nice talk on diagrams with alternate distance functions. One way to think of an arbitrary distance metric diagrams is to imagine dropping n stones in into a still pond at exactly the same time and a Voronoi edge is where to set of ripples meet, a vertex where 3 or more meet, and a region where it is one stone's ripples that are dominant. But to turn this into a different metric, we replace a circle, the shape that has a border which is equidistant from a point in Euclidean space, to the shape whose border is equidistant to a point in that metric. For example in L_1 it is a diamond and in L_∞ it is a square. This view of a Voronoi cell being formed by an expanding wave is a helpful imagine to keep in mind. Chew et. al. go further and discuss what happens if your distance function is not even a proper metric. They prove that when considering the Voronoi diagram of n point with distance function d, the cells will be star shaped if and only if the triangle inequality holds with respect to d. Additionally, they say that a Voronoi cell for a point generator is a single piece if the distance function is convex. If this is the case, then the vertices, edges, and faces will all be $O(n)$.
We will now talk about a more niche distance function but not give an explicit algorithm to construct them here. One type of distance function called the skew distance which considers a slanted plane where the vertical distance we travel matters. Aichholzer et. al. develop this idea [5]. Notice that the difference in height between points p and q is $p_z - q_z = sin(\theta) * (p_y - q_y)$ where $sing(\theta)$ is the angle the between this slanted plane and the xy plane. They give the distance function d with parameter k to be $d_k(p,q) = E(p,q) - k(p_y - q_y)$. This curious distance function as some interesting properties. It is not symmetric and can be negative if $k > 1$ but does obey the triangle inequality. Furthermore consider the set of points that are 0 distance away from point p under different k. If $k < 1$ then this set is just p, if $k = 1$ it is a ray, and if $k > 1$ it is two rays with slopes $\frac{\pm 1}{\sqrt{k^2-1}}$. They also briefly discuss an algorithm to construct the Voronoi diagram in $O(n\log n)$.

3.3.1 Constructions

Chew et. al. [11] give an $O(n\log n)$ time based off of the divide and conquer algorithm described above. But it is important to note that one can't directly apply the algorithm or else might run into some pitfalls. First off, the 'bisector' of two points in the L_1 metric can actually be a region and not just a a curve [23]. We can get around this issue by breaking the tie in this region consistently. In figure 3, they choose the vertical line of these regions to by the de facto bisecting line. The bisecting line between two points can be multiple lines like in this example, if our the unit-distance shape(i.e. a diamond in L_1 or a circle in L_2) for our distance function has corners, points without a well defined tangent.

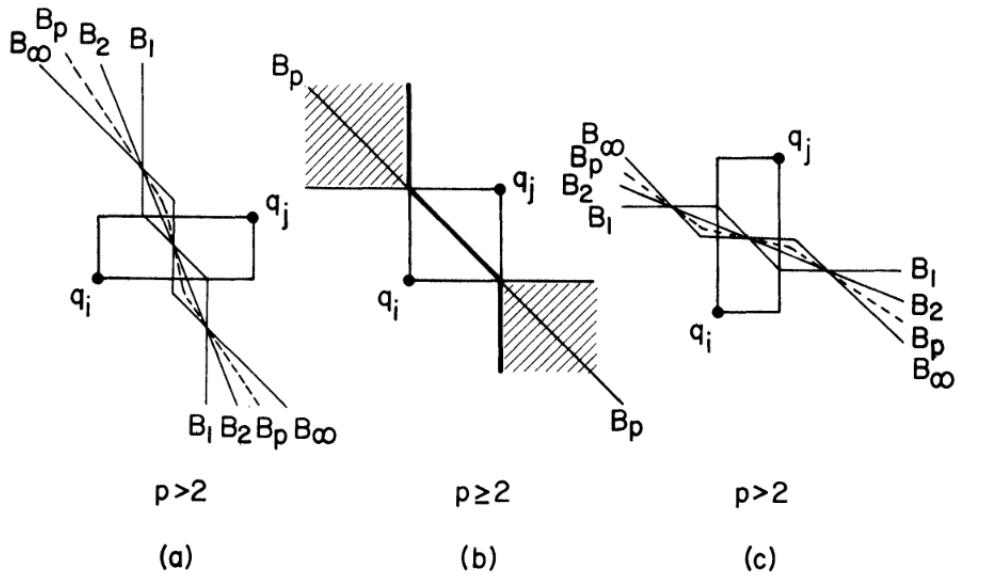

FIG. 2. ***Bisectors in different metrics.***

Figure 3: From lee et. al. [23]. The shaded regions in the middle figure represent an area of points with the L_1 that are equidistant from to generators

