A Sweepline Algorithm for Voronoi Diagrams

Steven Fortune
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

We present a transformation that can be used to compute Voronoi diagrams with a sweepline technique. The transformation is used to obtain simple algorithms for computing the Voronoi diagram of point sites, of line segment sites, and of weighted point sites. All algorithms have $O(n \log n)$ worst case running time and use $O(n)$ space.

1. Introduction

We present simple algorithms for the construction of Voronoi diagrams of a set of sites in the plane. The sites can be points or line segments. The algorithms are based on the sweepline technique [S83]. The sweepline technique conceptually sweeps a horizontal line upwards across the plane, noting the regions intersected by the line as the line moves. Computing the Voronoi diagram directly with a sweepline technique is difficult, because the Voronoi region of a site may be intersected by the sweepline long before the site itself is intersected by the sweepline. Rather than compute the Voronoi diagram, we compute a geometric transformation of it. The transformed Voronoi diagram has the property that the lowest point of the transformed Voronoi region of a site appears at the site itself. Thus the sweepline algorithm need consider the Voronoi region of a site only when the site has been intersected by the sweepline. It turns out to be easy to reconstruct the real Voronoi diagram from its transformation; in fact in practice the real Voronoi diagram would be constructed, and the transformation computed only as necessary. The sweepline algorithms all have asymptotic running time $O(n \log n)$ and space usage $O(n)$.

Previous algorithms for Voronoi diagrams fall into two categories. First are incremental algorithms, which construct the Voronoi diagram by adding a site at a time. These algorithms are relatively simple, but have worst case time complexity $O(n^2)$. However, the algorithms may have good expected time behavior [BWY80] [GS77] [OIM84].

The second category of algorithms are divide-and-conquer algorithms. The set of sites is split into two parts, the Voronoi diagram of each part computed recursively, and then the two Voronoi diagrams merged together. If the sites are points, then they can be split simply by drawing a line that separates the sites into two halves[SH75]. If the sites are line segments, then more complex partitioning is necessary [Y84]. With care, the divide-and-conquer algorithms can be implemented in worst case time $O(n \log n)$. The difficulty of the divide-and-conquer algorithms is generally with the merge step that combines two Voronoi diagrams together. While the time required for this step is only linear in the number of sites, the details of the merge are complex and hard to implement.

The sweepline algorithms presented in this paper are competitive in simplicity with the incremental algorithms. Since they avoid the merge step, they are much simpler to implement than the divide-and-conquer algorithms. But they have the same worst-case time complexity as the divide-and-conquer algorithms.

We also present an algorithm to compute the Voronoi diagram of weighted point sites, in which each site has an additive weight associated with it. The algorithm uses exactly the same sweepline technique, and has time complexity $O(n \log n)$. No algorithm with this time complexity was previously known for this problem.

All algorithms presented in this paper are assumed to be implemented in exact real arithmetic. We hope to investigate stability issues of the algorithms in more reasonable models of floating point arithmetic.

2. Point Sites

We first consider the case that all sites are points in the plane. This simple case has been discussed extensively in the literature. For more details, see for example [SH75] or [LD81]. For a more general situation than point sites, see [LS85a].

If $p \in R^2$, then p_x and p_y are the x and y coordinates of p, respectively. Points $p,q \in R^2$ are *lexicographically ordered*, $p<q$, if $p_y<q_y$ or $p_y=q_y$ and $p_x<q_x$.

Let S be a set of points in the plane, called *sites*. For $p \in S$, $d_p: R^2 \rightarrow R$ is the (Euclidean) distance from a point in R^2 to p, and $d: R^2 \rightarrow R$ is $\min\limits_{p \in S} d_p$. The *Voronoi circle at* $z \in R^2$ is the circle centered at z of radius $d(z)$. The *bisector* B_{pq} of p, $q \in S$ is $\{z \in R^2 : d_p(z)=d_q(z)\}$. B_{pq} is of course a line, the usual bisector of p and q. R_{pq} is $\{z \in R^2 : d_p(z) \leqslant d_q(z)\}$, and the *Voronoi region of p*, R_p, is $\bigcap\limits_{q \neq p} R_{pq}$. R_p is convex, possibly unbounded. The *Voronoi diagram* $V(S)$, or V for short, is $\{z \in R^2$: there is $p \neq q$ with $d(z)=d_p(z)=d_q(z)\}$. See Figure 2.1.

V consists of the union of segments and halflines. A *vertex* of V is a point of V with at least three incident segments or halflines; equivalently a vertex is a point equidistant from at least three sites. An *edge* of V is a maximal straightline subset of V containing at least a segment and not containing any vertices in its interior. An edge either has two vertices as endpoints or is a halfline with one vertex as endpoint. $B_{pq} \cap V$ is always connected; if it properly contains a point, then it is an edge, labelled e_{pq}. e_{pq} forms part of the boundary of R_p and R_q. There are at most $O(\#S)$ edges and vertices of V, since V forms a planar graph.

2.1. The transformation *

Consider the mapping $*: R^2 \rightarrow R^2$ defined by $*(x,y) = (x, y+d((x,y)))$. $*$ maps the point $z \in R^2$ to the topmost point of the Voronoi circle at z. Note that $*$ is continuous, fixes all sites, and maps each vertical line into itself. In general we refer to $*(B)$ as B^*, for B a subset of the plane.

