An $O(n \log n)$ Plane-Sweep Algorithm for L_1 and L_∞ Delaunay Triangulations

by

Gary M. Shute
Linda L. Deneen
Clark D. Thomborson(a.k.a. Thompson)

Technical Report 87-5

University of Minnesota
Duluth

Computer Science
Mathematics
and Statistics

An $O(n \log n)$ Plane–Sweep Algorithm for L_1 and L_∞ Delaunay Triangulations

by

Gary M. Shute
Linda L. Deneen
Clark D. Thomborson(a.k.a. Thompson)

Technical Report 87-5

Department of Computer Science
University of Minnesota
Duluth, Minnesota

September, 1987

An $O(n \log n)$ Plane-Sweep Algorithm for L_1 and L_∞ Delaunay Triangulations

Gary M. Shute

Linda L. Deneen

Clark D. Thomborson(a.k.a. Thompson)

Department of Computer Science

University of Minnesota, Duluth

Duluth, Minnesota 55812

Keywords: Delaunay triangulation, plane-sweep algorithm, Voronoi diagram, L_1 metric, L_∞ metric, computational geometry, minimal spanning tree.

Abstract

The *Delaunay diagram* on a set on points in the plane, called *sites*, is the straight-line dual graph of the Voronoi diagram. When no degeneracies are present, the Delaunay diagram is a triangulation of the sites, called the *Delaunay triangulation*. When degeneracies are present, edges must be added to the Delaunay diagram to obtain a Delaunay triangulation. In this paper we describe an $O(n \log n)$ plane-sweep algorithm for computing a Delaunay triangulation on a possibly degenerate set of sites in the plane under the L_1 metric or the L_∞ metric.

1 Introduction.

The *Voronoi diagram* on a set of points in the plane, called *sites*, is a subdivision of the plane into regions, each site corresponding to a single region consisting of all points in the plane that are closer to that site than to any other [8,19]. The *Delaunay diagram* is the straight-line dual graph of the Voronoi diagram [9]. When no degeneracies are present, the Delaunay diagram is a triangulation of the sites, called the *Delaunay triangulation*. Degeneracies will be discussed carefully in Section 2.

Many problems that are described as applications of the Voronoi diagram are, in fact, solved by using the Delaunay triangulation, so algorithms that compute the Delaunay triangulation directly are useful in their own right [5,9,13,17]. Among these applications are minimal spanning tree [10,19,21], all nearest neighbors [19], relative neighborhood graph [21,22], and heuristics for various optimization problems [3,16]. Although the original work on Voronoi and Delaunay diagrams was done for the Euclidean (L_2) metric, it has since been extended to convex distance functions [4], including the L_p metrics [4,10,12,14]. The L_1 metric is of special interest to those developing heuristics for approximating the rectilinear minimal Steiner tree, a problem arising in the design of VLSI circuits [2,11,15,20].

Three computational techniques have been used to compute Voronoi and Delaunay diagrams. The incremental technique, where sites are added to the diagram one at a time, has a worst-case complexity of $O(n^2)$ [8,13,17].

The divide-and-conquer technique has a worst-case complexity of $O(n \log n)$ [4,5,9,10,12,13,14]. Most recently Fortune has developed an $O(n \log n)$ plane-sweep algorithm for computing the L_2 Voronoi diagram and Delaunay triangulation [7]. His algorithm avoids the difficult merge step of the divide-and-conquer technique.

In this paper we describe an $O(n \log n)$ plane-sweep algorithm for computing L_1 and L_∞ Delaunay triangulations on sets of sites in the plane. We prove our algorithm's correctness by direct geometrical argument. An alternative method of proof might be through a generalization of Fortune's hyperbolic transformation $*$ [7], although this proof would be complicated by the fact that $*$ is singular for L_1 and L_∞. In fact, the transformation $*$ is singular even in L_2 whenever the point set is degenerate. Nevertheless, it appears that our algorithm is a natural extension of Fortune's in which substantial simplifications can be seen. First, we are not faced with the problem of inverting a singular transformation. Second, our calculations are simple arithmetics as opposed to the quadratic forms required in L_2. Third, we do not compute Voronoi edges, which streamlines our data structures. Finally, we compute a Delaunay triangulation even when degeneracies are present.

Another interpretation of our triangulation algorithm is as an extension of Fortune's algorithm for polygon containment [6]. Viewed in this light, we have extended Fortune's definition of a polygonal obstacle to allow point obstacles. The problem here is that in [6] an obstacle must have a set of edges upon which

normal vectors can be erected. We allow the sides of our square object to be parallel to the coordinate axes, something that is specifically excluded from Fortune's treatment. Finally, we can handle degenerate point sets.

2 Fundamentals

The L_1 and L_∞ distance functions are given by

$$d_1((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

and

$$d_\infty((x_1, y_1), (x_2, y_2)) = max(|x_1 - x_2|, |y_1 - y_2|).$$

Distances in both of these metrics are dependent upon the location of the coordinate axes. Circles in the L_1 metric are squares with diagonals parallel to the coordinate axes, and circles in the L_∞ metric are squares with sides parallel to the coordinate axes.