The next issue to tackle is who to pick our starting segment in the merge step of between the left and right regions $Vor(Left)$ and $Vor(Right)$ for merge chain. In Euclidean space, we use the convex hull to help us find but in the general L_p metric points corresponding to unbounded cells aren't necessarily on the convex hull. Chew et. al. give an example.

But we can find our starting segment in linear time with the following procedure outlined by [12]. The core idea of this process being that if b is a bisector of the left and right regions and extends to infinity then it is part of a bisector of exactly one point in the left region and one in the right region. So we can check in linear time all the pairs of points in the left and right regions whose intersection has infinite area if the bisector lies in this region.

1. Find all pairs (L_i, R_i) where L is in $Vor(Left)$ and R is in $Vor(Right)$ and the intersection of L and R has infinite area
2. For each pair L_i, R_i do:
   (a) Let p_l be the point generating L_i and p_r is the point generating R_i
   (b) Find the bisector $b(p_l, p_r)$ between p_l and p_r
   (c) If this bisector extends to infinity in the intersection of L_i and R_i then it is the bisector we are looking for and we return it

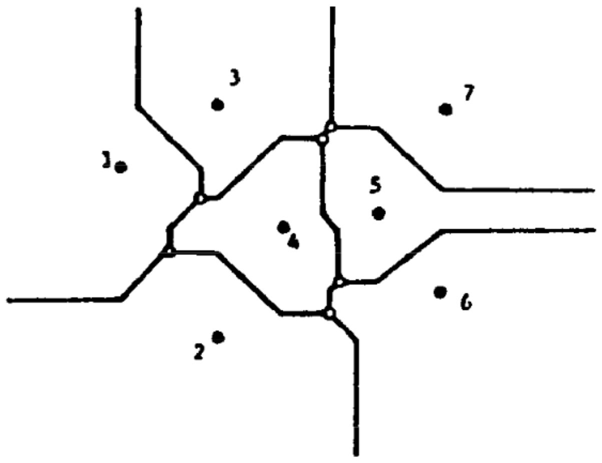

Figure 4: An example of the L_1 metric having an unbounded region belonging to a point not on the convex hull

Since we are only considering distance functions where there are $O(n)$ edges, we can now do the rest of the merging step in $O(n)$ in the same fashion that we did in the traditional divide and conquer algorithm, taking a total of $O(n \log n)$ time.

3.4 Higher Dimensions

For much of this discussion on all the variations of Voronoi diagrams we have not discussed a rather straightforward in concept variant: the higher dimension Voronoi diagram. This this the Voronoi diagram in $\mathbb{R}^n$ where $n \geq 3$. It is known that the complexity of a Voronoi diagram in 3d is $O(n^2)$. But this is a somewhat pessimistic bound and is a fairly large bound of research on if you take a randomly generated set of points(i.e. they all lie in a cube or sphere etc) what is the expected complexity and show in many cases the expected complexity is better than $O(n^2)$. Golin and Na show that if you take your points from the surface of closed convex polytope from a Poisson distribution then the expeded complexity is $O(n)$ [17]. Given a closed convex polytope P we define the convex distance between a point w and a line l to be $\inf(t \geq 0 | (w+tP) \bigcap l \neq \emptyset)$. If our polytope has k edges then it has been show that the complexity of a set of lines in 3 dimensional space is bounded by $O(n^2 k \log(n))$ [13]. We will see a construction of the nth dimensional Voronoi diagram that uses a transformation.

