Computing the Implicit Voronoi Diagram in Triple Precision

David L. Millman and Jack Snoeyink

Department of Computer Science, University of North Carolina - Chapel Hill,
Box 3175, Brooks Computer Science Building, Chapel Hill, NC, 27599-3175, USA
{dave,snoeyink}@cs.unc.edu
http://cs.unc.edu/~dave

Abstract. In a paper that considered arithmetic precision as a limited resource in the design and analysis of algorithms, Liotta, Preparata and Tamassia defined an "implicit Voronoi diagram" supporting logarithmic-time proximity queries using predicates of twice the precision of the input and query coordinates. They reported, however, that computing this diagram uses five times the input precision. We define a reduced-precision Voronoi diagram that similarly supports proximity queries, and describe a randomized incremental construction using only three times the input precision. The expected construction time is $O(n(\log n + \log \mu))$, where μ is the length of the longest Voronoi edge; we can construct the implicit Voronoi from the reduced-precision Voronoi in linear time.

Keywords: Voronoi diagram, Low-degree primitives, Randomized algorithm, Robust computation.

1 Introduction

Geometric algorithms that have been proved correct may still fail due to numerical errors that occur because geometric predicates and constructions require higher precision than is readily available. For example, computing the topological structure of the Voronoi diagram of n sites requires four times the input precision, Voronoi vertices of sites with integer coordinates have rational coordinates of triple precision over double, and testing whether a query point is above or below a segment joining two Voronoi vertices requires six times input precision. Liotta, Preparata, and Tamassia [1] derived a structure from the Voronoi diagram that supports logarithmic-time proximity queries for points on a grid using only two times the input precision. Unfortunately, they report that computing their diagram requires five times the input precision.

We introduce a structure that similarly supports proximity queries, but is computed incrementally using at most triple precision in $O(n(\log n + \log \mu))$ expected time, where μ is the length of the longest Voronoi edge. From our structure it is easy to obtain the structure of Liotta *et al.* in linear time.

Computing Voronoi diagrams is a well studied problem and many optimal algorithms have been proposed [2, 3, 4, 5]. Most are designed for a RealRAM or other computational model in which coordinate computations may be carried out to arbitrary precision, allowing the computer to work with exact Euclidean geometry.

F. Dehne et al. (Eds.): WADS 2009, LNCS 5664, pp. 495–506, 2009.

There are four popular ways to handle the numerical precision issues that arise when geometric algorithms are implemented on computers with limited precision: rounding, exact geometric computation, topological consistency, and degree-driven algorithm design. *Rounding* to machine precision is simplest, and results in fast execution, but calculations with incorrect values may cause algorithms to have unexpected behavior or even fail. The *exact geometric computation* paradigm encapsulates the numerical computations in geometric and combinatorial predicates and constructions that are guaranteed to produce correct decisions. These predicates can be built into libraries, such as CORE [6, 7], CGAL [8] and LEDA [9], for reuse by many algorithms. These libraries support various techniques for implementing correct predicates, including *arbitrary precision* in software, which is slow but always correct, *arithmetic filters* [10, 11, 12], which use precomputed error bounds for machine arithmetic so that arbitrary precision is needed only when machine precision is insufficient, and *adaptive predicates* [13], which evaluate only to the precision needed to guarantee a correct solution. Sugihara and Iri [14] suggest that *topological consistency* is more important than geometric correctness – that inaccurate values of coordinates can be tolerated provided that the data structures satisfy topological invariants needed by algorithms for correct operation. For example, a Voronoi diagram algorithm may be allowed to round vertex coordinates so that the embedding becomes non-planar, but as the graph itself is connected and planar, a graph-based traversal will at least terminate. Topological consistency produces correct results when the numerical computation gives correct predicate decisions, and at least avoids catastrophic failure when one or more predicate decisions are incorrect. *Degree-driven algorithm design* considers arithmetic precision as a limited resource that should be optimized along with running time and memory. Input is assumed to be single precision; often restricted to an integer grid for convenient analysis. Liotta, Preparata, and Tamassia [1] named this technique in their work on point location, in which they suggested polynomial degree to capture the complexity of predicates. The technique has also been applied to computing segment intersections [15, 16, 17]. Our work described here falls into degree-driven algorithm design.