The L_1 and L_∞ metrics have a useful relationship to one another, as noted in [13]. When points in the plane under the L_1 metric are transformed by using

$$u \leftarrow x + y, v \leftarrow y - x,$$

then

$$d_\infty((u_1, v_1), (u_2, v_2)) = d_1((x_1, y_1), (x_2, y_2)).$$

It follows that an algorithm that computes an L_∞ Delaunay triangulation on a set of sites by reporting an appropriate set of edges between pairs of sites

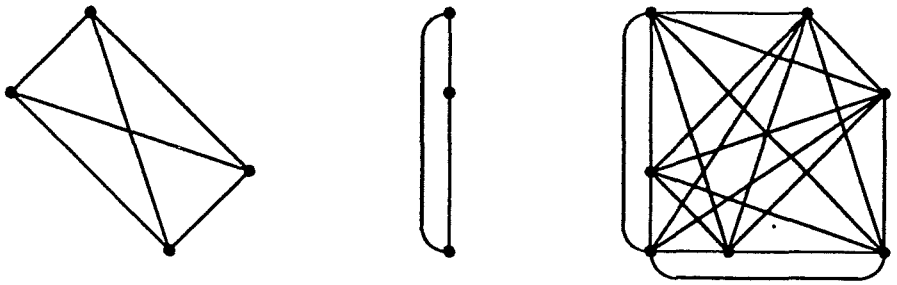
can also be used to report an L_1 Delaunay triangulation on a set of sites by adding a preprocessing step in which the sites are transformed by the preceding transformation. Throughout the remainder of this paper, we will concentrate on the L_∞ problem.

It is well-known that two sites in an L_2 Delaunay diagram have an edge between them if and only if there is a circle through the two sites containing no other sites in its interior or on its boundary. If $k \geq 4$ sites lie on the boundary of a circle with empty interior, their respective Voronoi regions meet in a single point, and the face in the corresponding Delaunay diagram containing this point is a k-gon. Such degeneracies prevent the Delaunay diagram from being a triangulation, and extra edges between sites on this empty circle must be added to make it one. See [9] for a careful discussion of these degeneracies in L_2.

In L_∞ the analogue of the L_2 circle is a square with sides parallel to the coordinate axes. Following [9] we say that an edge between two sites p and q is an (L_∞) *Delaunay edge* if there is a square with sides parallel to the coordinate axes containing p and q on its boundary and having no sites in its interior. We call such a square an *empty square through p and q*. If there is an empty square through p and q with no other site on its boundary, then $\overline{pq}$ is called a *strictly Delaunay edge*. Throughout the rest of the paper, all squares and rectangles will have their sides parallel to the coordinate axes.

If an empty square has multiple sites on its boundary, there may be crossing or overlapping Delaunay edges. More specifically, four or more sites on the

boundary of the square produce crossing Delaunay edges, and three or more sites on the same vertical or horizontal line produce overlapping Delaunay edges. See Figure 1.



(a) Four sites on the boundary of an empty square have a pair of crossing Delaunay edges.

(b) Three sites on the same vertical (horizontal) line have a Delaunay edge overlapping two others.

(c) Multiple sites on the boundary of an empty square cause multiple crossings and overlaps of Delaunay edges.

Figure 1

In L_2 the Delaunay diagram is defined to be the dual of the Voronoi diagram. In L_∞ the Voronoi diagram itself is not always well-defined. This is because the Voronoi diagram is constructed from the bisectors of pairs of points, and pairs of points on the same vertical or horizontal line have bisectors of dimension two. See Figure 2.

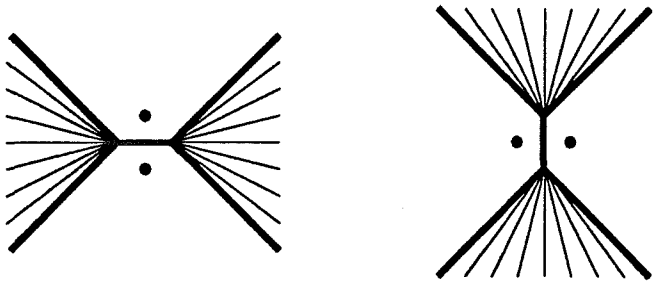
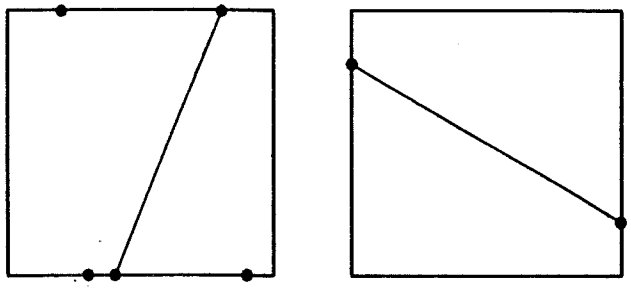


Figure 2

To maintain the property that the Voronoi diagram divides the plane into regions, one site per region, one can replace each region in the bisectors above by a ray from the vertex of the region, as in [4,10,12,14]. In light of all this, we will say that a set of sites in the L_∞ metric *has degeneracies* if it contains two or more sites on the same horizontal or vertical line or it contains four or more sites on the boundary of the same empty square.