3.4.1 Constructions

First we show how to construct the 1st order 3 dimensional Voronoi diagram in an output sensitive manner by finding the convex hull in 4 dimension. For higher dimensions the same type of transformation holds but the convex hull is harder to calculate in higher dimensions. Recall that in 2 and 3 dimensions the maximum number of faces on the convex hull is $O(n)$ but in general the number of faces is $O(n^{\lfloor d/2 \rfloor})$. Chan et. al. [9] find an algorithm for the convex hull for four dimensions(which we can use to get the 3D Voronoi diagram) that if f faces need to be reported can be done in $O((n+f)\log^2(f))$ and $O(n+f)$ space. We define a map from a point p in $\mathbb{R}^n$ to a halfspace in $\mathbb{R}^{n+1}$ as follows: $p* = \{\overline{x} \geq 2p_1x_1 + 2p_2x_2 + \cdots + 2p_nx_n - p \cdot p\}$ [9]. To now compute the d dimensional Voronoi diagram we did to find the intersection of these $d+1$ dimensional halfspaces and project back down to d dimensions. The dual problem of computing the intersection of half spaces is computing the convex hull so we can now apply Chan's algorithm to find the Voronoi diagram in three dimensions in $O((n+f)\log^2(f))$ and $O(n+f)$ space.

4 Delaunay Triangulation

The Delaunay Triangulation is the dual of the Voronoi diagram. Where the Voronoi diagram divides the plane into regions for which all points in a region are nearer to the generator in that region than to any other generator for a set of n generators, S, the Delaunay Triangulation is a triangulation of S for which the minimum interior angle of the triangles is maximized. We call this triangulation the dual of the Voronoi diagram because one can be easily constructed from the other. Specifically, given either the Voronoi diagram or the Delaunay Triangulation, the other can be computed in $O(n)$ time. Since the Delaunay Triangulation maximizes the minimum interior angle of the triangles, it discourages 'poorly shaped' triangles which are long and thin. This makes the Delaunay Triangulation a good candidate for use in modeling applications, such as the modeling of terrain. The 'well-shaped' triangles provided by the Delaunay Triangulation allow for some topological guarantees that are useful in this setting. [18]

4.1 The Dual Graph

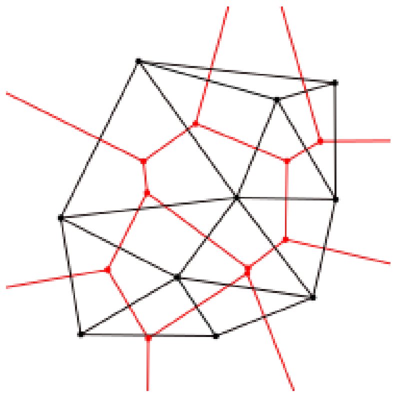

Figure 5: The Delaunay Triangulation (black) and the corresponding Voronoi diagram (red)

A key property of the Delaunay Triangulation is the way in which edges are drawn. An edge will be drawn between two generators if and only if these two generators share an edge in the Voronoi diagram. Another property, called the 'empty circle property', gives also that for any triangle in the Delaunay Triangulation, the circumcircle of the three generators which define its vertices will be empty. Note that the border of this circumcircle is not included. In fact, if there are $k \geq 4$ co-circular points, the result will be a k-sided polygon in what is called the Delaunay Graph, which is then triangulated according to the property that the minimum interior angle is maximized (in the case of the Delaunay Triangulation being constructed without the Voronoi diagram). Typically, the construction of the Delaunay Triangulation assumes that the generators of S are in general position, and therefore there are no more than three co-circular points, but as with the Voronoi diagram, the algorithms can be easily modified to handle those cases as well. Here, we will discuss the process to construct the Voronoi diagram from the Delaunay Triangulation, and vice versa.

We begin by constructing the Delaunay Triangulation from the Voronoi diagram. At first glance, it would seem that we need to check all $\binom{n}{2}$ generator pairs to check if there is an edge between the regions of each pair in the Voronoi diagram. However, since the size of the Voronoi diagram is $O(n)$, and specifically, the number of edges is $O(n)$, we can simply go the other direction, and check for each edge in the Voronoi diagram which two generators define the regions separated by that edge, and draw the corresponding edge in the Delaunay Triangulation. This procedure is obviously $O(n)$ as well.

