A Simple Linear Time Greedy Triangulation Algorithm for Uniformly Distributed Points*

Robert L. Scot Drysdale†
Department of Computer Science
6211 Sudikoff Laboratory
Dartmouth College, Hanover NH 03755-3510, USA
scot.drysdale@dartmouth.edu

Günter Rote
Technische Universität Graz
Institut für Mathematik
Steyrergasse 30, A-8010 Graz, AUSTRIA
rote@ftug.dnet.tu-graz.ac.at

Oswin Aichholzer
Technische Universität Graz
Institut für Grundlagen der Informationsverarbeitung
Klosterwiesgasse 32/2
A-8010 Graz, AUSTRIA
oaich@igi.tu-graz.ac.at

February 16, 1995

1 Introduction

1.1 Overview of the Results

The greedy triangulation (GT) of a set S of n points in the plane is the triangulation obtained by starting with the empty set and at each step adding the shortest compatible edge between two of the points, where a compatible edge is defined to be an edge that crosses none of the previously added edges. In this paper we present a simple, practical algorithm that computes the greedy triangulation in expected time $O(n)$ and space $O(n)$, for n points drawn independently from a uniform distribution over some fixed convex shape $\mathcal{C}$.

This algorithm is an improvement of the $O(n \log n)$ algorithm of Dickerson, Drysdale, McElfresh, and Welzl [7]. It uses their basic approach, but generates only $O(n)$ plausible greedy edges instead of $O(n \log n)$. It uses some ideas similar to those presented in Levcopoulos and Lingas's $O(n)$ expected time algorithm [18]. Since we use more knowledge about the structure of a random point set and its greedy triangulation, our algorithm needs only elementary data structures and simple bucketing techniques. Thus it is a good deal simpler to explain and to implement than the algorithm of [18].

1.2 Background

Efficiently computing the greedy triangulation is a problem of long standing, going back at least to 1970 [9]. A number of the properties of the GT have been discovered [16, 19, 22, 23] and the greedy algorithm has been used in applications [5, 23].

The naive approach to computing the GT is to compute all $\binom{n}{2}$ distances, sort them, and then build the GT an edge at a time by examining each pair in order of length and adding or discarding it based on its compatibility with the edges already added. It is easy to see that this method requires $O(n^2)$ space and $O(n^3)$ time, because in the compatibility test each pair must be tested for intersection with $O(n)$ edges already in the partial triangulation. Gilbert [10] presented a data structure allowing an $O(\log n)$ time compatibility test and an $O(n \log n)$ time update, thus improving the algorithm's overall time complexity to $O(n^2 \log n)$ without adversely affecting space complexity. He does this by building a segment tree for each point in the set, where the endpoints of the "segments" are the polar angles between the given point and every other point

*This paper benefited from discussions with Franz Aurenhammer, Kurt Melhorn, Matt Dickerson, and Emo Welzl.
†This study was supported in part by the funds of the National Science Foundation, DDM-9015851, and by a Fulbright Foundation grant that helped support this author's sabbatical visit to the Freie Universität Berlin and a visit to Graz, where much of this work took place.

in the set. Manacher and Zobrist [23] have since given an $O(n^2)$ expected time and $O(n)$ space greedy triangulation algorithm that makes use of a probabilistic method for *pretesting* compatibility of new edges.

Dickerson, Drysdale, McElfresh, and Welzl [7] showed how to improve on this "generate and test" paradigm by supplying an edge compatibility test that takes constant time for a test or an update and by showing how to generate all "plausible" greedy edges in expected $O(n \log n)$ time. Our result reduces the expected number of edges generated to $O(n)$ and shows that their lengths can be bucket-sorted in $O(n)$ expected time, creating an $O(n)$ expected time algorithm.

An alternate approach to "generate and test" is to generate only compatible edges. One way to do this was discovered independently by Goldman [11] and by Lingas [20]. The method uses the generalized or constrained Delaunay triangulation [3, 14, 27]. The constrained Delaunay triangulation is required to include a set of edges E. The rest of the edges in the triangulation have the property that the circumcircle of the vertices of any triangle contains no point visible from all three vertices.

This alternate approach computes the constrained Delaunay triangulation of the points with the current set of GT edges as the set E. The next edge to be added to the GT can be found in linear time from the constrained Delaunay triangulation. The triangulation must then be updated to include the new edge in E, which takes $O(n \log n)$ time in the worst case. This gives an $O(n^2 \log n)$ time and $O(n)$ space algorithm, thus improving the space complexity of Gilbert's algorithm without affecting the worst case time. Lingas [20] shows that his method runs in $O(n \log^{1.5} n)$ for points chosen uniformly from the unit square.

Recently Levcopoulos and Lingas [17], and independently Wang [25], have shown how to do the update step in $O(n)$ time, using a modification of the linear-time algorithm for computing the Voronoi diagram of a convex polygon [1], leading to an $O(n^2)$ time and $O(n)$ space algorithm in the worst case. More recently Levcopoulos and Lingas give a modification of this algorithm that is expected to take $O(n)$ time for points uniformly distributed in a square [18]. These methods are elegant, but are significantly more complicated to implement than our method and should be slower for practical-sized problems. They do have one definite advantage over our method — they only require $O(n)$ space in the worst case, whereas our approach could take $O(n^2)$ space (although $O(n)$ space is expected with high probability).

The greedy triangulation and the minimum weight triangulation. One use of the greedy triangulation is as an approximation to the minimum weight triangulation (MWT). A Minimum Weight Triangulation (MWT) of a point set in the plane is a triangulation that minimizes the total length of all edges. The MWT arises in numerical analysis [19, 21, 24]. In a method suggested by Yoeli [28] for numerical approximation of bivariate data, the MWT provides a good approximation of the sought-after function surface. Wang and Aggarwal use a minimum-weight triangulation in their algorithm to reconstruct surfaces from contours [26]. Though it has been shown how to compute the MWT in time $O(n^3)$ for the special case of n-vertex polygons [13], there are no known efficiently computable algorithms for the MWT in the general case [24]. We therefore seek efficiently computable *approximations* to the MWT.