What is the effect of $*$ on a bisector? We claim that the image under $*$ of the Voronoi edge between sites at different heights is a section of a hyperbola, and between sites at the same height is a section of a vertical line. To see this, let $*_p(x,y) = (x, y+d_p((x,y)))$. Then $*_p(B_{pq}) = *_q(B_{pq}) = \{(x, y+d_p((x,y))) : (x,y) \in B_{pq}\}$. If $p_y > q_y$, then $*_p(B_{pq})$ is a hyperbola open upwards, with minimum point p. Note that $*_p(R_{pq})$ lies above $*_p(B_{pq})$, and $*_p(R_{qp})$ lies below $*_p(B_{pq})$. Otherwise we have $p_y = q_y$, so B_{pq} is the vertical line through $r = (p_x + q_x)/2$, and $*_p(B_{pq})$ is $\{(r, y+d_p((r,y)))\}$, which is the vertical halfline above (r, p_y) not containing (r, p_y). Now if e_{pq} is an edge of V, then $d = d_p = d_q$ on e_{pq}, so $e^*_{pq} = *_p(e_{pq})$, and e^*_{pq} is a section of a hyperbola or a vertical line. See Figure 2.3.

What is the effect of $*$ on a region R_p? Note that $* = *_p$ on R_p. Consider a vertical line not passing through p. It is easy to see that $*$ is monotonically increasing on the section of the vertical line through R_p, and maps every point to another above p. On the vertical line through p, $*$ maps every point in R_p below p to p, and is monotonically increasing in R_p above p. Notice that p must be the lowest point of R^*_p.

Now $*$ must be 1-1 on V, since no vertical segment incident to a site can be contained in V, and $*$ fails to be 1-1 only on such segments. Hence we can decompose V^* into edges and vertices, the images of edges and vertices of V, and V^* has the same incidence structure as V.

Lemma 2.1: If v is a vertex of V and v^* is not a site, then exactly one edge of V^* extends upwards from v^*, and the rest extend downwards. If v^* is a site, then exactly two edges extend upwards from v^* and the rest extend downwards.

Proof: Staightforward. □

Using the preceding discussion and Lemma 2.1, we can analyze every point x in the range of $*$. The following cases are mutually exclusive and exhaustive: x is in the interior of some region R^*_p; x lies on some edge e^*_{pq} but is neither a site nor a vertex; x is a site, is not a vertex, and is the minimum point of both R^*_x and e^*_{xp}, some p; x is a vertex, not a site, with at least two edges incident downwards and exactly one incident upwards; and x is a vertex and a site, with exactly two edges incident upwards and at least one incident downwards, and is the minimum point of R^*_x.

2.2. The algorithm for V^* and for V

A sweepline algorithm suffices to compute V^*. The algorithm conceptually moves a horizontal line upward across the plane, maintaining the regions of V^* intersected by the horizontal line. A region is encountered for the first time at a site, and a region disappears at the intersection of two edges. The coordinates of these two events are easily computed, since all sites are known initially, and since intersections of edges can be computed as the regions bounded by the edges are encountered.

The algorithm is given formally as Algorithm 1. Algorithm 1 produces the Voronoi diagram V^* as a list of bisectors. Each bisector is marked with the vertices that are the endpoints of the corresponding Voronoi edge. If a bisector is marked with only a single vertex, then the corresponding edge is a halfline.

Algorithm 1 is written as if degeneracies could not happen, where a degeneracy is a site lying on a bisector or four or more cocircular points. As Theorem 2.2 shows, this simplification is still adequate to compute the Voronoi diagram in the presence of degeneracies. However, one consequence of this approach is that Algorithm 1 can produce zero length edges, specifically bisectors with the same point as the two endpoints. These edges must be removed from the list if their presence causes problems. Alternatively, Algorithm 1 can be written to test explicitly for degeneracies, though it then becomes more complex. Also, Algorithm 1 assumes that the set S of sites contains a unique bottommost site. This assumption is not essential; lines 2 and 3 need to be modified to initialize the list L correctly if S contains several sites with identical minimal y coordinate.

If $p_y > q_y$, then $*_p(B_{pq})$ is a hyperbola open upwards, and a horizontal line can intersect it at two points. To simplify the presentation of Algorithm 1, we split $*_p(B_{pq})$ into two pieces, C^+_{pq} to the left of and containing p, and C^-_{pq} to the right of and containing p. Then C^-_{pq} is monotonically decreasing, C^+_{pq} is monotonically increasing, and a horizontal line can intersect either of them at most once. If $p_y = q_y$ then we set $C^-_{pq} = \emptyset$ and $C^+_{pq} = *_p(B_{pq})$. We call C^-_{pq} and C^+_{pq} *boundaries*. We use C_{pq} to denote one of C^-_{pq} and C^+_{pq} when the choice is unimportant or can be determined from context. For example, in line 15 of Algorithm 1, C_{qs} is C^+_{qs} either if p is to the right of the higher of q and s or if q and s are cohorizontal; otherwise C_{qs} is C^-_{qs}.

Line 8 of Algorithm 1 is a search to find the region containing a newly encountered site. The search can be implemented as a binary search on list L, since L contains the regions and boundaries in order on the horizontal line. If the site actually falls on a boundary, the search can return the region on either side of the boundary. Note that the actual x coordinate where a boundary intersects the horizontal line is determined by the y coordinate of the line.