2 Geometric Preliminaries and Related Work

The Voronoi diagram is well known in computational geometry, but because we will be concerned with the precision of input, we start with a restricted definition and remind the reader of some properties. Assume that we are given a set of n *sites*, $S = \{s_1, s_2, \ldots, s_n\}$, with each $s_i = (x_i, y_i)$ having single precision coordinates and all of them lying in a region of interest in the plane that can be described with a $O(1)$ coordinate values. (The easiest assumption is that S lay in a bounded rectangle in the integer grid.) The distance metric is Euclidean.

The Voronoi diagram, $\mathrm{VoD}(S)$, is the partition of our bounded rectangle into maximally connected regions with the same set of closest sites. The partition includes *Voronoi regions* closest to one site, *Voronoi edges* equidistant to two

closest sites, and *Voronoi vertices* equidistant to three or more closest sites. The Voronoi cell $\mathrm{VR}_S(s_i)$ is the closure of the region of s_i:

$$\mathrm{VR}_S(s_i) = \{x \in \mathbb{R}^2 \mid \|x - s_i\| \leq \|x - s_j\|,\ \forall s_j \in S\}.$$

Note that we will suppress the S and write $\mathrm{VR}(s_i)$ when S is clear from the context. Let b_{ij} denote the locus of points equidistant to s_i and s_j, the perpendicular bisector of the segment $\overline{s_i s_j}$. Note that as we are interested in a particular region, we can clip bisectors and Voronoi edges to finite segments.

Aurenhammer surveys [18] properties of the Voronoi diagram. In particular, we use the following:

- Bisectors are straight lines.
- A Voronoi edge on the boundary of cells $\mathrm{VR}(s_i)$ and $\mathrm{VR}(s_j)$ lies on the bisector of s_i and s_j.
- A Voronoi cell is the intersection of closed half planes; this implies that Voronoi cells are convex.
- A site s_i is contained in its cell, $s_i \in \mathrm{VR}(s_i)$.

The following properties of the Voronoi diagram are also known, but we state them in a form that will be helpful for our constructions later in the paper.

Lemma 1. *The order in which Voronoi cells intersect a line ℓ is the same as the order of the corresponding sites orthogonal projection onto ℓ.*

For our incremental construction we will need to decide if a new cell intersects a horizontal or a vertical segment. A corollary of Lemma 1 will give us a convenient way of making this decision.

Corollary 2. *Without loss of generality, consider a horizontal segment σ and the set of sites $\mathcal{S}$ whose Voronoi cells intersect σ. Let q be a new site, and let s_i, s_j be the sites of S whose projections onto σ form the smallest interval containing the projection of q; site s_i or s_j can be taken as infinite if no finite interval exists. To determine if $\mathrm{VR}_{\mathcal{S}}(q)$ intersects σ, it suffices to test if an endpoint of σ or the intersection of $\sigma \cap b_{ij}$ is closer to q than to both s_i and s_j.*

Proof. Assume that q is above σ and that we would like to determine if $\mathrm{VR}_{\mathcal{S}}(q)$ appears below σ. Let $x_i < x_j$, and $c_i, c_j \in \sigma$ be points in the cells of s_i and s_j respectively. Consider the point q' that has the same x coordinate as q, but is raised to infinity. Now, lower q' continuously, computing the Voronoi diagram of $\sigma \cup \{q'\}$ until the cell of q' intersects σ, and let $c_{q'}$ be this intersection point. Lemma 1 tells us that $c_i < c_{q'} < c_j$. In addition, $c_{q'}$ must be on b_{ij}, otherwise $c_{q'}$ would be in the middle of the cell of s_i or s_j causing the cell to be non-convex. In fact, $c_{q'}$ is the point equidistant to s_i, q' and s_j. Now, if q' is above q then the point equidistant to s_i, q and s_j must be below $c_{q'}$ so the cell of q must intersect σ. Alternatively, if q' is below q then the point equidistant to s_i, q and s_j is above $c_{q'}$ so the cell of q is completely above σ and therefore, their intersection is empty. ∎

Next we show that the intersection of a Voronoi diagram and a convex region is a collection of trees.