Although neither the GT nor the Delaunay triangulation (DT) yields the MWT [22, 21], the GT appears to be the better of the two at approximating it. In fact, for convex polygons the GT approximates the MWT to with a constant factor while the DT can be a factor of $\Omega(n)$ larger [16]. For general point sets, the DT can be a factor of $\Omega(n)$ larger than the MWT, but the best lower bound for the GT is $\Omega(\sqrt{n})$ [12, 15]. For points lying on a convex polygon or uniformly distributed points in a square, both the GT and the DT are expected to be within a constant factor of the MWT [18, 2]. Therefore a large amount of effort has gone into finding efficient methods for computing the greedy triangulation.

2 The Greedy Triangulation Algorithm that We Improve Upon

We assume throughout that the input consists of n points drawn independently from a uniform distribution over some fixed convex shape $\mathcal{C}$. For convenience we also assume that $\mathcal{C}$ has area 1, and we denote the perimeter of $\mathcal{C}$ by U. Some of the bounds that we derive depend on U, but we will regard U as constant. Dickerson, Drysdale, McElfresh, and Welzl [7] create fast greedy triangulation algorithms for uniformly distributed points by using the generate and test approach, but reducing the number of pairs generated and the time for a test from previous algorithms.

They define a plausible pair of points as follows. For two points p and q in the plane, and for a real number $r > 0$, let $D(p,q,r)$ denote the closed disk of radius r centered at $(p+q)/2$. The line through p and q defines two closed half-disks of $D(p,q,r)$, denoted by $D'(p,q,r)$ and $D''(p,q,r)$. (It does not matter which half is D' and D''.)

Given a set S of n points in the plane, they call a pair $\{p,q\}$ of points in S *plausible*, if $D'(p,q,r_1) \cap S = \emptyset$ or $D''(p,q,r_1) \cap S = \emptyset$, with $r_1 = \gamma|p-q|/2$. They show that if $\gamma = 1/\sqrt{5}$, only plausible pairs can be in a greedy triangulation.

They then show that for a set S of n points uniformly distributed over a convex shape $\mathcal{C}$, the expected number of plausible pairs is $O(n)$. They furthermore show in a precise way that every plausible pair is expected to be "short" (at most $O(\sqrt{\log n/n})$ long) or has both points "close" (within $O(\sqrt{\log n/n})$) to the boundary of $\mathcal{C}$. They define a candidate pair as a pair that is "short" enough or "close" enough to the boundary. They show that there are only $O(n \log n)$ candidate pairs, and that they are easy to generate by using fixed-radius near neighbor search and by pairing up points close to the boundary of $\mathcal{C}$.

This leads to their first algorithm. They first generate $O(n \log n)$ candidate pairs which are almost certain to contain all greedy edges. They then reduce this set to $O(n)$ plausible pairs by testing each candidate pair for plausibility in constant expected time per test. They sort these $O(n)$ plausible pairs in $O(n \log n)$ time. Finally they insert them into a partial greedy triangulation in order of length, using an edge compatibility test that requires $O(1)$ time for a test or a data structure update. This leads to an $O(n \log n)$ algorithm. (This version only finds the GT with very high probability, but they later modify it to guarantee that it always finds the greedy triangulation.)

Our algorithm is a modification of this algorithm. There are two major differences. First, instead of generating $O(n \log n)$ candidate pairs and then reducing them to $O(n)$ plausible pairs, we only generate $O(n)$ candidate pairs to start with, and guarantee that all plausible pairs are candidate pairs, so that the GT is a subset of these edges.

The second difference is the sorting. We show that we can sort these $O(n)$ candidate pairs in linear time using a bucket sort. The uniform distribution of the points is enough to let us show that a bucket sort is expected to run in $O(n)$ time. Given the set of candidate pairs in sorted order, we then use the edge compatibility test in [7] to find the triangulation in $O(n)$ expected time.

3 A Necessary Condition for an Edge to be in the Greedy Triangulation

Our algorithm uses a definition of plausibility based on shapes other than disks. It is based on the following lemma and corollary. First we need a definition.

Definition 1 *Let p, q be a pair of points in a set S. Consider the disc D of radius $\delta = |p,q|/(2\sqrt{5})$ centered at the midpoint of pq (cf. Figure 1). Let p_u and p_l be the upper and lower points of intersection between D and tangent lines from p, and define q_u and q_l similarly. Then we call the union of D, triangle pp_up_l, and triangle qq_uq_l the* exclusion region *for p and q. The segment pq divides the exclusion region into two* half-regions.

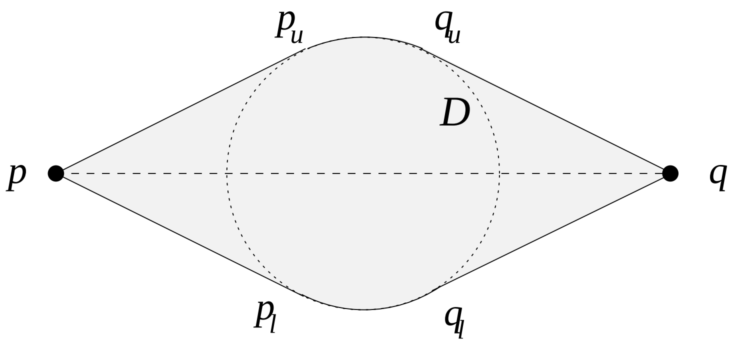

Figure 1: Our Exclusion Region

Lemma 1 *If each half-region of the exclusion region of two points p and q contains at least one point in S (other than p and q) then the edge pq cannot be in the GT of S.*

This lemma is an extension of Lemma 3 in [7], where the exclusion region was only disk D. It is proved in a similar way, and the figure in the proof appeared originally in that paper. Theorem 3 in Levcopoulos and Lingas [18] is also similar. We define a *plausible pair* (or a *plausible edge*) to be a pair where at least one of the halves of the exclusion region defined in the lemma contains no points from S.