Theorem 2.2: Let S be a set of point sites, with unique bottommost site. Then Algorithm 1 computes $V^*(S)$.
Proof: Omitted due to lack of space. □

Theorem 2.3: Algorithm 1 can be implemented to run in time $O(n \log n)$ and space $O(n)$.
Proof: We first claim that the number of iterations of the **while** loop is at most $O(n)$; it then follows that the number of bisectors ever created is $O(n)$. The number of iterations of the loop for a point p not a site is one less than the number of boundaries intersecting at p, and for p a site is one more than the number of boundaries p lies on. Since the number of intersecting boundaries at p is bounded above by the degree of p as a vertex of V, the number of iterations summed over all vertices and sites is $O(n)$.

Now Q contains at most two entries per boundary and one per site, hence at most $O(n)$ entries. Q needs operations insert and extract-min; Q can be implemented as a heap at time cost $O(\log n)$ per operation and storage cost $O(n)$. Similarly, L can contain at most $O(n)$ entries, since a horizontal line can intersect each bisector at most twice. L needs operations insert, delete, and search (for Line 10); a balanced tree scheme can implement these at time cost $O(\log n)$ per operation and storage cost $O(n)$. Thus each iteration of the **while** loop takes time $O(\log n)$ for a total time of $O(n \log n)$. □

Theorem 2.4: Algorithm 1 can be modified to compute V in time $O(n \log n)$ and space $O(n)$.
Proof: We claim that Algorithm 1 can be modified to use only untransformed bisectors and vertices. Algorithm 1 creates a Voronoi edge by first creating the bisector containing the edge and then later marking the bisector with the endpoints of the edge; these operations can be performed directly. Event queue Q contains sites and the intersections of boundaries. The intersection of two boundaries can be obtained by computing the intersection of two untransformed bisectors and then adding to the y-coordinate of the intersection the distance to any of the sites determining the bisectors. Similarly, the list L can contain untransformed regions and bisectors; the transformation can be computed explicitly during the search of line 10. □

Algorithm 1 can be easily modified to compute the Delaunay triangulation. In the case of the intersection of boundaries, starting at line 12, the Delaunay triangle corresponding to vertices q, r, and s should be output. This will actually produce a full triangulation, with diagonals added arbitrarily to Delaunay regions with more than three sides.

3. Line segment sites

Algorithm 1 can be extended to compute the Voronoi diagram of a set of line segments. The general idea of the algorithm is the same; we transform the Voronoi diagram so that the lowest point of a Voronoi region appears at the Voronoi site, and then use a sweepline technique. However, the details are more complex, because the Voronoi diagram itself is more complex, and because the transformation * is not as well behaved.

The Voronoi diagram is constructed using the bisectors between pairs of sites, where the "bisector" is the locus of points equidistant from the two sites. The bisector of two points is a line. While the bisector of two disjoint segments is still a simple curve, it can have up to seven sections, each a section of a line or a parabola. Worse, the bisector of two segments sharing a common endpoint need not even be one-dimensional, since there is a two-dimensional region for which the common endpoint is the closest point to both sites.

We can simplify the situation somewhat by giving a slightly modified definition of the Voronoi diagram. We require that every segment be split into three sites, two for the endpoints and one for the segment itself. Then we distinguish between the endpoints of the segment or the segment itself being closest. As will be seen, this has the consequence that the bisector between two sites is always a section of a line or a parabola. Also, the two-dimensional region that used to be the bisector of two coincident segments becomes the Voronoi region of the common endpoint of the segments. A similar idea has been used in previous papers on the Voronoi diagram of line segments [K79] [Y84].

The transformation * in the line segment case is not always one-one, as it was in the point site case. This does not alter the asymptotic time or space complexity of the algorithm, but more care is needed in handling special cases.

3.1. The Voronoi diagram of line segments

Let S, the set of *sites*, be a set of points and line segments in the plane, with the properties that line segments intersect only at endpoints and that every endpoint of a line segment is a point site. For $p \in S$, $d_p : R^2 \rightarrow R$ is the *distance* to p and is defined by $d_p(z) = \operatorname{glb}_{x \in p} e(x,z)$, where $e(x,z)$ is the Euclidean distance from x to z. The *tangent contact region* of p, t_p, is R^2 for p a point, and for p a segment it is the band containing p bounded by the two lines through the endpoints of p perpendicular to p. (A circle centered at $z \in t_p$ of radius $d_p(z)$ is tangent to p.) $d : R^2 \rightarrow R$ is defined by $d(z) = \min_{p : z \in t_p} d_p(z)$. The *bisector* B_{pq} is $\{z \in t_p \cap t_q : d_p(z) = d_q(z)\}$. R_{pq} is $\{z \in t_p : d_p(z) \leqslant d_q(z)\}$ and the *Voronoi region* R_p is $\bigcap_{q \neq p} R_{pq}$. The *Voronoi diagram* $V = V(S)$ is $\{z \in B_{pq} : p, q \in S$ and $d(z) = d_p(z)\}$.

The bisector B_{pq} falls into one of four categories, depending on whether p and q are two points, a segment and its endpoint, a segment and a disjoint point, or two segments. If p and q are distinct point sites, then B_{pq} is the line bisecting p and q, as before. If p is an endpoint of segment q, then B_{pq} is the line perpendicular to q through p. If p is a point not collinear with segment q, then B_{pq} is a section of a parabola. (Note that if p is a point collinear with segment q, not an endpoint, then B_{pq} is empty.) Finally, if p and q are segment sites, then B_{pq} is a point, a segment, possibly empty if $t_p \cap t_q$ contains no points equidistant from p and q, or if p and q are collinear sharing a common endpoint, then B_{pq} is the line perpendicular to p and q through the common endpoint. See Figure 3.1.