We make a similar argument for the construction of the Voronoi diagram from the Delaunay Triangulation. Consider any triangle in the Delaunay Triangulation. Now, by the empty circle property, the circumcircle of the three generators must be empty, which means that the three generators on its circumference are the closest generators to that point. Also, since each generator is on the circumference of the circumcircle, they are equidistant from its center. This means that the centers of the circumcircles of each triangle in the Delaunay Triangulation define the vertices of the Voronoi diagram. To define the edges of the Voronoi diagram, we must observe one additional fact about the Delaunay Triangulation. If two triangles share an edge in the Delaunay Triangulation, then the two generators which define that edge are equidistant from the centers of two different circumcircles, namely the circumcircles corresponding to each of the triangles. Thus, the centers of these two circumcircles lie on the perpendicular bisector of the edge in the Delaunay Triangulation. This bisector defines all the points equidistant from both generators. Since the circumcircles are necessarily empty, it is the case that all points on the segment connecting the centers of these circumcircles are nearer to these two generators than to any other generator, and equidistant from both generators. Therefore, this segment will be an edge in the Voronoi diagram. Additionally, the edges which define the convex hull of the Delaunay Triangulation are used to generate the semi-infinite edges in the Voronoi diagram. Since the number of edges in the Voronoi diagram is $O(n)$, the number of edges and therefore triangles in the Delaunay Triangulation is also $O(n)$, and thus the construction of the Voronoi diagram from the Delaunay Triangulation is $O(n)$. For this reason, the bound on construction of the Delaunay Triangulation without the Voronoi diagram is $O(n\log(n))$, by the same reduction as given above.

4.2 Construction Optimization

Given the linear time construction of the Delaunay Triangulation from the Voronoi diagram, obviously the strategies given in section 2 will work to provide a Delaunay Triangulation for a convex polygon in linear time, using the Voronoi diagram as an intermediary. However, there has also been some research done into improving the $O(n\log(n))$ bound for the construction of the Delaunay Triangulation from scratch. Dwyer [14] gives an algorithm with expected runtime $O(n\log(\log(n)))$ for a large class of distributions of generators. This algorithm works by partitioning the unit square into $O(\frac{n}{\log(n)})$ squares, and then performing a similar lifting transformation as above, followed by a merge process to complete the triangulation at a global level. Note that this algorithm uses the probability distributions of the point-set S to determine the expected case runtime, but the worst case remains to be the optimal $O(n\log(n))$ bound.

5 Applications

All the work done on Voronoi diagrams would be disappointing if there were no practical use of a Voronoi diagram. Luckily, there is a substantial set of applications of Voronoi diagrams. The full range of appliations of Voronoi diagrams extends to robotics [6], VLSI [27], imaging [7], networking [24] and beyond. We will go over just a several notable applications here.

5.1 The post office problem

The post office problem was originally defined by Knuth in 1973 [21]. The problem is given a set of n sites and later a query find out which site is closest to a given query point. If we are considering points in the plane then this problem has the application of given n records with 2 attributes and a separate query record find the record that is most similar to this target query. But clearly, we can solve this problem by building the Voronoi diagram and given a query point return what Voronoi cell it belongs to. Shamos was the first to realize this connection which turns the post office problem in a point location problem [31]. It has been shown that this can be done $O(\log(n))$ and $O(n)$ space which an efficient solution to this problem, for example Kirkpatrick [20].

5.2 Largest empty shape

The largest empty shape problem is given a set of points in the plane and a shape find the largest instance of that shape that contains none of the given points. This problem has applications to industrial engineering and facility location. An interesting example is to consider a sheet of fabric with n defects. We would like to salvage the large piece of a predetermined shape that we can that has no defects. By shapes we mean any convex distance function but first let of consider the case where the shape is a circle. This problem can be easily solved in $O(n \log(n))$ time by noticing that the largest circle is either centered at a Voronoi vertex or at the intersection of a Voronoi edge and the convex hull. It takes $O(n \log(n))$ to build the Voronoi diagram and an additional $O(n)$ time to look at where each possible circle could be place. This was shown by Shamos and Hoey in 1975 [32]. Now we extend this idea to having a different convex shape other than a circle. We can solve this problem in the same way but now we replace the distance function from the Euclidean distance to a distance function in which that shape is the unit distance shape. For example if we want the largest empty square we can use the L_∞ metric instead of the L_2 metric. A harder but related problem is to find the largest empty axis parallel rectangle since we don't know what distance function to use before hand. If one is determined to use Voronoi diagrams albeit a generalized type of diagram this problem can still be solved $O(n \log^3(n))$ [10] but Aggarwal and Suri in 1987 showed that there is a better algorithm that takes only $O(n \log(n))$ time [4].