For the proof of the lemma see the appendix.

We do not believe that our bound is tight. The worst example that we have been able to find is a 'distorted' diamond with pq as its diameter, and opposite edges of the same length, namely $\sqrt{8/17}\ |p,q|$ and $\sqrt{9/17}\ |p,q|$. There are two additional points just outside of the longer edges of the diamond, at distance $1/3\sqrt{9/17}\ |p,q|$ from p and q respectively. This case approaches the bound $\delta = \sqrt{2/17}\ |p,q|$ as closely as desired and thus disproves the conjecture of the Open Problem 3 in Dickerson, Drysdale, McElfresh, and Welzl [7].

From Lemma 1 we derive the following crucial corollary on which our algorithm is based.

Corollary 1 *Given four points p, q, a, and b with $|p,a|, |p,b| \leq |p,q|/2$ such that the angles apq and qpb are at most $\arctan(1/\sqrt{5})$ (a bit more than $24°$) and pq intersects ab, then pq cannot be a GT edge.*

Proof. Consider points p, a, b, and q as described in the lemma. Let m be the midpoint of pq. Consider a right triangle with pm as one leg and a segment cm of length δ perpendicular to pm as the other leg. There are two such triangles, and a must be contained in one of them and b in the other. But these two triangles are subregions of the two halves of the exclusion region for pq described in Definition 1. Therefore by this lemma pq cannot be a greedy edge. □

This bound could be slightly improved because a slightly larger circular sector of radius $|p,q|/2$ can be fit into the exclusion area.

4 Generating the Candidate Pairs

In this section we show how to generate $O(n)$ candidate pairs that include all plausible pairs in $O(n)$ time.

Here is an intuitive description of our algorithm: Grow two concentric circles C and D around p, D always double the size of C. If two points that have been passed by C are closer than $24°$, the wedge between them is closed off. All points that are swept by D and that are not closed off are matched with p. Of course, regions where D moves out of $\mathcal{C}$ can also be closed off. Our decomposition into fixed wedges is just an easy "bucketing" way to implement this, very much like in the linear-time solution of the MAX-GAP problem.

4.1 Description of the Approach

The points are treated one by one as follows. Consider a fixed point p. We divide the plane into s wedges of size $\gamma = 2\pi/s$ centered at p. For our algorithm, $s = 30$ and γ is $12°$.

Conceptually we sweep through each wedge W, moving a growing circular arc centered at p across W and examining points as we find them. As we sweep W we will keep track of the closest point w to p, the leftmost point l found so far, and the rightmost point r found so far.

We sweep each wedge until we are sure that we need not search further to find plausible pairs containing p. All points found in this process are paired with p and added to the set of candidate pairs to be sorted and tested for compatibility in the greedy algorithm.

The usual way that we know that we have swept far enough depends on the nearest neighbors of p in the two adjoining wedges. We call the wedge lying left (counterclockwise) from W wedge W_l and the one lying right W_r. We call the nearest neighbors to p in these wedges w_l and w_r respectively. The angles qpw_l and qpw_r are never more than $24°$ for any $q \in W$. We therefore need not consider points $q \in W$ such that $|p,q| \geq 2\max(|p,w_l|, |p,w_r|)$. By Corollary 1 such edges are not possible greedy edges.

However, if one of the adjacent wedges is empty (which is likely if p lies near the boundary of $\mathcal{C}$), then our wedge W cannot be bounded in this way. Always sweeping such a wedge to the border of $\mathcal{C}$ would be too expensive, so we must find an alternate way to bound W in this case.

This is where we use the leftmost and rightmost points. If l is the leftmost point found so far in W and we have swept far enough to ensure that l is indeed the leftmost point in all of W, we can use l in place of w_l in the above formula. (In fact, even if w_l exists we use l instead of w_l if it is closer to p.) A similar situation holds on the right side.

We summarize the above conceptual sketch of the algorithm.

1. For each wedge W, find the point w closest to p.
2. For each wedge W, repeat the following steps.

   Let W_l and W_r be the adjacent wedges of W.
   Denote by w_l and w_r the closest points to p in these wedges.
   Set the thresholds $t_l := |p, w_l|$ and $t_r := |p, w_r|$;
     (If any of w_l and w_r does not exist, set the corresponding quantity to ∞.)
   Start sweeping W, i.e., visit the points q in order of increasing distance $v = |p, q|$:
   while $v \leq 2\max(t_l, t_r)$ **and** $S(W, v)$ does not yet cover $W \cap \mathcal{C}$ **do**
     Let q be the next point swept.
     Generate the pair p, q as a possible edge.
     If this point is the first point swept, set $l := q$ and $r := q$;
     Otherwise, if q is to the left of l, set $l := q$;
     Otherwise, if q is to the right of r, set $r := q$;
     If the circular arc with radius v centered at p, between the left edge of W
       and the ray pl, lies outside $\mathcal{C}$, set $t_l := \min(t_l, |p, l|)$;
     If the circular arc with radius v centered at p, between the ray pr and the
       right edge of W lies outside $\mathcal{C}$, set $t_r := \min(t_r, |p, r|)$;
   end while

4.2 Implementation Details

Our implementation will first cover $\mathcal{C}$ by a grid of squares with side $1/\sqrt{n}$ and will bucket the points into these squares. On the average there will be one point per square, and there are $n + O(U\sqrt{n}) = O(n)$ squares.

For each point p we search out from p in each wedge W, examining points in overlapping squares until we find the nearest point w (or run out of possible squares). We can then continue searching in each wedge until we are beyond the distance at which further points in the wedge can possibly pair with p to form a greedy edge.