The Voronoi region R_p need not be convex, however it is *starshaped from p*: for each $x \in R_p$, there is $y \in p$ so that the line segment xy is entirely contained in R_p (in fact, y can be chosen to be the point of p closest to x).

It is possible that the Voronoi region of a point site p is degenerate. If p is the common endpoint of two collinear segments, then R_p is contained in the line through p perpendicular to the two segments. If p is the common endpoint of three or more segments and every angle formed by two consecutive segments is less than π, then R_p is p itself.

A *vertex* of V is a point of V incident to three curves contained in V. A vertex is always a point equidistant from three sites and in all three tangent contact regions. (Note that the converse statement fails on the degenerate case mentioned first in the last paragraph.) An *edge* of V is a maximal one-dimensional curve connecting two distinct vertices, or with one vertex as endpoint and extending to infinity. An edge is always contained in a single bisector B_{pq}. $B_{pq} \cap V$ may not be connected, however it can have at most $\#S$ connected components, each an edge. There are at most $O(\#S)$ edges and vertices of V, since V forms a planar graph. Figure 3.2 is a simple example of the Voronoi diagram.

3.2. The transformation *

As before, we define $*: R^2 \rightarrow R^2$ by $*(x,y) = (x, y + d(x,y))$, and $*_p: R^2 \rightarrow R^2$ by $*_p(x,y) = (x, y + d_p(x,y))$.

What is the effect of $*_p$ on a bisector B_{pq}? Note that $*_p$ is 1-1 everywhere except on a vertical segment below the lowest point of p, or below all of p if p is a horizontal segment. The first case is that p and q are distinct point sites; then as before $*_p(B_{pq})$ is a hyperbola if $p_y \neq q_y$, and is an open vertical halfline if $p_y = q_y$. Second, if point site p is an endpoint of segment q, then if q is not horizontal, then $*_p(B_{pq})$ is two halflines with endpoint p, one directed up and to the left, the other directed up and to the right; if q is horizontal, then the vertical halfline below p contained in B_{pq} collapses to p, and $*_p(B_{pq})$ is the vertical halfline with bottom endpoint p. The third case is that p is a point site not collinear with segment q. Then $*_p(B_{pq})$ is a parabola, except if q is horizontal and $q_y > p_y$, in which case $*_p(B_{pq})$ is q.

Finally, if p and q are both segment sites, and B_{pq} is a segment, then in general $*_p(B_{pq})$ is also a segment. However, if p and q are both horizontal segments sharing a common endpoint, then again the vertical halfline below the common endpoint contained in B_{pq} collapses to the common endpoint, and B^*_{pq} is the vertical halfline above the common endpoint. See Figure 3.4.

As before, $d = d_p = d_q$ along $B_{pq} \cap V$, so $*_p = *_q = *$ along $B_{pq} \cap V$. Hence the edges of V^* are sections of lines, parabolas, and hyperbolas. Also, we again split $*_p(B_{pq})$ into two boundaries. If u and v are the lexicographically minimal points of p and q, respectively, and $u_y > v_y$, then $C^-_{pq} = \{(x,y) \in *_p(B_{pq}): x \leqslant u_x\}$ and $C^+_{pq} = \{(x,y) \in *_p(B_{pq}): x \geqslant u_x\}$. If $p_y = q_y$, then $C^+_{pq} = *_p(B_{pq})$ and $C^-_{pq} = \emptyset$. Thus C^-_{pq} if nonempty is monotonically decreasing, and C^+_{pq} is monotonically increasing.

Now $*$, like $*_p$, can fail to be one-one; this happens only on a vertical segment below a point z on which the value of d is identically the distance from z. Such a point z arises in three ways: if z is an isolated point site; if z is the lowest endpoint of a line segment site, and z is not the upper endpoint of any other nonhorizontal segment site; or if z is an interior point of a horizontal segment site.

Unfortunately, $*$ can fail to be 1-1 on V. As already noted, $*$ can collapse a bisector contained in the vertical halfline below the endpoint of a horizontal segment site into the endpoint of the segment. $*$ can also fail to be one-one in a different way: if point site z is the lower endpoint of a segment site, if no segment has z as its upper endpoint, and if there is some site p with y coordinate strictly below the y coordinate of z, then the vertical halfline below p intersects some point of V at the boundary of R_z. This point and z itself are both mapped to z by $*$. Note that z must be in V, since z lies on the bisector between z and the segment incident to z. Figure 3.2 exhibits both kinds of collapsing.

3.3. The Algorithm for V^* and for V

A sweepline algorithm, similar to Algorithm 1, suffices to compute V^*. Since $*$ is not always one-one, V is not identical topologically to V^*. However, it is easy to determine how V differs from V^* as V^* is being constructed, and hence to construct V at the same time as V^*. Actually, in an implementation it is probably preferable to only construct V, and compute the mapping $*$ as necessary.

The algorithm uses a list L containing the regions and boundaries intersected by the horizontal line and a priority queue Q of events, each event associated with a point in the plane. Unlike Algorithm 1, we process all the events associated with a point in the plane all at once.

What are the possible events? An event may be an intersection of boundaries, a new set of sites, or both. If the event is a site, there are two cases depending on whether the site occurs in the interior of the Voronoi region of some other site, or on a boundary. Also, we process a segment site when its lexicographically least endpoint is encountered. Hence we divide the segments incident to a point site into those for which the point site is the lower endpoint and those for which the point site is the upper endpoint.