Lemma 3. *Given a set of sites S and a convex region R containing no sites of S; the edges and vertices of the* VoD(S) *in R form a forest.*

Proof. If $R \cap \text{VoD}(S)$ contained a cycle then R would contain a Voronoi cell, and therefore a site. Since R contains no sites, we conclude that $R \cap \text{VoD}(S)$ does not contain a cycle. ∎

The Voronoi diagram can be used to determine the closest site to a query point q if we build a point location structure on top of it. The trapezoid method of point location [2] builds a logarithmic-depth directed acyclic graph (DAG) with two types of nodes: x-nodes evaluate whether q is left or right of a vertical line through some vertex v by comparing x coordinates, and y-nodes evaluate whether q is above or below the line through some edge e.

As suggested in the introduction, Voronoi vertices are rational polynomials in the input coordinates of degree three over degree two. So to compare x coordinates with a single precision input, it would suffice to clear fractions and evaluate the sign of a degree three polynomial using triple precision computation. (Triple precision is also necessary; the polynomial for this predicate is irreducible of degree three.)

If edge e were defined by two arbitrary Voronoi vertices, then y-node test would require degree six, but since it is enough to test Voronoi edges, which lie on bisectors, we can compare squared distances to pairs using double precision. Liotta et al. [1] further observed that when the query points are on a grid, an x-node can store coordinates of v rounded to half grid points, which reduces the x-node evaluation to single precision. Thus, they defined their *implicit Voronoi diagram*, which stores the topology of VoD(S) as a point location DAG, and for each edge stores the pair of sites defining the bisector, and for each vertex the Voronoi vertex rounded to a half-integer grid. This has the anomaly that the stored vertices do not lie on the stored edges. Nevertheless, point location with a grid point q as input will report the containing cell correctly, and in logarithmic time.

Unfortunately, the only method that Liotta et al. [1] suggest to build their implied Voronoi diagram is to build the true Voronoi diagram and round, which they report is a degree five computation. We will reduce this to degree three.

Voronoi diagrams on a pixel grid have been considered in both graphics and image processing. In graphics, Hoff *et al.* [19] used the GPU to render an image of the Voronoi diagram and to recover an approximate Voronoi diagram from the screen buffer. This method generalizes easily to sites that are segments, curves, or areas. The work does not consider precision or accuracy guarantees but discusses errors created from multi-precision distance computations rounded to machine precision.

In image processing, the distance and nearest neighbor transforms are two operations that can be viewed as querying only at pixels for the distance or name of the nearest feature pixel. Breu *et al.* [20] developed a linear-time algorithm for these transforms by computing the Euclidean Voronoi diagram and querying. Here, linear is in the number of pixels in the grid; the number of sites may be proportional. They avoid extra logarithmic factors by using the locality of the grid in point location and in divide and conquer construction. Their algorithm assumes

a RealRAM and uses at least four times input precision; a divide and conquer version of our algorithm would reduce the precision of computing distance and neighbor transforms without sacrificing their worst-case running time.

3 Predicates and Constructions

The traditional measures in the theory of algorithms are asymptotic time and space, usually described up to a multiplicative constant by big-O notation. Liotta *et al.* [1, 15] suggest that we can analyze the arithmetic precision required by combinatorial and geometric algorithms, up to an additive constant, by expressing predicates and constructions as rational polynomial functions of the input variables and looking only at the polynomial degree.

We assume that input coordinates are single variables, which are degree-one polynomials. The degree of a monomial is the sum of the degrees of its variables, and the degree of a polynomial is the maximum degree of its monomials. The degree of a predicate is the maximum degree of its polynomials, and the degree of an algorithm is the maximum degree of its predicates.

Bisector Side Predicate: To clarify, we illustrate this concept with a *bisectorSide* predicate that determines whether a query point q is closer to site p or site r by comparing squared distances:

($a1$) Evaluate $(q_x - p_x)^2 + (q_y - p_y)^2 \lessgtr (q_x - r_x)^2 + (q_y - r_y)^2$.
($a2$) The result $<$ implies that p is closer, $>$ implies r is closer, and $=$ implies q is on the bisector of p and r.

Since this computation can be performed by evaluating the sign of a degree 2 polynomial, it suffices to use double precision plus a couple of bits for possible carries. In the rest of this section we briefly define three other predicates or constructions that operate on bisectors.