Practically, the approach of looking at wedges one at a time would cause squares near p to be re-examined lots of times. (But not more than s times.) Instead one should begin spiraling out from p, putting points into whichever wedges they belong. More specifically, we start with the square containing p, continue with all squares intersecting a circle of radius $1/\sqrt{n}$ around p, then all squares intersecting a circle of radius $2/\sqrt{n}$ that have not been examined, and so on. As wedges are *closed off* one should skip squares lying only within those wedges and look only into wedges that are still *active*. We quit when all wedges are closed off. We show in section 5.5 that we can afford to look at all points in a square even if only a part of the square intersects some active wedge.

So what the algorithm really does is as follows. Departing from the description in section 4.1, we don't vary v continuously, stopping at every distance where some event occurs, but vary it in fixed steps, and even that only approximately. For $i = 0, 1, 2, \ldots$, we go around the point p following a circle of radius $i/\sqrt{n}$. Every square that intersect this circle and an active wedge is handled completely, in the sense that we go through all its points in an arbitrary order, pair them with p, and update w, l, and r for their respective wedges, as appropriate. (Of course, a square that has been considered in the previous round need not be considered again.) Only after processing all these squares and points for round i we update the thresholds t_l and t_r for every wedge which is still active and decide if any of them can be closed. This is the case when either $v := i/\sqrt{n} > 2\max(t_l, t_r)$ or the wedge is exhausted. We can thus avoid such troubles as sorting the points in a square according to their distance from p, which we would have to do if we followed the sketched algorithm of section 4.1 literally.

This approach allows us to generate $O(n)$ candidate pairs which include all plausible pairs in time $O(n)$. The proof will be given in section 5.

In the analysis we will first assume that the algorithm proceeds as originally described above in section 4.1, and we will later account for the "error" that we make in being too liberal in the selection of squares and points that we examine.

The above description in section 4.1 assumes that the algorithm knows the shape $\mathcal{C}$ from which the points are drawn. If $\mathcal{C}$ is not given we first compute the convex hull of the points using one of the algorithms that perform this task in $O(n)$ expected time for uniformly distributed points [24]. We compute the area of this hull, scale the data so that the area becomes 1, and then use the hull instead of $\mathcal{C}$. The analysis is of course unchanged. The only place where $\mathcal{C}$ enters the algorithm is when testing if a wedge is exhausted, and whether the current leftmost or rightmost point is the true leftmost or rightmost point of the whole wedge.

The first test is very easy to carry out. For example, as we go around the circle of radius $i/\sqrt{n}$ in round i, we may build a list of squares to be visited in round $i+1$. In this list we skip all squares that do not intersect an active wedge or that lie completely outside $\mathcal{C}$. Every square which should appear in this list is adjacent to at least one other square on the list or to a square which is visited in round i, and thus this list can easily be built. With this list, a wedge is automatically and implicitly "closed off" as soon as all its squares that lie inside $\mathcal{C}$ have been visited.

For the second test we have to decide whether the angular region (sub-wedge) between, say, the ray pl and the left edge of W, may possibly contain any further points. As we walk around the circle of radius v and build up the list of squares for the next round, we can easily determine if the relevant angular region is closed off in this round, or whether any further squares will intersect this angular region in the next round. In the latter case we cannot be sure that l is the leftmost point. In the first case, we know that we have looked at all points that may possible have been further to the left than l. That is, we either have found a new point l or we have established that the old previous point l was indeed the leftmost point in its wedge.

In a preprocessing step, after initially computing the convex hull, we can determine for each square whether it intersects the boundary at all, and if so, we can find the intersection points of the hull boundary with the boundary of the square. There can be only a constant number (at most 8, theoretically) of such intersection points. Then we can determine which of the four adjacent squares intersect $\mathcal{C}$ or lie completely outside $\mathcal{C}$. With this information it is straightforward to carry out all the above operations.

4.3 Possible Improvements

The remaining part of this section discusses some possibilities to speed up the practical implementation. It can be skipped without affecting the remainder of the paper.

Dickerson et al. [7] use a plausibility test that runs in linear expected time to reduce the number of pairs that are sorted and tested for inclusion into the triangulation. It is based on bucketing. Our algorithm need not use such a plausibility test because it generates only a linear number of candidate pairs. However, the constants for space (and probably time) can be improved by pretesting the candidate pairs for plausibility.

One could use the method described in [7] to test for plausibility, but an alternate method is available. A simple way to test a pair (p, q) for implausibility when it is first found in wedge W is to look at the near neighbors and leftmost and rightmost points in the wedges that overlap the exclusion region for pq. Overlapping wedges consist of the wedge W and its two adjacent wedges. If one of these points lies in one half of the exclusion region and another lies in the other half, then pq is not plausible.

It is possible that an implausible pair is not rejected by this test, because the two points that prove the implausibility of pair (p, q) are not necessarily the closest or the leftmost or the rightmost points in their respective wedges. However, this test is fast and easy and should work often enough to make it worthwhile. The test of [7] is probably too expensive, in view of the fact that the remaining steps which an edge undergoes once it is generated are very simple and fast.

We find each pair of points twice, once from each endpoint. Thus we can search only half (say the upper half) of the wedges, plus two additional boundary wedges that are only searched out until the first point is found (to be used for bounding their neighbors). Then each pair would be found only when searching from the lower of the two points. Alternatively, we generate all pairs and eliminate those pairs which are generated only once. A radix sort based on the indices of the endpoints can accomplish this in linear time, or it can be done while sorting the generated edges by length. This involves more work but it reduces the number of edges for the rest of the algorithm.