We give a case analysis to illustrate the actions necessary for each possible event. If the event at point p is simply the intersection of boundaries, or is a point site with no incident segments, then the necessary action is straightforward, similar to Algorithm 1. Another simple case is if p is a point site that is the upper endpoint of a single segment. Then p lies in the interior of the Voronoi region of the segment, and the bisector between point site p and the segment site must be created. See Figure 3.5a.

Now suppose p is both a point site and the lower endpoint of a single segment and lies in the interior of some Voronoi region of a site q. See Figure 3.5b. Then two bisectors need to be created: between p and q, and between p and the segment site incident to p. Notice that * maps two points of V to p. A degenerate case of this is if the segment is horizontal. See Figure 3.5c. Then the bisector between p and the segment is vertical, and the portion of it below p is collapsed into p. Three bisectors need to be created: betweeen p and q, between p and the segment site, and between q and the segment site. In V^*, point p is the endpoint of all three bisectors; in V the endpoint of the vertical segment below p is the endpoint of all three bisectors.

If p is the lower endpoint of several segments, still in the interior of the region for site q, the situation is similar. The bisectors necessary are between site p and q, between site p and the most counterclockwise segment, between site p and the most clockwise segment, and between adjacent pairs of segments. Again, if the most clockwise segment is horizontal, there is a degeneracy and an additional bisector must be created. Note that in this case two vertices of V and a vertical edge of V are all mapped by * to p.

If p is the upper endpoint of a single segment q and lower endpoint of at least one segment, then one additional complication can arise. If the angle between q and either its clockwise or counterclockwise neighbor is exactly π, then the Voronoi region for p degenerates to a section of a line. If the angle is less than π the Voronoi region for p disappears entirely. See Figure 3.5d. The bisectors created must be adjusted accordingly.

In the remaining cases point site p lies on a boundary. The first way this can happen is if p is the upper endpoint of at least two segments. Then p lies on the boundary separating every adjacent pair of segments. Note that these boundaries were created before p was encountered. If in fact p is the upper endpoint of three or more segments, there is also an intersection of boundaries at p. See Figure 3.5e. The bisectors necessary are similar to previous cases.

The final possibility is that point site p lies on the boundary between two sites q and r, at least one of which is not a segment incident to p. Again, the bisectors necessary are similar to previous cases. In this case it possible that two vertices of V not connected by an edge of V map to the same point of V^*: the center c of the circle with p, q and r on its boundary is always a vertex of V and p may be a vertex of V. See Figure 3.5f.

Theorem 3.1: Let S be a set of point and line segment sites, and $V=V(S)$. Then V^* or V can be computed in time $O(n \log n)$ and space $O(n)$.
Proof: Similar to the proofs of Theorems 2.2, 2.3, and 2.4, using the case analysis discussed above. □

4. Weighted Point Sites

As a final, easy, application of the sweepline technique we consider weighted point sites. Now every site is a point and has a nonnegative weight associated with it. We wish to partition the plane into regions so that each region consists of points closest to a particular site, where the metric is the sum of the weight of the site and the distance to the site. The Voronoi diagram that results has a similar appearance to the unweighted point site case, except that bisectors are sections of hyperbolas. Also, the region of a site may be empty if its weight is bigger than the weighted distance from some other site.

The algorithm for this case is almost identical to Algorithm 1, with two minor differences. First, the mapping * no longer fixes sites, but maps them upward by a distance equal to their weight. Explicitly, * is defined by $*(x,y) = (x, y+\min_{p \in S}\{d_p(x,y)+w_p)\}$, where w_p is the weight of site p. Second, the region of a site p may be empty. This happens if $*(p)_y$ is less than p_y+w_p; this can be tested when p is encountered by the sweepline. The remaining details are similar to Algorithm 1 and are omitted.

An alternative definition of weighted Voronoi diagram has been studied by Aurenhammer and Edelsbrunner [AE84]. In this model each site has a multiplicative weight: the metric is the Euclidean distance to a site multipled by its weight. The diagram that arises from this metric is quite different from the usual Voronoi diagram; for example, one Voronoi region may completely encircle another Voronoi region.

5. A Geometric Interpretation of *

The mapping * may appear to be somewhat mysterious. A three-dimensional version of the Voronoi diagram may elucidate the role of *. This interpretation was first suggested by Edelsbrunner[E85].

Fix a set of point sites S. We view the sites as lying in the $z=0$ plane of R^3. For p a site, let the *cone* of p, c_p, be $\{(x,y,z) \in R^3 : d_p(x,y)=z\}$. Let the surface C be $\{(x,y,z) : (x,y,z) \in c_p$, some p, and for all q, if $(x,y,z') \in c_q$ then $z \leqslant z'\}$. Finally, let D be the subset of points of C contained in two or more cones. Clearly the Voronoi diagram V is the projection of D onto the plane $z=0$ in the direction parallel to the z axis. Consider the "oblique" projection of R^3 onto the plane $z=0$ in the direction parallel to the line $\{x=0, y+z=0\}$. V^* is just the oblique projection of D, and R_p^* just the oblique projection of $c_p \cap C$.

This interpretation of V in three dimensions gives a different geometric explanation of the sweepline algorithm. Suppose the sweepline is the line $y=c$ (in the $z=0$ plane). Let P_c be the plane $y+z=c$. Notice that the portion of the Voronoi diagram V^* intersected by the sweepline is just the oblique projection of $P_c \cap D$. Hence rather than translating a line in the $z=0$ plane, we can imagine translating a plane parallel to P_c. The intersection of the plane with D gives exactly the sequence of edges of V^* intersected by the sweepline. It is clear that the first time that the translating plane intersects a cone C_p, the plane is tangent to the cone, and the oblique projection of the intersection is the site p.