Stabbing Order Predicate: Given two bisectors b_{12}, and b_{34}, defined by input sites, and a vertical grid line ℓ that both bisectors intersect, the *stabbingOrder* predicate determines if the intersection $b_{12} \cap \ell$ is above, below or at the same point as the intersection $b_{34} \cap \ell$.

Lemma 4. *We can determine the stabbingOrder of two bisectors on a grid line using degree three computation in constant time.*

Proof omitted for extended abstract.

Bisector-Segment Intersection Predicate: Given bisector b_{12} and a non-vertical segment $\sigma \subset b_{34}$ with left and right endpoints on horizontal gridlines, ℓ_l and ℓ_r, the *bisectorSegmentIntersection* predicate determines if σ intersects b_{12}.

Lemma 5. *We can determine if a bisector intersects a segment whose end points lay on gridlines with degree three computation in constant time.*

Proof. The stabbingOrder of b_{12} and b_{34} on ℓ_l and ℓ_r is different if and only if we have found an intersection; two stabbingOrder tests suffice. ∎

Bisector Intersection Construction: Given a bisector b_{12} that intersects a non-vertical segment σ as defined for Lemma 5, the *bisectorIntersection* construction identifies the grid cell containing the intersection of b_{12} and σ.

Lemma 6. *We can identify the grid cell containing the intersection of a bisector and a segment whose end points lay on gridlines with degree three computation in time proportional to log of the length of the segment.*

Proof. We can do a binary search on the segment for the grid cell containing the intersection using the degree three stabbingOrder predicate. ∎

4 The Reduced-Precision Voronoi Diagram

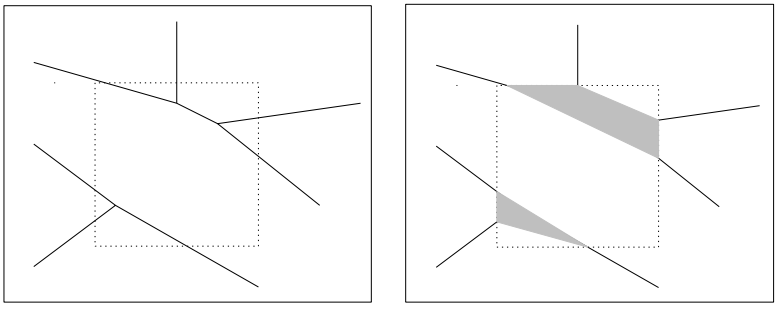
Given a set of n sites $S = \{s_1, s_2, \ldots, s_n\}$, whose coordinates are b-bit integers, we define a reduced-precision Voronoi diagram that is intermediate between the true Voronoi diagram $\mathrm{VoD}(S)$ and the implicit diagram of Liotta *et al.* [1]. Because we use predicates of at most degree three, we cannot know exactly how bisectors intersect inside of a grid cell. The predicates of the previous section, however, do provide full information at the grid cell borders. This low level of information inside of a grid cell gives us a "fuzzy" picture of Voronoi vertices that we contract to *rp-vertices*. Since we do however know precise information at the grid boundaries we maintain *rp-edges* that keep the same edge ordering as Voronoi edges entering the grid cell. In this way we keep enough control of the Voronoi vertices to perform constructions efficiently; in contrast, the implicit diagram rounds Voronoi vertices off their defining edges.

Let us consider the integer grid $\mathcal{G}$ as a partition into *grid cells* of the form $[i, i+1) \times [j, j+1)$ for integers i, j. The *rp-Voronoi* $\widehat{\mathrm{V}}(S)$ is the graph with *rp-vertices* and *rp-edges* defined by contracting every edge of the Voronoi diagram $\mathrm{VoD}(S)$ that lies entirely inside some grid cell.