5 Analysis of the Run Time

The sweep of the wedge W under consideration is stopped by the minimum of the two thresholds $t_l = \min(|w_l, p|, |l, p|)$ and $t_r = \min(|w_r, p|, |r, p|)$. They come, respectively, from the closest point w_l in W_l and the closest point w_r in W_r. Instead of thinking in terms of the wedge W being bounded, we will consider the wedges W_l and W_r doing the bounding. The wedge W_l contributes "half" of the bound for W and "half" of the bound for its left neighbor. We can thus construct the area around p to be searched by taking the union of $2s$ circular sectors. We define $S(W, v)$ as the circular sector which is the intersection of wedge W with a disk of radius v centered at the apex of W. For the wedge W_l, we construct $S(W, 2t_l)$ and a similar sector $S((W_l)_l, v)$ for the left neighbor of W_l. The union of these $2s$ sectors will be the area around p that needs to be considered, and its area will be less than the total area of all sectors.

We will show that the expected area of such a sector is $O(1/n)$, implying that the area that needs to be searched for a given point p before all wedges are closed off by points in adjacent wedges is $O(1/n)$. A constant number of points is thus examined for each p and the overall work is linear.

We analyze how the wedge W_l influences W via the threshold $t_l = \min(|w_l, p|, |l, p|)$, the situation for W_r being symmetric. The analysis is divided into two cases, corresponding roughly to the cases when $t_l = |w_l, p|$ and when $t_l = |l, p|$. Formally, the two cases of our analysis are distinguished as follows. Let b be the distance from p to the closest point of the boundary of $\mathcal{C}$ inside W_l.

- **Case M.** $b \geq t_l$: The arc sweeping W_l does not intersect the boundary of $\mathcal{C}$ before the distance $t_l = \min(|w_l, p|, |l, p|)$ is reached.
- **Case B.** $b < t_l$: The arc sweeping W_l intersects the boundary of $\mathcal{C}$ before the distance t_l is reached.

We first analyze case M, which is typical when the point p lies in the *middle* of $\mathcal{C}$. In this case, the sweep of the wedge W under consideration is usually stopped by w_l and w_r, not by l or by r. Case M is the predominant case for most wedges, and there we can give a fairly good estimate on the area of the wedge.

We then consider case B, which brings in the effect of the *boundary* of $\mathcal{C}$. At first glance the boundary seems to only make things better, because we can stop searching in a wedge once we cross the boundary. However, it can also make searches longer: if the wedge W_l is cut off and contains no points, the adjacent wedge W (which can be quite large and therefore contain a large number of points) will not be bounded by any point in the empty wedge. The approach of keeping track of the leftmost and rightmost point in each wedge provides an alternate bound in this case. We apply rather crude estimations, and we can show that the area is still $O(1/n)$.

5.1 Case M: Area Bounds Based on Adjacent Wedges

As mentioned above, the bound that will usually allow us to stop searching in the wedge W is the one that says we need look no farther than $2\max(|p, w_l|, |p, w_r|)$. In this section we analyze the expected number of points that we will have to consider in searching a wedge W assuming that the boundary of $\mathcal{C}$ does not interfere.

Thus our problem is reduced to finding the expected area of $S(W, 2|p, w_l|)$, which is four times the area of $S(W_l, |p, w|)$. Formally, we define a random variable A_1 which is equal to this area $(4\gamma t_l^2)$ if the arc with radius t_l does not intersect the boundary of $\mathcal{C}$ inside W_l (case M). The radius $V_1 = 2t_l$ is an upper bound on the distance v at which the sweep of W stops. If case M does not hold we define A_1 and V_1 to be 0.

Clearly, if $V_1 = 2t_l$ is to be bigger than some threshold v, the sector $S(W_l, v/2)$ of half the radius must be empty, because otherwise the point w_l would be in this sector and t_l would be at most v. Therefore we have

$$\Pr[A_1 \geq u] = \Pr[\text{ the sector } S(W_l, \sqrt{u/(4\gamma)}) \text{ of area } u/4 \text{ is empty }] = (1 - u/4)^{n-1}.$$

This equation holds when the sector $S(W_l, \sqrt{u/(4\gamma)})$ is contained in $\mathcal{C}$; otherwise we have $\Pr[A_1 \geq u] = 0$. In any case the right side is an upper bound for $\Pr[A_1 \geq u]$. Thus we get the following bounds for the expected values of A_1 and V_1.

Lemma 2 *The expected value of the area A_1 of the part of the wedge W which is swept in case M is at most $4/n$. The expected radius V_1 of this sector is at most $\sqrt{s/(n-1)}$.*

Proof.

$$E(A_1) = \int_{u=0}^{1} \Pr[A_1 \geq u]\, du \leq \int_{u=0}^{1} (1-u/4)^{n-1} du = 4/n$$

For the radius V_1, which is related to the area by the formula $A_1 = V_1^2 \cdot 2\pi/s$, we have

$$E(V_1) = \int_{v=0}^{\infty} \Pr[V_1 \geq v]\, dv = \int_{v=0}^{\infty} \Pr[A_1 \geq v^2 \cdot 2\pi/s]\, dv \leq \int_{v=0}^{\infty} (1 - v^2 \cdot \pi/(2s))^{n-1}\, dv$$
$$\leq \int_{v=0}^{\infty} e^{-2\pi(n-1)/(2s)\cdot v^2}\, dv = \sqrt{\frac{s}{n-1}}. \qquad \square$$

5.2 Case B: Area Bounds Based on Leftmost and Rightmost Points

Now we consider the case where the arc sweeping a wedge W_l reaches the boundary before distance t_l. This includes the case where the sweep of W_l still finds a point w_l at a later time, and the distance of that point is used as t_l to bound the sweep of W. Typically, however, the sweep of W is stopped by the leftmost point l in W, i. e., $t_l = |p, l|$. Intuitively, we rotate a ray around p, starting from the left edge of W_l, rotating clockwise until we hit l. The distance $|p, l|$ determines the radius of the sector $S(W, 2|p, l|)$ in W which we have to sweep. As in the previous section, we would like to relate its area to the area over which the ray has swept before hitting l, which must necessarily be empty. Now, if p is close to the boundary of $\mathcal{C}$, the sector $S(W, 2|p, l|)$ may be very large although W contains very little of C. Our bound on the expected number of generated points will therefore involve the distance $\beta = \beta(p)$ of p from the boundary of $\mathcal{C}$. We will then average of the possible values of β to determine expected area of the sector for a random point p.