The extension to additively weighted point sites is immediate from this interpretation. The cone for each site is simply pushed in the positive z direction by a distance equal to the weight of the site.

6. Conclusions

The algorithms presented in this paper are simple and asymptotically efficient. Several issues need to be addressed before they can be considered practical.

The first issue is numerical stability and the handling of degeneracies. Very little formal attention has been paid this issue for any Voronoi diagram algorithm, indeed for almost any geometric algorithm. The sweepline algorithm needs geometric primitives to create bisectors and to determine intersections of lines, the same as any Voronoi diagram algorithm. In addition, the sweepline algorithm must manipulate the transformation *. Now * can introduce numeric instability since even if it is mathematically one-one, it can map distant points very close. Most of the instabilities introduced by * can be addressed by performing geometric operations on untransformed objects, which is the natural way to perform the operations anyway. We hope to address numeric issues in a future paper.

A second issue is whether, in practice, other algorithms perform better than the sweepline algorithm. At least two other algorithms for Voronoi diagrams have been proposed that have expected linear running time on uniformly distributed data [BWY80] [OIM85]. These algorithms rely on bucketing techniques to obtain the linear time bound. The sweepline algorithm performs quite favorably, at least compared to the second of these algorithms, if similar bucketing techniques are used to implement the sweepline data structures. If comparsion-based data structures are used instead, the sweepline algorithm is not as efficient[T85].

Voronoi diagrams are often used for nearest neighbor searching. This use requires an algorithm to search the planar polygonal regions defined by the Voronoi diagram. A data structure for such searches has recently been proposed by Sarnak and Tarjan [ST85]. The data structure uses linear space and gives $O(\log n)$ search time, and appears relatively simple. A sweepline approach is used to produce the data structure from the polygonal regions. Thus a direct use of the data structure for nearest neighbor searching requires two sweeps: one to produce the Voronoi diagram, and the second to produce the data structure itself. An interesting question is whether the two sweeps can be combined together.

The combination of transformation and sweepline is a powerful technique, and the question arises of what other problems can be solved with it. In a companion paper we will consider the case of Voronoi diagrams of line segments under polygonal convex distance functions [CD85]. Convex polygonal distance functions generalize the metrics L_1 and L_∞ and have applications to versions of the piano mover's problem. (A preliminary version of the companion paper appeared in [F85], and Leven and Sharir [LS85b] obtain a similar result, though with a more complex algorithm.) Perhaps the sweepline algorithm is adequate for even more general metrics.

References

[AE84] F. Aurenhammer, H. Edelsbrunner, An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane, *Pattern Recognition* 17(2), 1984, pp. 251-257.

[BWY80] J.L. Bentley, B. W. Weide, A.C. Yao, Optimal Expected-Time Algorithms for Closest Point Problems, *ACM Transactions on Mathematical Software*, 6(4), 1980, pp. 563-580.

[CD85] L.P. Chew, R.L. Drysdale, Voronoi Diagrams based on Convex Distance Functions, *Proceedings of the Symposium on Computational Geometry*, June 1985, pp. 235-244.

[E85] H. Edelsbrunner, private communication, 1985.

[F85] S.J. Fortune, Fast Algorithms for Polygon Containment, *Automata, Languages, and Programming, 12th Colloquium*, in *Lecture Notes in Computer Science 194*, Springer-Verlag, New York, pp. 189-198.

[GS77] P.J. Green, R. Sibson, Computing Dirichlet Tesselations in the Plane, *Computer Journal*, 21(22), 1977, pp. 168-173.

[K79] D. Kirkpatrick, Efficient Computation of Continuous Skeletons, *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, 1979, pp. 18-27.

[LD81] D.T. Lee, R.L. Drysdale, Generalizations of Voronoi Diagrams in the Plane, *Siam J. Computing 10(1)*, 1981, pp. 73-87.

[LS80] D.T. Lee, B.J. Schacter, Two Algorithms for Constructing a Delauney Triangulation, *International Journal of Computer and Information Sciences*, 9(3), 1980, pp. 219-227+.

[LS85a] D. Leven,M. Sharir, Intersection Problems and Applications of Voronoi Diagrams, in *Advances in Robotics, Vol I: Algorithmic and Geometric Aspects of Robotics*, J. Schwartz, C.K. Yap, ed. Lawrence Erlbaum Associates, Inc, to appear.

[LS85b] D. Leven, M. Sharir, Planning a Purely Translational Motion for a Convex Object in Two-dimensional Space using Generalized Voronoi Diagrams, Technical Report 34/85, June 1985, Tel Aviv University

[L82] D.T. Lee, Medial Axis Transformation of a Planar Shape, *IEEE Trans. Pattern Analysis and Machine Intelligence*, VOL PAMI-4, 1982, pp. 363-369.

[OIM84] T. Ohya, M.Iri, K. Murota, Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms, *Journal of the Operations Research Society of Japan*, Vol 27(4), Dec 1984, pp. 306-336.

[P77] F.P. Preparata, The Medial Axis of a Simple Polygon, *Proc. 6th Symp. Math Foundations of Comp. Science, Lecture Notes in Computer Science, Vol 53*, Springer-Verlag, New York, pp. 443-450, 1977.

[SH75] M.I. Shamos, D. Hoey, Closest-Point Problems, *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, 1975, pp. 151-162.

[S83] R. Sedgewick, Algorithms, Addison Wesley, 1983.