We would like to clarify the definition of a Delaunay triangulation in the presence of degeneracies in L_∞, but first we need a few preliminaries. Following [18] we refer to a planar graph that is embedded in the plane so that its edges are mapped to straight line segments as a *planar straight-line graph* or *PSLG*. A PSLG is called a *triangulation* if every interior face is a triangular. A PSLG G is said to satisfy the *square crossing condition* if for every square R through sites in S, with sites on both of two opposite sides of R and no sites on the

interiors of the other two sides, there is an edge in G that crosses the square. See Figure 3.



The square crossing condition

Figure 3

Now we define an (L_∞) ***Delaunay triangulation*** on a set S of sites in the plane to be a PSLG G on S such that G is a triangulation, every edge of G is Delaunay, and G satisfies the square crossing condition. The inclusion of the square crossing condition in this definition needs some justification. First of all, it is easy to see that the square crossing condition implies G is connected, a desirable trait. Secondly, every strictly Delaunay edge $\overline{pq}$ has an empty square R through p and q with no other sites on its boundary. By shrinking and sliding R if necessary, it is possible to get p and q on opposite sides of R, whence the square crossing condition ensures that $\overline{pq}$ is in G. Therefore, for a set of sites with no degeneracies, G contains only the strictly Delaunay edges on S and is the usual dual of the Voronoi diagram. Finally, the square crossing condition is a generalization of Preparata and Shamos's Lemma 6.2 in [18, p. 220], and with

it, their proof can be modified to show that G contains a minimal spanning tree on S.

3 Description of the Algorithm.

In this section we give an informal description of an algorithm to produce an L_∞ Delaunay triangulation on a set of distinct sites in the plane. The formal specification of the algorithm is given in the next section, and the proof of correctness is in Section 5. This algorithm uses the plane-sweep technique, where the sweep line L is vertical and moving from left to right across the plane. As is usual in a plane-sweep algorithm, we use a priority queue X to control the position of the sweep line. Events in X are of two types: activation of a site and inactivation of a site. The details of the prioritization of these events in X will be discussed shortly. When an activation record is produced by X, its site is inserted into a second data structure, Y, and that site is called *active* as long as it remains in Y. When an inactivation record is produced by X, its site is deleted from Y, and the site becomes *inactive*.

The data structure Y is a dictionary, and sites are stored in Y in reverse lexicographic order on their coordinates (x, y); that is, the sites are ordered first on y and then on x. It will be shown that adjacent sites in Y have a Delaunay edge between them. It is these Delaunay edges that will be reported by the algorithm, and we will prove that they form a Delaunay triangulation. At each step of the algorithm, the Delaunay edges joining adjacent sites in Y form a

polygonal path with endpoints monotonically nondecreasing in y. We call this path the *Y-frontier.*

Next we will describe the interaction between the two data structures X and Y. We begin the algorithm by inserting activation records for all the sites into X, using the usual lexicographic order on the coordinates of the sites to determine priority: sites smaller in the lexicographic order have higher priority. Dummy sites $(-\infty, -\infty)$ and $(-\infty, +\infty)$ are placed in Y to ensure that all other sites have two neighbors in Y, making computation simpler. It is easy to avoid reporting the edges with dummy endpoints if desired. The algorithm is driven by X: X produces the next event, causing one or two Delaunay edges to be reported and some updates to be made to the data structures. When the event produced by X is an activation for site p, p is inserted into Y, and edges are reported between p and its two new neighbors in Y. When the event produced is an inactivation for site p, p is deleted from Y, and a new edge is reported between p's former neighbors. Moreover, with either type of event, it may be necessary to insert new inactivation records into X or to change the priority of inactivation records that are already there. Because X produces sites for activation sequentially, we can think of sites as having an age with respect to the algorithm. If site a's activation record is produced by X before site b's, we say that a is *older than* b or that b is *younger than* a.

Next we examine the management of the inactivation records in X. We must consider the circumstances under which a site p is due to become inactive: there

must be sites to p's right, both above and below it, that prevent an empty square with p on its left boundary from being maintained as the sweep line L moves to the right. When this occurs, p can no longer be the endpoint of a Delaunay edge whose other endpoint is to the right of L, so p should become inactive. The obvious candidates for the bounding sites that cause this inactivation are the neighbors of p in Y; if they both lie to the right of p, we set up an inactivation record for p in X. Let $p = (p_x, p_y)$; let $r = (r_x, r_y)$ be the predecessor of p in Y; let $q = (q_x, q_y)$ be the successor of p in Y. If $r_x > p_x$ and $q_x \geq p_x$, then p must become inactive when L reaches $p_x + q_y - r_y$. See Figure 4.