Let us bound the expected area of the sector $S(W, 2t_l)$ in case B. Similarly as before, let us denote by V_2 the random variable whose value is $2t_l$, the radius of the sector, and by $A_2 = V_2^2 \cdot 2\pi/s$ its area. If case B does not hold we define A_2 and V_2 to be 0.

To estimate the probability that $V_2 \geq v$ for some threshold v, let f be the leftmost point of $\mathcal{C} \cap W$ that lies at distance v from p (see Figure 2). Denote by f' the midpoint between p and f, and let g be the point on the left edge of W_l whose distance from p is equal to the distance β from p to the boundary of $\mathcal{C}$. Since we assume that case B holds for the wedge W, we can have $V_2 \geq v$ only if $v/2 \geq \beta$.

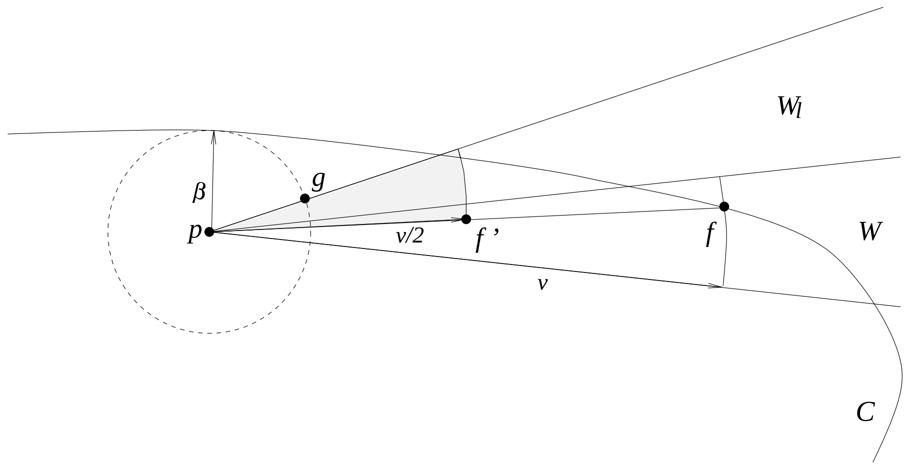

Figure 2: Bounding W

Now, the sweep of W can reach distance v only if the shaded sector of radius $v/2$ between f' and the left edge of W_l contains no points of S. The triangle pgf', whose area is denoted by L, is contained both in this sector, since $|p, g| \leq v/2$, and in $\mathcal{C}$, by convexity. Therefore, the probability $\Pr[V_2 \geq v]$ is at most $(1-L)^{n-1}$. This triangle has a base pf' of length $v/2$. Its height is at least $\beta \sin\gamma$, and so we can bound L as follows.

$$L \geq \frac{1}{2} \cdot v/2 \cdot \beta \sin\gamma = cv\beta$$

with the constant $c = (\sin\gamma)/4 \geq 0.05$. So we get

$$\Pr[V_2 \geq v] \leq (1-L)^{n-1} \leq e^{-L(n-1)} \leq e^{-cv\beta(n-1)}.$$

Lemma 3 *For a point p with distance β from the boundary of $\mathcal{C}$, the expected value of the threshold distance $V_2 = 2t_l$ in case B is at most $1/(c\beta(n-1))$. The expected value of the area A_2 of the corresponding part of the wedge W which is swept is at most $O(1/(\beta n)^2)$.*

Proof.

$$E(V_2) = \int_{v=0}^{\infty} \Pr[V_2 \geq v]\, dv \leq \int_{v=0}^{\infty} e^{-cv\beta(n-1)}\, dv = \frac{1}{c\beta(n-1)}.$$

Let us now bound the expected area $A_2 = V_2^2 \cdot 2\pi/s$ of the sector $S(W, V_2)$.

$$\begin{aligned} E(A_2) &= \int_{a=0}^{\infty} \Pr[A_2 \geq a]\, da = \int_{a=0}^{\infty} \Pr[V_2^2 \cdot 2\pi/s \geq a]\, da = \int_{x=0}^{\infty} \Pr[V_2^2 \cdot 2\pi/s \geq x^2] \cdot 2x\, dx \\ &= \int_{x=0}^{\infty} \Pr[V_2 \geq x\sqrt{s/(2\pi)}] \cdot 2x\, dx \leq \int_{x=0}^{\infty} 2x\, e^{-cx\beta(n-1)\sqrt{s/(2\pi)}}\, dx = \frac{4\pi}{c^2 s\beta^2(n-1)^2} \qquad \square \end{aligned}$$

The above expression still depends on the distance $\beta = \beta(p)$ from the boundary. So we integrate over all possible values of β, using an upper bound of U on the density function of β for a random point p. (In other words, the "probability" that a point is at distance β from the boundary of $\mathcal{C}$ is at most $U\, d\beta$.) Thus we get the following bound on the total number of point pairs that are generated for a random point p.