Figure 1(a) depicts and example grid cell $G \in \mathcal{G}$ and shows the intersection $G \cap \mathrm{VoD}(S)$, which by Lemma 3 is a forest. In the graph structure of the rp-Voronoi, $\widehat{\mathrm{V}}(S)$, we therefore contract each tree of the forest to an rp-vertex. Edges that leave the grid cell are preserved as rp-edges. Notice that the planar embedding of the Voronoi $\mathrm{VoD}(S)$ gives a natural planar embedding of $\widehat{\mathrm{V}}(S)$ in which the ordering of edges entering a grid cell is preserved as the ordering of edges around the rp-vertex. We find it useful to depict these rp-vertices as the convex hulls of the intersections of rp-edges, as in Figure 1(b). Although we never actually compute these convex hulls they bound the locations where the tree of $\mathrm{VoD}(S)$ can lie.

Each rp-vertex v maintains the grid cell G_v containing v, and an list of its incident rp-edges in counter-clockwise order by their entry to G_v. Each rp-edge e stores the generator sites s_1 and s_2 of the corresponding Voronoi diagram edge, and pointers to its location in the lists of its two rp-vertices. Standard data structures, like the doubly-connected edge list [2], allow us to maintain the order in the planar subdivision represented by $\widehat{\mathrm{V}}(S)$.

Finally we define the *boundary of the rp-region* of s to be the alternating sequence of rp-edges storing site s and the connecting rp-vertices that form a



(a). Voronoi diagram **(b).** rp-Voronoi diagram

Fig. 1. The Voronoi vertices in a grid cell on the left contract to two rp-vertices, depicted as gray convex polygons on the right. The ordering of the edges entering the grid cell is maintained in both diagrams.

cycle. We call the *rp-region* all the points enclosed in this cycle, and the *rp-cell* the union of the rp-region with its boundary.

Observation 7. *The number of rp-vertices and rp-edges in the rp-Voronoi diagram of S is less than or equal to the number of Voronoi vertices and Voronoi edges in the Voronoi diagram of S respectively.*

As we will show in Section 5.3, we can retrieve the implicit Voronoi diagram once we have constructed the rp-Voronoi.

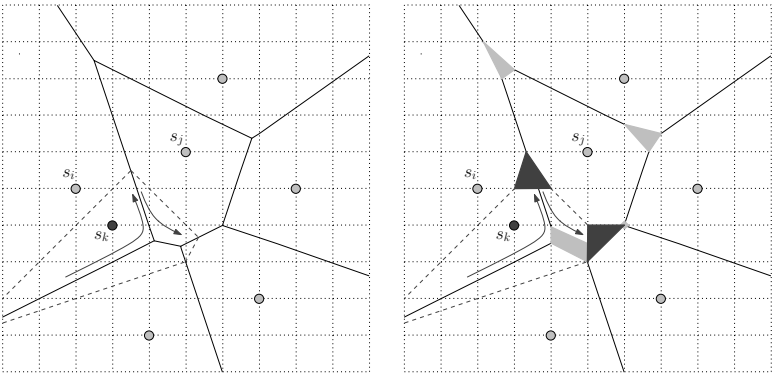
5 Constructing the Reduced-Precision Voronoi Diagram

Next we describe how to construct the rp-Voronoi, analyze the expected time and space, describe how to use the rp-Voronoi for point location and show how to convert the rp-Voronoi to the implicit Voronoi diagram.

We create the rp-Voronoi by a randomized incremental construction [2] that parallels Sugihara and Iri's method [14]: inserting a new site by "carving" out the new cell from the previous diagram. Inserting a new site invalidates a subgraph of the Voronoi diagram, referred to as the *conflict region*. Sugihara and Iri made the observation that the conflict region is a tree, and that by walking the tree we identify the invalid sub-graph.

Specifically, their method constructs a Voronoi diagram of the first $k-1$ sites and then inserts site s_k. To start carving, the site s_i closest to s_k is identified and the bisector b_{ik} is traced until it enters the neighboring Voronoi cell, $\mathrm{VR}(s_j)$. The bisector b_{jk} is then traced, and the process continues until it returns back to $\mathrm{VR}(s_i)$. The tracing process requires the identification of the next bisector intersection with a Voronoi edge. Sugihara and Iri do this by walking around the cell of s_j on the side of the new site s_k until the next intersection is found (see Figure 2a).

Our method does the same computation, but since we restrict ourselves to degree three, it is too costly to compute and compare coordinates of bisector interesections.



(a). Voronoi diagram **(b).** rp-Voronoi diagram

Fig. 2. The cell for the new dark gray site s_k is "carved" out of the diagram of light gray sites. The traced bisectors are emphasized with dotted lines, and the tree walk is shown with gray arrows.