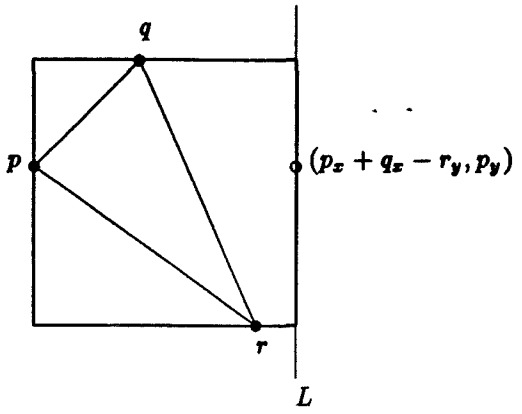


Figure 4

Thus, an inactivation record for p with priority $(p_x + q_y - r_y, p_y)$ is entered into X. If this inactivation record has the same priority as an activation record already in X, we give higher priority to the inactivation record. It is not hard to see that two inactivation records can never tie in X. Notice that p becomes

inactive somewhat prematurely in the degenerate case where $p_x = q_x$. By doing this, we prevent some Delaunay edges with p as endpoint from being reported, but the algorithm will still produce a Delaunay triangulation, as we will prove in section 5. Whenever a site in Y gets a new neighbor, we must check to see if it needs an inactivation record. In case a site already has an inactivation record but a new neighbor has caused a change in the inactivation priority, it is necessary to update the inactivation record, increasing its priority in X.

The position of the sweep line L is determined by the the event being processed. If the event is an activation for site $p = (p_x, p_y)$, then L is the line $x = p_x$. If the event is an inactivation for p with priority $(p_x + q_y - r_y, p_y)$, then in most instances L is $x = p_x + q_y - r_y$. If, however, the activation of a new site causes subsequent inactivations with $p_x + q_y - r_y$ smaller than the x-coordinate of the newly activated site, then L remains at the activation site while the inactivations are done. Thus, L moves from left to right as the algorithm proceeds. Moreover, the processing of each event from X moves the Y-frontier to the right. Inactive sites lie to the left of the Y-frontier, and the sites not yet seen, those with activation records still in X, lie to the right of the Y-frontier. Edges of the triangulation on or to the left of the Y-frontier have been reported; edges of the triangulation to the right of the Y-frontier have yet to be reported. See Figure 5.

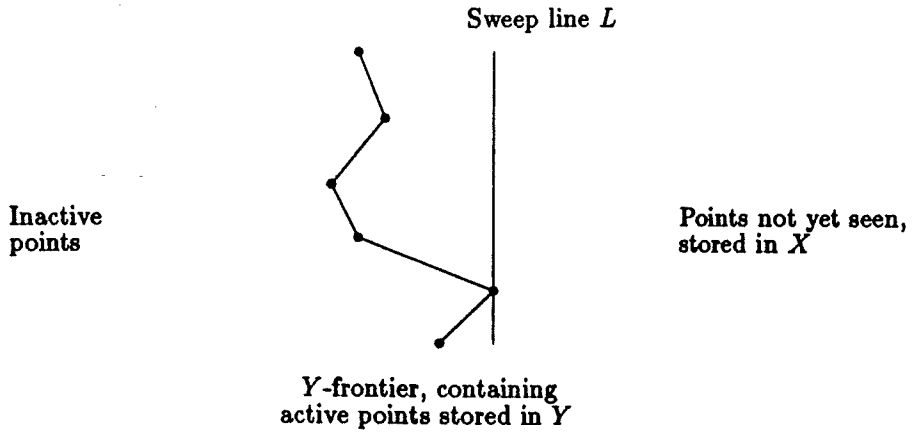


Figure 5

4 Formal Specification of the Algorithm

Input S is a set of distinct sites in the plane for which the algorithm will produce an L_∞ Delaunay triangulation.

Data Structure X is a priority queue containing activation and inactivation records for sites $p = (p_x, p_y)$ in S. The records of X are ordered lexicographically on the triples $(trans, y, status)$, where *trans* and y are real numbers under the usual ordering and *status* is the enumerated type $(inact, act)$. The activation record for p has $trans = p_x$, $y = p_y$, and $status = act$. A site p has an inactivation record in X only if it has a predecessor $r = (r_x, r_y)$ in Y with $r_x > p_x$ and a successor $q = (q_x, q_y)$ in Y with $q_x \geq p_x$. In this case, $trans =$

$p_x + q_y - r_y$, $y = p_y$, and *status* = *inact*. X supports the following operations within time bound $O(\log k)$ when there are k records in X.

- Min(X) returns the minimal record in X.
- Insert($X, p, newtrans, newstatus$) inserts a new record for site p into X with *trans* = *newtrans* and *status* = *newstatus*.
- ChangePriority($X, p, newtrans$) changes the *trans* field of the inactivation record for p in X to *newtrans* and adjusts the location of this record in X accordingly.

Data Structure Y is a dictionary containing sites from S in reverse lexicographic order; that is, $(p_x, p_y) < (q_x, q_y)$ in Y if and only if (i) $p_y < q_y$ or (ii) $p_y = q_y$ and $p_x < q_x$. Y supports the following operations within time bound $O(\log k)$ when k sites are in Y.