Lemma 4 *For a random point p, the expected value of the threshold distance $V_2 = 2t_l$ in case B is at most $O((\log n)/n)$. The expected value of the area A_2 of the corresponding part of the wedge W is at most $O(1/n)$.*

Proof. The distance β is bounded by $U/(2\pi)$, and V_2 can be at most $U/2$. Thus, the previous lemma gives

$$\begin{aligned} E(V_2) &\leq \int_{\beta=0}^{U/(2\pi)} \min\left(\frac{1}{c(n-1)\beta}, \frac{U}{2}\right) U\, d\beta = \frac{2}{cU(n-1)} \cdot \frac{U^2}{2} + \frac{U}{c(n-1)} \cdot \int_{\beta=2/(cU(n-1))}^{U/(2\pi)} \frac{d\beta}{\beta} \\ &= \frac{1}{c(n-1)} + \frac{U}{c(n-1)} \cdot \ln\frac{cU^2(n-1)}{4\pi} = O\Big(\frac{\log n}{n}\Big) \end{aligned}$$

From the bound on V_2 we conclude that the area is bounded from above by $(U/2)^2\gamma = U^2\pi/(2s)$. Thus, the result of the previous lemma gives

$$\begin{aligned} E(A_2) &\leq \int_{\beta=0}^{\infty} \min\left(\frac{4\pi}{c^2 s\beta^2(n-1)^2}, \frac{U^2\pi}{2s}\right) U\, d\beta \\ &= \frac{\sqrt{8}}{cU(n-1)} \cdot \frac{U^3\pi}{2s} + \frac{4\pi U}{c^2 s(n-1)^2} \cdot \int_{\beta=\sqrt{8}/(cU(n-1))}^{\infty} \frac{d\beta}{\beta^2} \\ &= \frac{\sqrt{2}\pi U^2}{cs(n-1)} + \frac{4\pi U}{c^2 s(n-1)^2} \cdot \frac{cU(n-1)}{\sqrt{8}} = \frac{\sqrt{2}\pi U^2}{cs(n-1)} + \frac{\sqrt{2}\pi U^2}{cs(n-1)} = O(1/n). \qquad \square \end{aligned}$$

5.3 The Total Number of Pairs Generated

The area of a sector $S(W, t_l)$ can simply be written as $A_1 + A_2$, to account for the two cases. We have therefore established above that, for a wedge W emanating from a random point p, the expected area of $S(W, t_l)$ is $O(1/n)$.

We have to multiply this area by $2s$ to account for every wedge W and its two adjacent wedges W_l and W_r, and by $n-1$ to get an upper bound on the expected number of new points. The resulting expression must be multiplied by n to get the expected overall sum over all points p.

Thus we have the following theorem.

Theorem 1 *The expected number of candidate pairs that the algorithm generates is $O(n)$.* $\square$

Based on some considerations which are partially heuristic, we shall now derive a more accurate estimate of the number of candidate pairs.

For points p in the middle, case B should be very unlikely; but even if the point p lies at the boundary of $\mathcal{C}$, the analysis of Lemma 3, and hence of Lemma 4, overestimates the area in which points for candidate pairs are found by some factor of the order s: Among the s wedges surrounding p only the two wedges which contain a large part of the boundary of $\mathcal{C}$ are cases where the area bound used in the proof of Lemma 3 is tight (see Figure 2). Therefore, by looking at the constants in the lemmas one can see that the contribution of case B to the number of generated pairs should be rather small in comparison to case M.

Concentrating thus on Lemma 2, we know that the expected number of points in a sector is bounded by $(n-1)\cdot E(A_1) \approx 4$. This is a very precise estimate, because the only error that we commit is that A_1 is the area of the whole sector, even if part of it lies outside $\mathcal{C}$. In case M this should be negligible. Since we create $2s$ bounding sectors, the expected number of points in their union is $8s$. Actually, each sector gets covered up to two times in this analysis, when once is enough. Therefore a more accurate estimate of $6s$ can be shown: We have to consider the maximum of two areas which are distributed like A_1 instead of their sum. The overall number of pairs generated can therefore be estimated as $6sn$. For our choice of $s = 30$ this is $180n$, a factor of 60 times the number of edges actually in the triangulation. Because many edges are generated twice (once from each endpoint), this figure over-estimates of the number of generated pairs by a factor at most 2.

All these considerations hold for the algorithm as it is described in section 4.1. With the bucketing technique, we generate more pairs, but it is difficult to estimate the precise effect of this.

5.4 Area Bound for Searching the Closest Point

Before searching each wedge up to its threshold in order to find candidate pairs, we must search every wedge to find its closest point w (step 1 of the algorithm). In practice these two searches are not carried out separately, but we have to account for the fact that we always have to search out to the first point w in W, even if the bounds from the adjacent regions would tell us to stop before we found w, and wp will not be generated as a candidate edge.

The analysis is similar to case M, except that we now use the points in W itself to bound the search in W, not one of the adjacent wedges. Thus, let A_3 be the area of $S(W, |p,w|) \cap \mathcal{C}$ which is searched until w is found. In contrast to the previous analyses, we do not take the full sector but only the part which belongs to $\mathcal{C}$. If W is empty we take the whole area of $W \cap \mathcal{C}$. Similarly as in section 5.1, we have

$$\Pr[A_3 \geq u] = \Pr[\text{ the region } S(W, \sqrt{u/\gamma}) \cap \mathcal{C} \text{ of area } u \text{ is empty }] = (1-u)^{n-1},$$

when u is less than the maximum area of $W \cap \mathcal{C}$, and $\Pr[A_3 \geq u] = 0$ otherwise.

Lemma 5 *The expected value of the area A_3 of the part of the wedge W which is searched until the closest point w is found is at most $1/n$. The expected value of the corresponding radius V_3 is $O(1/\sqrt{n})$.*

Proof. The area bound follows from the analogous calculation as in the proof of Lemma 2. For the radius, we make a similar case distinction as above. In the case when w is found before the sweeping arc reaches the boundary of $\mathcal{C}$, V_3 is equal to $V_1/2$ from Lemma 2, and therefore its expected value is $O(1/\sqrt{n})$. Otherwise, we use a similar argument as in case B to prove a statement analogous to Lemma 3. Assume that $V_3 \geq v$, where $v \geq \beta$. Let f any point of $\mathcal{C} \cap W$ that lies at distance v from p (see Figure 3). Let g and g' be the two points at distance β on the edges of W. Then the quadrilateral $pgfg'$ must be empty. Its area is

$$L' = 1/2 \cdot v \cdot (\sin \angle gpf + \sin \angle g'pf) \geq v/2 \cdot \sin \gamma,$$

which is twice the empty area L which was used to bound V_2 in Lemma 3. Thus we can conclude as in Lemmas 3 and 4 that $E(V_3) = O((\log n)/n)$ ◻

5.5 The Number of Buckets Searched

In the above analyses, we have shown how to restrict each wedge W to a subset W' that must be examined in order to find a superset of all plausible pairs.