5.1 Incremental Construction

We initialize the rp-Voronoi diagram with two sites s_1 and s_2, and use their bisector b_{12} to split the initial region of interest (the grid) by using the binary search of bisectorIntersection.

Now, assume that we have already constructed the rp-Voronoi of $k-1$ sites S_{k-1} and that we would like to insert site s_k. The *rp-Voronoi Update Procedure* takes as input the rp-Voronoi of S_{k-1} and a new input site s_k and returns the rp-Voronoi of $S_{k-1} \cup \{s_k\}$.

rp-Voronoi Update Procedure: We sketch the procedure in this paragraph and then fill in the details in the remainder of the section. We first locate the site $s_i \in S_{k-1}$ closest to s_k, and proceed in two steps. We find the subgraph T that consists of the set of rp-vertices and rp-edges that are no longer part of the rp-Voronoi of $S_{k-1} \cup \{s_k\}$. In the Voronoi diagram, the conflict region is a tree and the sum of all conflict region sizes is linear in expectation. In the rp-Voronoi we walk a subset T of the edges of this tree, and their vertices; once we have identified this subset, we maintain our data structure in time proportional to its size, which is therefore also linear (see Figure 2b).

To identify T we start by tracing out the s_i, s_k bisector b_{ik}. We walk around the boundary of the region of s_i until we find the grid cell G containing the intersection of b_{ik} and the boundary of the rp-region of s_i. As in Sugihara and Iri's algorithm, we would like to pick the next bisector to trace, thus, allowing us to continue our tree walk. To *pick the next bisector* for the rp-Voronoi with limited precision there are two cases: one simple and the other interesting.

In the simple case, b_{ik} intersects the rp-edge e that stores sites s_i and s_j. This intersection is determined by applying the bisectorIntersection construction. We switch to the s_k, s_j bisector and continue building T by walking around the boundary of the region of s_j on the side of s_k.

In the more interesting case, b_{ik} intersects an rp-vertex v. This intersection is determined by first checking if b_{ik} passes through the grid cell containing v. If it does, we compare the stabbingOrder of b_{ik} and the two rp-edges incident on v of the rp-cell of s_k to determine if b_{ik} intersects v.

Let G_N, G_E, G_S and G_W be the north, east, south and west grid walls, respectively, of G, and without loss of generality, assume that we have entered G from the south, with s_i below the b_{ik} bisector (see Figure 3).

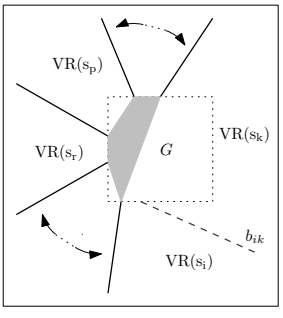


Fig. 3. We enter grid cell G from the south while walking the tree of the new site s_k along the s_i, s_k bisector b_{ik} in dashed gray. The projections of sites s_p and s_r onto the west grid line are directly above and below the projection of s_k onto the west grid line respectively.

Since Voronoi cells are convex, the new cell of s_k can intersect each of the grid cell boundary walls at most twice. This gives us four cases for how the traced bisectors of the new Voronoi cell enter and exit a grid wall G_x of G (see Figure 4). Traced bisectors of the new Voronoi cell,

($c1$) do not exit through G_x.
($c2$) exit through G_x and do not return to G.
($c3$) exit through G_x and return through G_x.
($c4$) exit through G_x and return through a different grid wall G_y.

First, we determine if the Voronoi cell $\mathrm{VR}(s_i)$ pokes out of the G_W grid wall. We find the two sites s_p and s_r whose Voronoi cells intersect G_W and whose y coordinate is directly above and below the y coordinate of s_k, respectively. We then determine if $\mathrm{VR}(s_i)$ pokes out by applying Corollary 2.