- Insert(Y, p) inserts p into Y.
- Delete(Y, p) deletes p from Y.
- Successor(Y, p) returns the site above p in Y.
- Predecessor(Y, p) returns the site below p in Y.

The algorithm to form an L_∞ Delaunay triangulation on S is given in Figure 6. The algorithm for updating inactivation records is given in Figure 7.

```
Initialization:
    X ← ∅; For all p in S, Insert(X, p, p_x, act)
    Y ← {(−∞, −∞), (−∞, +∞)}
Triangulation:
while X is not empty
    P ← Min(X)
    if P is an activation record for p
        Insert(Y, p)
        Report edges E_1 between p and Successor(Y, p)
            and E_2 between p and Predecessor(Y, p)
        Update the inactivation records in X for
            the older endpoints of E_1 and E_2
    else {P is an inactivation record for p}
        q ← Predecessor(Y, p)
        Delete(Y, p)
        Report an edge E between q and Successor(Y, q)
        Update the inactivation record in X for
            the older endpoint of E
    end if
end while
```

Figure 6. Algorithm to form an L_∞ Delaunay Triangulation on S.

```
r ← Predecessor(Y, p)
q ← Successor(Y, p)
if (r_x > p_x) and (q_x ≥ p_x)
      if p has no inactivation record
            Insert(X, p, p_x + q_y − r_y, inact)
      else {p already has an inactivation record}
            ChangePriority(X, p, p_x + q_y − r_y)
      end if
end if
```

Figure 7. Algorithm to update the inactivation record of site p.

5 Proof of Algorithm Correctness.

The main result of this section is Theorem 5.8, showing that our algorithm produces a Delaunay triangulation. We prove this theorem by a sequence of shorter results. Throughout this section, we let S denote the set of sites and G denote the graph on S produced by our algorithm.

Lemma 5.1 *G is a PSLG on S.*

Proof: Each edge in G has endpoints in S and is a straight line. Thus, we need only show that G has no crossing or overlapping edges. It suffices to show that the new edges reported for an event produced by X do not cross or overlap any previously reported edges. There are two cases to consider.

First, suppose that an activation record for site p is to be inserted into Y between sites r and q, where r is the predecessor of q in Y. It follows from the orderings on X and Y that $r_y < q_y$. Let U be the region bounded on the left by $\overline{rq}$, below by $y = r_y$, above by $y = q_y$, and on the right by $x = p_x$. See Figure 8.

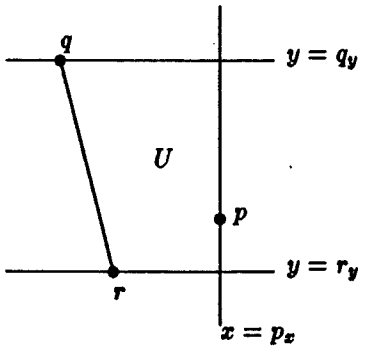


Figure 8

Just before p is inserted into Y, no edge in the Y-frontier crosses into the interior of U, because $\overline{rq}$ is in the Y-frontier, and the sites in the Y-frontier are monotone nondecreasing in their y-coordinates. Furthermore, since all other edges reported by the algorithm so far lie to the left of the Y-frontier, none of them can cross into the interior of U either. Therefore, $\overline{rp}$ and $\overline{pq}$ cross no edges previously reported by the algorithm. Moreover, since $p_x \geq r_x$ and $p_x > q_x$, $\overline{rp}$ and $\overline{pq}$ cannot overlap $\overline{qr}$.

Second, suppose that an inactivation record is produced for site p in Y, and suppose r is the predecessor and q is the successor of p in Y. Consider the region U bounded by $\overline{rp}$, $\overline{pq}$, $y = q_y$, $x = p_x + q_y - r_y$, and $y = r_y$, as shown in Figure

9.

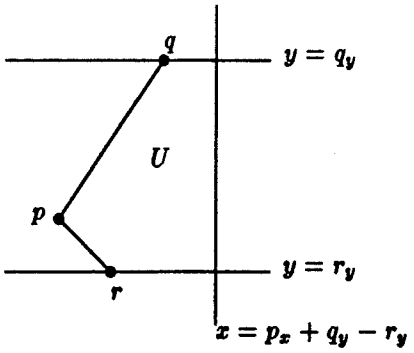


Figure 9

As in the first case, no previously reported edge can cross into the interior of U, so the new edge $\overline{rq}$ cannot cross any previously reported edge. Moreover, $q_x \geq p_x$ and $r_x > p_x$, so $\overline{rq}$ cannot overlap $\overline{rp}$ or $\overline{pr}$.

∎

Lemma 5.2 *G is a triangulation.*

Proof: Each event from X processed by the algorithm produces a triangular face. When the event is an activation for a site p, as in Figure 8, the new edges $\overline{rq}$ and $\overline{pq}$ form a triangle with edge $\overline{rq}$. When the event is an inactivation for a site p, as in Figure 9, the new edge $\overline{rq}$ forms a triangle with edges $\overline{rp}$ and $\overline{pq}$. If any of the endpoints involved in either of the cases are dummy endpoints, an implementation of the algorithm might not report the incident edges, but these would be edges in the exterior face in any case. Therefore, all interior faces of

G are triangles.