[ST85] N. Sarnak, R.E. Tarjan, Planar Point Location using Persistent Search Trees, manuscript, 1985.

[T85] R. Tuminaro, Computation of the Two Dimensional Voronoi Diagram in Expected Linear Time, manuscript, 1985.

[Y84] C.K. Yap, An $O(n\log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments, manuscript, October 1984.

Algorithm 1:	Computation of $V^*(S)$.
Input:	S is a set of $n \geq 1$ points with unique bottommost point.
Output:	The bisectors and vertices of V^*.
Data structures:	Q: a priority queue of points in the plane, ordered lexicographically. Each point is labelled as a site, or labelled with a pair of boundaries. Q may contain duplicate instances of the same point with distinct labels; the ordering of duplicates is irrelevant. L: a sequence $(r_1, c_1, r_2, \cdots, r_k)$ of regions and boundaries. Note that a region can appear more than once on L.

```
1. initialize $Q$ with all sites
2. $p \leftarrow$ extract_min($Q$)
3. $L \leftarrow$ the list containing $R_p$.
4. **while** $Q$ is not empty **begin**
5.    $p \leftarrow$ extract_min($Q$)
6.    **case**
7.    $p$ is a site:
8.        Find an occurrence of a region $R^*_q$ on $L$ containing $p$.
9.        Create bisector $B^*_{pq}$.
10.       Update list $L$ so that it contains ..., $R^*_q$, $C^-_{pq}$, $R^*_p$, $C^+_{pq}$, $R^*_q$, ... in place of $R^*_q$.
11.       Insert intersections between $C^-_{pq}$ and $C^+_{pq}$ with neighboring boundaries into $Q$.
12.   $p$ is an intersection:
13.       Let $p$ be the intersection of boundaries $C_{qr}$ and $C_{rs}$.
14.       Create the bisector $B^*_{qs}$.
15.       Update list $L$ so it contains $C_{qs} = C^-_{qs}$ or $C^+_{qs}$, as appropriate, instead of $C_{qr}$, $R^*_r$, $C_{rs}$.
16.       Delete from $Q$ any intersections between $C_{qs}$ and their neighbors.
17.       Insert any intersections between $C_{qs}$ and its neighbors into $Q$.
18.       Mark $p$ as a vertex and as an endpoint of $B^*_{qr}$, $B^*_{rs}$, and $B^*_{qs}$.
19.   **end**
```

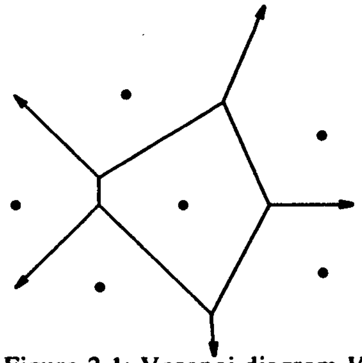
Figure 2.1: Voronoi diagram V

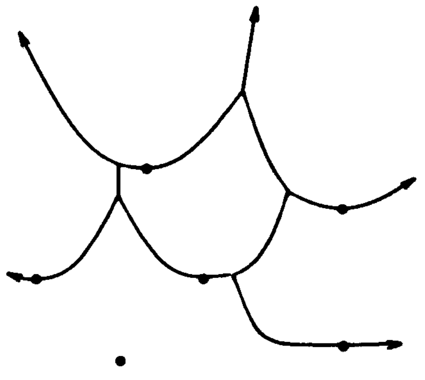
Figure 2.2. V^* for points in Figure 2.1

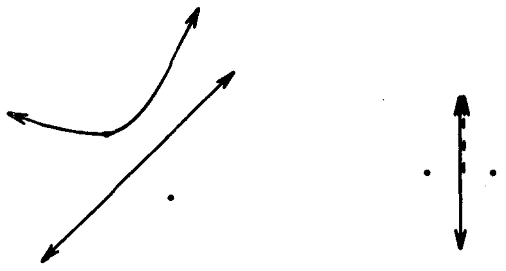

Figure 2.3. Bisectors and transformed bisectors

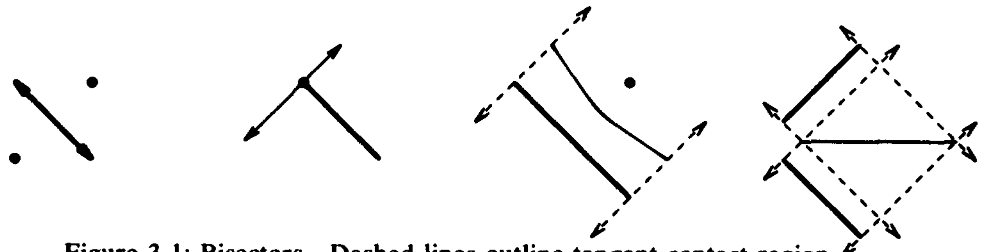

Figure 3.1: Bisectors. Dashed lines outline tangent contact region.