If $\mathrm{VR}(s_k)$ does not poke out of G_W (case 1) we repeat the process with G_N, followed by G_E. If $\mathrm{VR}(s_k)$ does poke out then there are some points in the $\mathrm{VR}(s_r)$ that are now in the $\mathrm{VR}(s_k)$. Since no site has an empty cell there must be a Voronoi edge $e \in \mathrm{VR}(s_k)$ that is a subset of the s_r, s_k bisector. We trace b_{rk}, following the tree, towards b_{ik} until we return back to G. Now, we have identified the next bisector to trace for our tree walk. In addition, we just walked backwards through a subtree of T, and we continue the procedure by tracing b_{rk} in the opposite direction as before.

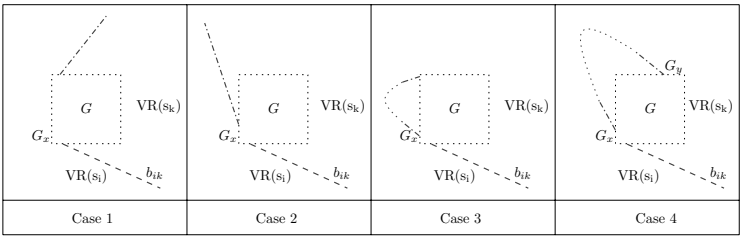


Fig. 4. Grid cell G with grid walls G_x and G_y as west and north walls respectively. A bisector enters G through the south wall by tracing the s_i, s_k bisector b_{ik}, in dashed gray. In alternating dashed and dotted gray are the four cases per wall for bisector tracing.

The other three cases are determined by continuing the walk. Case 3 corresponds to the walk returning back to the grid cell through the same cell wall it exited. Case 4 occurs when the walk returns back to the grid cell, but through a different grid wall. This allows us to determine cases 3 and 4 that can cause multiple rp-vertices to occur in one grid cell.

We continue this process until we have completed the cycle, identified T and the new rp-vertices and rp-edges. We update the rp-Voronoi of S_{k-1} to get the rp-Voronoi of $S_{k-1} \cup \{s_k\}$.

Note that if a Voronoi vertex v is outside our region of interest we do not need to identify the grid cell containing v since it will not be used for proximity queries. However, to continue with our tree walk we can apply Corollary 2 similar to the case where a bisector intersects an rp-vertex.

5.2 Analysis

Point location is accomplished by the standard method of maintaining the construction history [4] allowing for point location in expected $O(\log n)$ time. To achieve a degree two algorithm we use grid cells in x-nodes and bisectorSide for y-nodes, much like the structure in [1]. The incremental construction described above relies on bisectorSide and bisectorIntersection operations, which are of degree two and three respectively, as shown in Section 3.

As explained by Observation 7, the rp-Voronoi and Voronoi diagram have the same combinatorial complexity. The update procedure creates at most as many rp-vertices as Voronoi vertices. As shown by [2, 4] the number of Voronoi vertices created is expected linear throughout the algorithm. Furthermore, the tree walk touches only edges that are modified, and the number of modified edges is constant in expectation [4].

However, we must pay two additional charges in each update. First there is an extra $O(\log \mu)$ charge for finding bisector segment intersections. Secondly, we must pay an $O(\log n)$ to find the upper and lower neighbors when a bisector intersects an rp-vertex. So we have shown the expected time to insert a new site into the reduced-precision Voronoi diagram of size n is $O(\log n + \log \mu)$ where μ

is the length of the longest Voronoi edge, and that this insertion can be done with degree three predicates. We conclude:

Theorem 8. *We can construct a reduced-precision Voronoi diagram of n sites with a degree three algorithm in expected $O(n(\log n + \log \mu))$ time where μ is the length of the longest Voronoi edge.*

5.3 Reduced-Precision Voronoi to Implicit Voronoi

Next we describe how to convert the reduced-precision Voronoi to the implicit Voronoi of [1], described in Section 4. An rp-vertex corresponds to a tree T, of Voronoi vertices and Voronoi edges. Some of the Voronoi vertices of T may be on grid lines and the implicit Voronoi would assign these vertices integer coordinates. To create the implicit Voronoi we must separate these vertices from an rp-vertex.

Theorem 9. *We can convert the reduced-precision Voronoi diagram to the implicit Voronoi diagram in $O(n)$ time with degree three computations.*

Proof omitted for extended abstract.