∎

Lemma 5.3 *G satisfies the square crossing condition.*

Proof: There are two cases to consider.

Case 1: Let R be a square with sites on its left and right boundaries and no sites in its interior or on the interiors of its top and bottom boundaries. Let p be the topmost site on its left boundary and q be the bottommost site on its right boundary. We will show that $\overline{pq}$ is in G. See Figure 10.

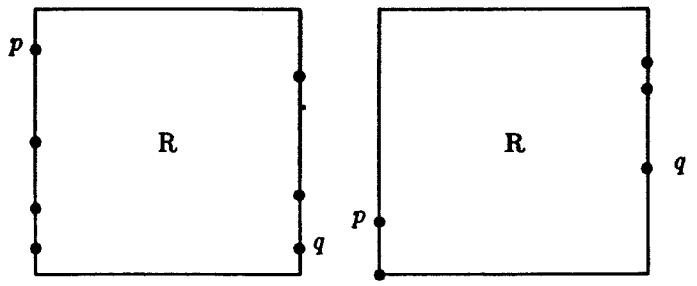


Figure 10

If p were to become inactive before q becomes active, it must have a predecessor and a successor in Y that are both younger than p. Because R is empty, p's predecessor a must be in the region A below R, and its successor b must be in the region B above R. See Figure 11.

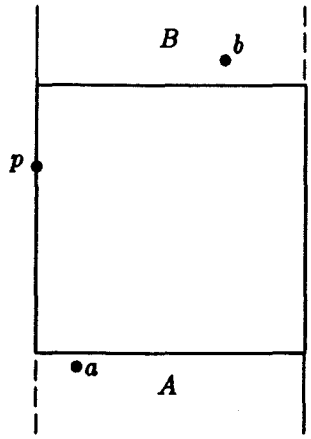


Figure 11

Then $p_x + b_y - a_y > q_x$, which implies that q becomes active before p becomes inactive.

Next we need to show that p and q become adjacent in Y. Consider the strip U bounded by $y = q_y$, $y = p_y$, and the left boundary of R. The top and right boundaries of U are closed, and the bottom boundary is open. See Figure 12.

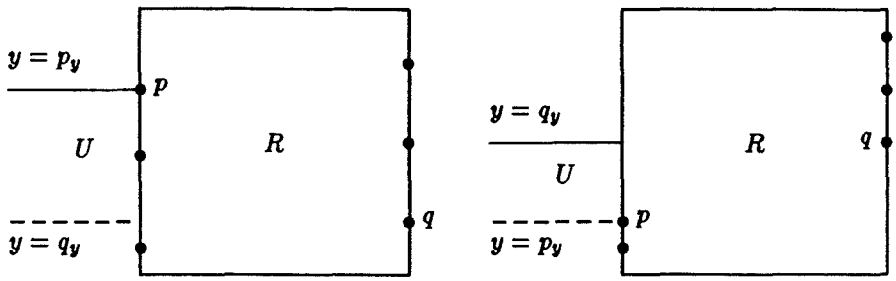


Figure 12

Let u be the oldest active site in U immediately after q has entered Y. If $u = p$, then p and q are already adjacent in Y; otherwise u has a predecessor t and a successor v in $U \bigcup \{p, q\}$ that are both younger. Thus, the inactivation time for u is $u_x + v_y - t_y \leq u_x + |p_y - q_y| < q_x$. Therefore, u becomes inactive before p becomes inactive and before any new sites can be inserted into Y. The same argument can be repeated for any other active sites in U, so p and q must become adjacent in Y, and $\overline{pq}$ is an edge of G.

Case 2: Let R be a square with sites on its top and bottom boundaries and no sites in its interior or the interiors of its left or right boundaries. Let p be the leftmost site on the top boundary of R and q be the rightmost site on the bottom boundary of R. We will show that $\overline{pq}$ is in G. See Figure 13.

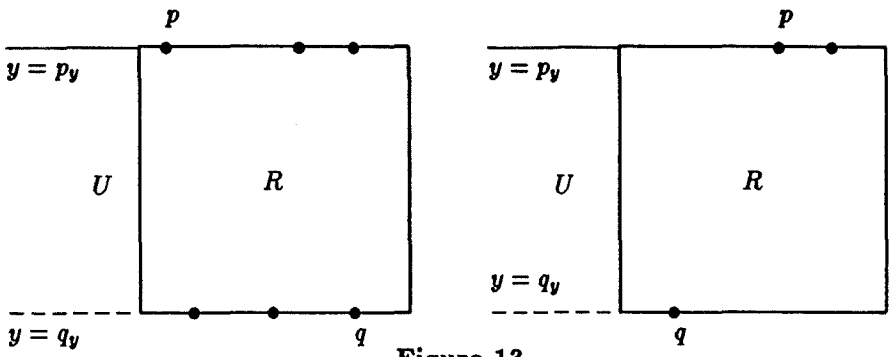


Figure 13

If p is older than q, the argument given in case 1 applies again to show that p is active when q becomes active. If q is older than p, a symmetric argument holds to show that q is active when p becomes active. Finally, we apply the same

argument as in case 1 to show that all active sites in U become inactive before the older of p and q and before any new sites can be entered into Y between p and q, so that p and q become adjacent in Y and $\overline{pq}$ is in G.