References

[1] Pankaj K Agarwal, Mark De Berg, Jirí Matousek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM journal on computing*, 27(3):654–667, 1998.

[2] Pankaj K. Agarwal and Jirı Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

[3] Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, Dec 1989.

[4] Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the third annual symposium on Computational geometry*, pages 278–290. ACM, 1987.

[5] Oswin Aichholzer, Danny Z Chen, DT Lee, Asish Mukhopadhyay, Evanthia Papadopoulou, and Franz Aurenhammer. Voronoi diagrams for direction-sensitive distances. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 418–420. ACM, 1997.

[6] Franz Aurenhammer. Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[7] Etienne Bertin, Franck Parazza, and Jean-Marc Chassery. Segmentation and measurement based on 3d voronoi diagram: application to confocal microscopy. *Computerized Medical Imaging and Graphics*, 17(3):175–182, 1993.

[8] Timothy M Chan. Random sampling, halfspace range reporting, and construction of\lowercase (k)-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.

[9] Timothy M Chan, Jack Snoeyink, and Chee-Keng Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional voronoi diagrams. *Discrete & Computational Geometry*, 18(4):433–454, 1997.

[10] Bernard Chazelle, RL Drysdale, and DT Lee. Computing the largest empty rectangle. *SIAM Journal on Computing*, 15(1):300–315, 1986.

[11] L Paul Chew. Building voronoi diagrams for convex polygons in linear expected time, 1990.

[12] L Paul Chew and Robert L Scot Dyrsdale III. Voronoi diagrams based on convex distance functions. In *Proceedings of the first annual symposium on Computational geometry*, pages 235–244. ACM, 1985.

[13] L Paul Chew, Klara Kedem, Micha Sharir, Boaz Tagansky, and Emo Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *Journal of Algorithms*, 29(2):238–255, 1998.

[14] Rex A. Dwyer. A faster divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, 2(1):137–151, Nov 1987.

[15] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.

[16] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153, 1987.

[17] Mordecai J Golin and Hyeon-Suk Na. On the average complexity of 3d-voronoi diagrams of random points on convex polytopes. *Computational Geometry*, 25(3):197–231, 2003.

[18] M Gopi, Shankar Krishnan, and Cláudio T Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Computer Graphics Forum*, volume 19, pages 467–478. Wiley Online Library, 2000.

[19] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM transactions on graphics (TOG)*, 4(2):74–123, 1985.

[20] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[21] Donald E Knuth. The art of computer programming, vol. 3: Sorting and searching addison, 1973.

[22] Der-Tsai Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE transactions on computers*, 100(6):478–487, 1982.

[23] Der-Tsai Lee and CK Wong. Voronoui diagrams in l_1(l_) metrics with 2-dimensional storage applications. *SIAM Journal on computing*, 9(1):200–211, 1980.

[24] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1380–1387. IEEE, 2001.

[25] Rashid Bin Muhammad.

[26] Takao Ohya, Masao Iri, and Kazuo Murota. Improvements of the incremental method for the voronoi diagram with computational comparison of various algorithms. *Journal of the Operations Research Society of Japan*, 27(4):306–337, 1984.

[27] Evanthia Papadopoulou. Critical area computation for missing material defects in vlsi circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):583–597, 2001.

[28] Evanthia Papadopoulou and Maksym Zavershynskyi. On higher order voronoi diagrams of line segments. In *International Symposium on Algorithms and Computation*, pages 177–186. Springer, 2012.

[29] Sandeep Kumar Poonia. Lecture25, Apr 2014.

[30] Edgar A. Ramos. On range reporting, ray shooting and k-level construction, 1999.

[31] Michael Ian Shamos. Geometric complexity. In *Proceedings of the seventh annual ACM symposium on Theory of computing*, pages 224–233. ACM, 1975.

[32] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975.

[33] Chee K Yap. Ano (n logn) algorithm for the voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2(4):365–393, 1987.