6 Conclusion and Open problems

To our knowledge, this is the first construction of a proximity query structure in less than degree four and sub-quadratic time, that allows for logarithmic query times. In addition, we believe that this is the first construction of the implicit Voronoi diagram without fully computing the Voronoi diagram. Is there a reasonable algorithm for creating a proximity query structure with only degree two predicates? We believe that there is, but perhaps at the cost of an additional logarithmic factor in space and time.

There is more to discover with respect to restricted predicates for computing Voronoi diagrams. It is easy to generalize the ideas presented in this paper to the power diagram, but further investigation is necessary to understand how these methods effect the complexity of the power diagram, as well as other generalized Voronoi diagrams with linear bisectors. The basis for the method of bisector intersection relies on linearity. What if the diagram's bisectors are more complicated, such as hyperbolic arcs, as in the Voronoi diagram of disks?

One final observation is the algebraic complexity of standard predicates, such as orientation and inSphere, increases with dimension. Distance based predicates, on the other hand, maintain the same algebraic complexity regardless of the dimension. Our last question is can we compute some form of a Voronoi diagram with reduced precision in any dimension?

References

[1] Liotta, G., Preparata, F.P., Tamassia, R.: Robust proximity queries: An illustration of degree-driven algorithm design. SIAM J. Comput. 28(3), 864–889 (1999)

[2] de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, New York (2008)

[3] Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica 2, 153–174 (1987)

[4] Guibas, L.J., Knuth, D.E., Sharir, M.: Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica 7, 381–413 (1992)

[5] Shamos, M.I., Hoey, D.: Closest-point problems. In: SFCS 1975: Proceedings of the 16th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 151–162. IEEE Computer Society, Los Alamitos (1975)

[6] Karamcheti, V., Li, C., Pechtchanski, I., Yap, C.: A core library for robust numeric and geometric computation. In: SCG 1999: Proceedings of the fifteenth annual symposium on Computational geometry, pp. 351–359 (1999)

[7] Yap, C.K.: Towards exact geometric computation. Comput. Geom. Theory Appl. 7(1-2), 3–23 (1997)

[8] CGAL, Computational Geometry Algorithms Library, `http://www.cgal.org`

[9] Burnikel, C., Könemann, J., Mehlhorn, K., Näher, S., Schirra, S., Uhrig, C.: Exact geometric computation in LEDA. In: SCG 1995: Proceedings of the eleventh annual symposium on Computational geometry, pp. 418–419 (1995)

[10] Devillers, O., Preparata, F.P.: A probabilistic analysis of the power of arithmetic filters. Discrete and Computational Geometry 20(4), 523–547 (1998)

[11] Devillers, O., Preparata, F.P.: Further results on arithmetic filters for geometric predicates. Comput. Geom. Theory Appl. 13(2), 141–148 (1999)

[12] Fortune, S., Van Wyk, C.J.: Efficient exact arithmetic for computational geometry. In: SCG 1995: Proceedings of the eleventh annual symposium on Computational geometry, pp. 163–172 (1993)

[13] Shewchuk, J.R.: Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. Discrete & Computational Geometry 18(3), 305–363 (1997)

[14] Sugihara, K., Iri, M.: Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic. Proceedings of the IEEE 80(9), 1471–1484 (1992)

[15] Boissonnat, J.D., Preparata, F.P.: Robust plane sweep for intersecting segments. SIAM J. Comput. 29(5), 1401–1421 (2000)

[16] Boissonnat, J.D., Snoeyink, J.: Efficient algorithms for line and curve segment intersection using restricted predicates. In: SCG 1999: Proceedings of the fifteenth annual symposium on Computational geometry, pp. 370–379 (1999)

[17] Chan, T.M.: Reporting curve segment intersections using restricted predicates. Comput. Geom. Theory Appl. 16(4), 245–256 (2000)

[18] Aurenhammer, F.: Voronoi diagrams a survey of a fundamental geometric data structure. ACM Comput. Surv. 23(3), 345–405 (1991)

[19] Hoff, K.E., Keyser, J., Lin, M., Manocha, D., Culver, T.: Fast computation of generalized voronoi diagrams using graphics hardware. In: SIGGRAPH 1999: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 277–286 (1999)

[20] Breu, H., Gil, J., Kirkpatrick, D., Werman, M.: Linear time Euclidean distance transform algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence 17, 529–533 (1995)