∎

It remains to prove that every edge of G is Delaunay. In order to simplify the proof of this fact, it is helpful to show first that G satisfies a related property. We say that a PSLG satisfies the *empty rectangle condition* if for every edge $\overline{pq}$ in the graph, the rectangle with sides parallel to the coordinate axes and $\overline{pq}$ on its diagonal has interior free of any sites. Notice that this condition is satisfied vacuously if p and q are on the same vertical or horizontal line.

Lemma 5.4 *G satisfies the empty rectangle condition.*

Proof: We proceed inductively. The first edge reported by the algorithm clearly has an empty rectangle. Suppose that the edges reported during the processing of the first k events all have empty rectangles.

If the $(k+1)$st event is an activation, the situation is described in Figure 8. It was proved in Lemma 5.1 that U is empty. By the inductive hypothesis, $\overline{rq}$ has an empty rectangle R, so $R \bigcup U$ contains empty rectangles for both new edges $\overline{rp}$ and $\overline{pq}$.

If the $(k+1)$st event is an inactivation, the situation is described in Figure 9. It was proved in Lemma 5.1 that U is empty. By the induction hypothesis, $\overline{rp}$ and $\overline{pq}$ have empty rectangles R_1 and R_2, so $R_1 \bigcup R_2 \bigcup U$ contains an empty rectangle for the new edge $\overline{rq}$.

Next, we prove another lemma, which will be used to prove Theorem 5.6. First we need a definition: a PSLG satisfies the *rectangle crossing condition* if for every rectangle R with sides parallel to the coordinate axes, width w, length $l > w$, no sites on its interior, and sites on the interiors of both of its long sides, there is an edge of the graph that intersects the interiors of both of the long sides of R.

Lemma 5.5 *If H is a PSLG that satisfies the square crossing condition, then H satisfies the rectangle crossing condition.*

Proof: We can assume without loss of generality that the long sides of R are horizontal. Let s be either the leftmost site on the interior of the top side of R or the leftmost site on the bottom side of R, whichever is further right. Moreover, we can assume without loss of generality that s is on the top side of R. Now let t be the site on the interior of R's bottom side that is on or to the left of the vertical line through s and has minimal distance from the vertical line through s. See Figure 14.

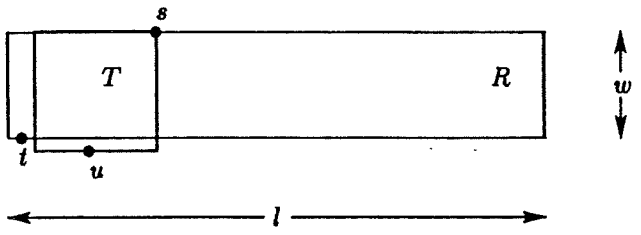


Figure 14

If $s_x - t_x \leq w$, then it is easy to see that there is a square with side w in R that has sites only on its top and bottom sides, with s on its top side and t on its bottom side. The square crossing condition ensures, therefore, that there is an edge in H that intersects the interiors of the top and bottom sides of R.

If $s_x - t_x > w$, let u be a site with $u_x \leq s_x$, $u_y \leq s_y$, and having minimal distance d from s. This is the case shown in Figure 14. Notice that $d \leq s_x - t_x$, so $u_x \geq t_x$. By the choice of u, the square T with side d and s at its upper right corner has an empty interior and no sites on its top or right sides except for s and possibly u (at the lower right corner). If there are any sites on the bottom side of T other than at the lower left corner, then T can be shifted slightly to the right to produce a square with sites only on its top and bottom sides. The square crossing condition then ensures an edge that intersects the interiors of the top and bottom sides of R. If there are no sites on the bottom side of T except possibly at the lower left corner, then the only sites on T are s and one or more sites on the left side of T that are on or below the bottom side of R. In this final case, the square crossing condition again ensures an edge of H intersecting the interiors of the top and bottom sides of R.

∎

We now prove a general result, from which our desired result follows.

Theorem 5.6 *If H is a PSLG satisfying the empty rectangle condition and the square crossing condition, then every edge in H is Delaunay.*

Proof: Let $\overline{pq}$ be an edge of H. Without loss of generality we can assume that the horizontal distance from p to q is greater than or equal to the vertical distance from p to q. Let U be the region between the vertical lines through p and q; let R be the rectangular section of U between the horizontal line through the lowest sites in the interior of U above $\overline{pq}$ and the horizontal line through the highest sites in in the interior of U below $\overline{pq}$. The empty rectangle condition implies that p and q are on the left and right sides of R. If the height of R is less than the width of R, then Lemma 5.5 implies that there is an edge of H crossing the interiors of both the top and bottom sides of R. This edge would cross $\overline{pq}$, however, contradicting the assumption that H is a PSLG. Therefore, the height of R is greater than or equal to the width of R, so R must contain an empty square through p and q, whence $\overline{pq}$ is Delaunay.