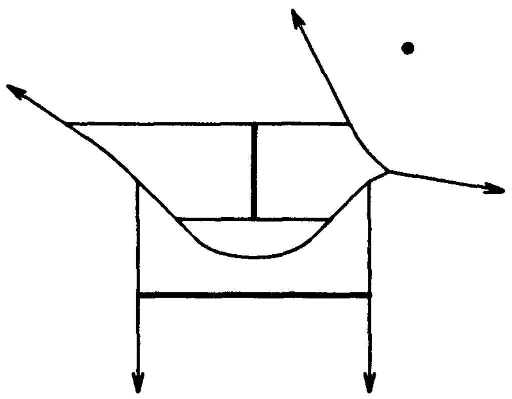

Figure 3.2: Voronoi diagram of line segments

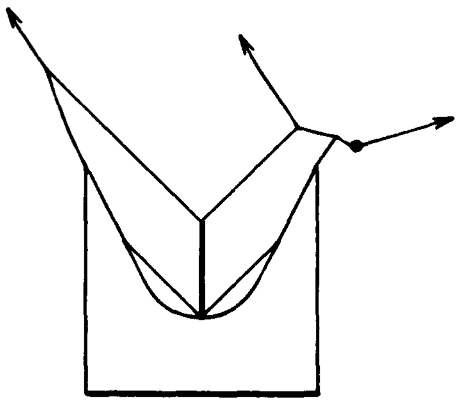

Figure 3.3: Transformed Voronoi diagram

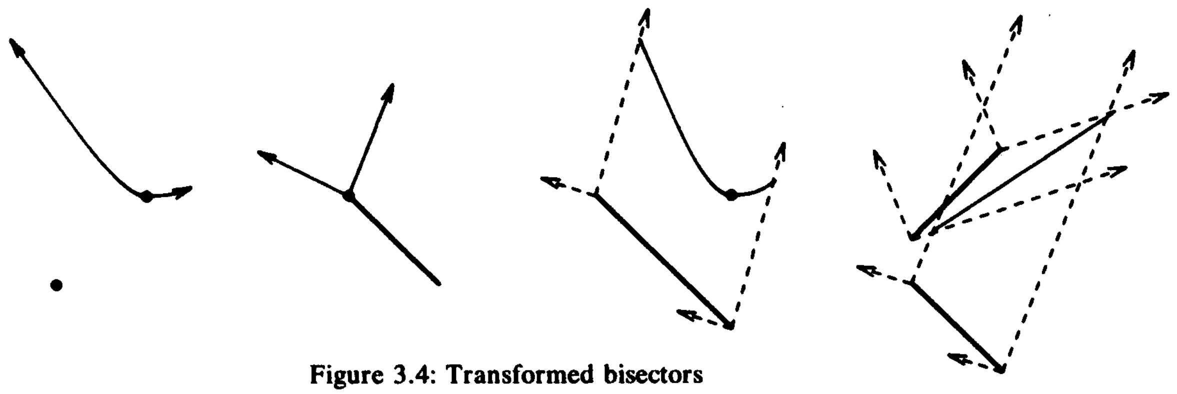

Figure 3.4: Transformed bisectors

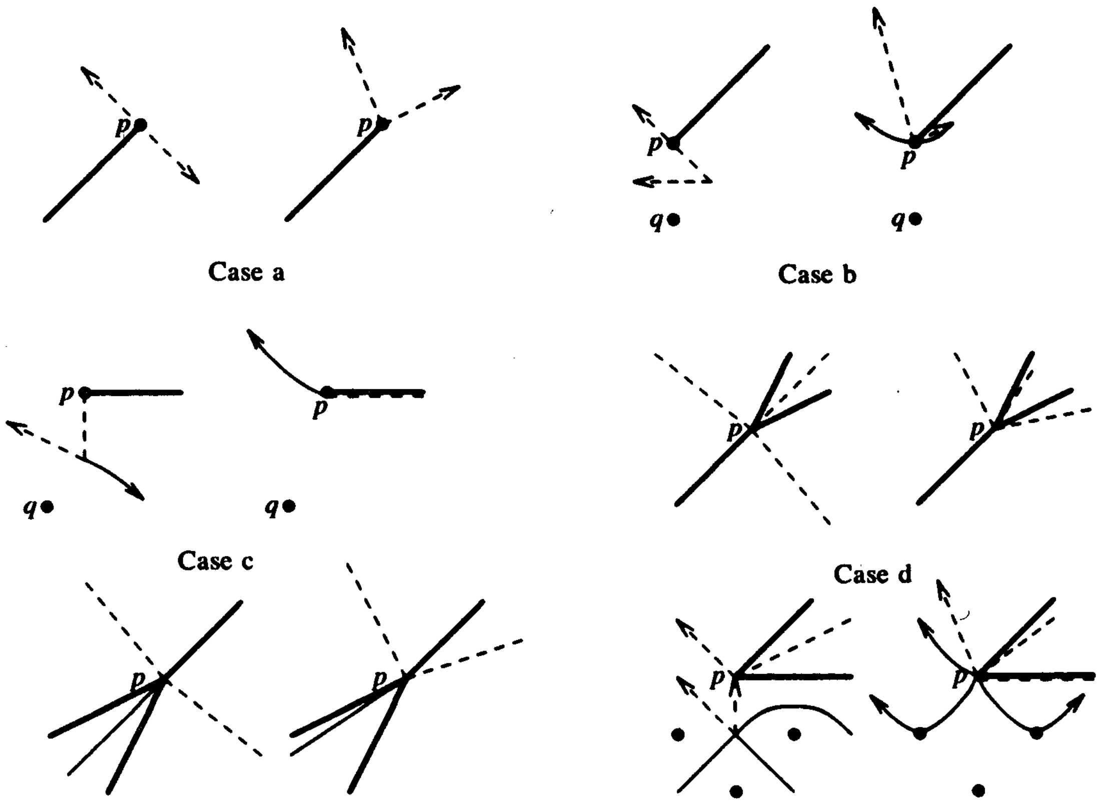

Figure 3.5: Dashed lines indicate bisectors new at p