∎

Corollary 5.7 *Every edge in G is Delaunay.*

The proof of the algorithm's correctness now follows from Lemmas 5.1, 5.2, 5.3, 5.4, and Corollary 5.7.

Theorem 5.8 *G is a Delaunay triangulation.*

6 Run Time of the Algorithm

The run time for the initialization step of the algorithm is $O(n \log n)$ to insert n sites into X. This can be improved to $O(n)$ using a technique described by

Bentley [1]. Because X contains at most two records for each site (one activation record and perhaps one inactivation record), the main loop in the triangulation step is entered at most $2n = O(n)$ times. Each execution of the loop takes in $O(\log n)$ time, so the run time for the entire algorithm is $O(n \log n)$. The space used is $O(n)$.

We have implemented the algorithm in C on an Encore Multimax. When run on a set of 1000 random sites, the program produced a Delaunay triangulation in 32.6 CPU seconds. A profile of the execution revealed that 88 per cent of this execution time was spent doing input and output. Execution of the program on a set of 25,000 random sites took 130 CPU seconds, excluding input and output.

Profiling further revealed that the program spent about 7.5 times as much time manipulating X as Y. This is not particularly surprising; we suspect that for a random point set, X contains approximately n records on the average and Y contains approximately $\sqrt{n}$ records on the average. The run time of the program could be improved even more in the average case by using a bucketing technique on the records in X [17]. Moreover, the algorithm can be adapted to handle very large data sets by dividing the plane into vertical strips, each containing approximately $\sqrt{n}$ sites, and processing the strips one at a time from left to right.

7 Conclusions

We have described an $O(n \log n)$ plane-sweep algorithm for computing a Delaunay triangulation on a set of sites in the plane under the L_∞ metric. A simple preprocessing step, when added to our algorithm, transforms it to an algorithm for computing an L_1 Delaunay triangulation. Our algorithm performs satisfactorily even when degeneracies are present, unlike many related algorithms.

References

[1] J. Bentley. Programming pearls. *Communications of the ACM*, 28(3):245–250, March 1985.

[2] M. W. Bern. Two probabilistic results on rectilinear Steiner trees. In *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, pages 433–441, May 1986.

[3] L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 169–177, 1986.

[4] L. P. Chew and R. L. Drysdale. Voronoi diagrams based on convex distance functions. In *Proceedings of the Symposium on Computational Geometry*, pages 235–244, 1985.

[5] R. A. Dwyer. A simple divide-and-conquer algorithm for constructing Delaunay triangulations in $O(n \log \log n)$ expected time. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 276–284, 1986.

[6] S. J. Fortune. A fast algorithm for polygon containment by translation. In *Automata, Languages, and Programming, 12th Colloquium, Lecture Notes in Computer Science 194*, pages 189–198, Springer-Verlag, New York, 1985.

[7] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 313–319, 1986.

[8] P. J. Green and R. Sibson. Computing Dirichlet tesselations in the plane. *Computer Journal*, 21(22):73–87, 1981.

[9] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[10] F. K. Hwang. An $O(n \log n)$ algorithm for rectilinear minimal spanning trees. *Journal of the ACM*, 26(2):177–182, April 1979.

[11] F. K. Hwang. An $O(n \log n)$ algorithm for suboptimal rectilinear Steiner trees. *IEEE Transactions on Circuits and Systems*, CAS–26(1):75–77, January 1979.

[12] D. T. Lee. Two-dimensional Voronoi diagrams in the L_p–metric. *Journal of the ACM*, 27(4):604–618, October 1980.

[13] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[14] D. T. Lee and C. K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. *SIAM Journal of Computing*, 9:200–211, 1980.

[15] J. H. Lee, N. K. Bose, and F. K. Hwang. Use of Steiner's problem in suboptimal routing in rectilinear metric. *IEEE Transactions on Circuits and Systems*, CAS–23(7):470–476, July 1976.

[16] A. Lingas. The greedy and Delaunay triangulations are not bad in the average case. *Information Processing Letters*, 22:25–31, 1986.

[17] A. Maus. Delaunay triangulations and the convex hull of n points in expected linear time. *BIT*, 24:151–163, 1984.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[19] M. I. Shamos and D. Hoey. Closest point problems. In *Proceedings of the 16th IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[20] J. M. Smith, D. T. Lee, and J. S. Liebman. An $O(n \log n)$ heuristic algorithm for the rectilinear Steiner minimal tree problem. *Engineering Optimization*, 4(4):179–192, 1980.

[21] K. J. Supowit. The relative neighborhood graph, with an application to minimal spanning trees. *Journal of the ACM*, 30(3):428–448, July 1983.

[22] G. T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.