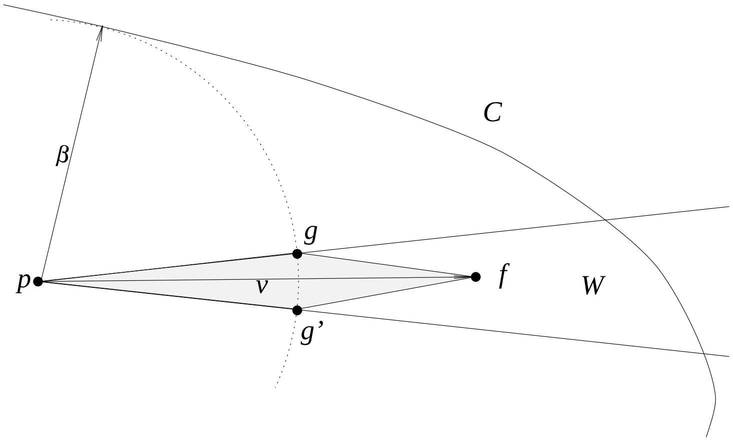

Figure 3: Bounding the area that is examined in searching for w

We have shown that the total area of all the W' regions is $O(1)$ and that the number of points found is therefore $O(n)$, if we follow the original approach of section 4.1. In addition, the total area searched in order to find the closest points w in every wedge is also $O(1)$ (Lemma 5). However, this does not immediately imply that the work done is $O(n)$, because we are using bucketing to find the points. Since a square may overlap W' only partially, the number of squares examined in searching all of the W' regions is not necessarily proportional to the total area.

In addition, we are somewhat sloppy in closing off wedges, delaying the decision whether to close a wedge until a whole round of squares is completed.

We treat both problems by enlarging every sector $S = S(W, v)$ to its *outer parallel body* S_ε that includes all points which are withing a distance ε of S. If we choose $\varepsilon = (3+\sqrt{5})/(2\sqrt{n}) \le 3/\sqrt{n}$, we are sure that every square that the algorithm looks at is contained in S_ε. This figure comes about as follows. When we process a square that intersects the circle C of radius $i/\sqrt{n}$ around p, we are only sure that the wedge should have remained active after the previous round, i. e., for distance $v = (i-1)/\sqrt{n}$. This delay contributes $1/\sqrt{n}$ to ε. In addition, we know only that the square intersects both the circle C and the wedge W. A point in the square can have distance up to $(1/2+\sqrt{5/4})/\sqrt{n}$ from $C \cap W$. This is the second contribution to ε.

The area of S_ε can be written as

$$\text{area}(S_\varepsilon) = \text{area}(S) + \varepsilon \cdot \text{perimeter}(S) + \varepsilon^2\pi \le A + \sqrt{3/n} \cdot V \cdot (2+\gamma) + 9\pi/n,$$

denoting by A the area of the sector S and by V its radius. The expected area $A = A_1 + A_2 = O(1/n)$ has already been considered in the previous section, and the sum of all areas has been shown to be $O(1)$. Thus the expected number of intersected squares (of area $1/n$) to which this term contributes is $O(n)$, the same as the expected number of points covered.

The second term, the expected radius $V = V_1 + V_2$, was also analyzed to be $O(\sqrt{1/n}) + O((\log n)/n) = O(\sqrt{1/n})$, and since this is multiplied by $\sqrt{3/n}$, this does not exceed the bound of $O(1/n)$ for the area of S_ε. The third term is also $O(1/n)$.

Thus we have shown that we may include all points in the enlarged wedges S_ε without destroying the validity of Theorem 1.

6 The Sorting Algorithm

We now must sort the $O(n)$ point pairs by their distances in linear time. We do this using bucket sort.

If we apply the function $g(x) = x^2\pi$ to the distance $|p, q|$ of a random point q from a given point p, the resulting random variable has probability density at most one. In fact, the density equals one as long as the circle with radius x around p is contained in $\mathcal{C}$, and it falls off for larger x. Thus, even if we take all $\binom{n}{2}$ pairwise distances, square them and put them into buckets of size $1/(\pi\binom{n}{2})$, the average number of elements per bucket is less than one. Since the maximum possible distance is $U/2$, the number of buckets is at most $\lceil U^2n^2\pi/8 \rceil$.

So we can first sort the $O(n)$ edge lengths into buckets using a two-stage radix sort on bucket indices, using $2 \cdot O(n + Un) = O(n)$ time and $O(Un)$ space. After that, we can apply any simple quadratic sorting method to sort the elements in each bucket, and maintain the $O(n)$ expected time bound.

7 Summary and Open Problems

We have shown how to modify Dickerson, Drysdale, McElfresh, and Welzl's algorithm [7] to generate a greedy triangulation of uniformly distributed points in expected linear time and space. Our improvements were to generate the plausible edges in linear time and to show that the edge lengths could be sorted in expected linear time. In the process we proved a necessary condition involving wedges that is an extension of earlier lemmas.

This approach also seems promising for non-uniform distributions. The correctness of the basic idea of searching out from a given point using wedges does not depend on the distribution. The bucketing schemes and time bounds do. If the distribution is known, the bucketing could take advantage of the known distribution. If not, then perhaps some sort of sampling or bucketing via quad trees or k-d trees would allow the points in the wedges to be found quickly on the average. It is also possible that some variation of searching on the Delaunay triangulation, as is done in papers by Dickerson, Drysdale, and Sack [6, 8], could prove useful. It seems likely that this sort of algorithm could work quite well for relatively smooth distributions.

Some obvious questions arise from this work.

Problem 1. What are reasonable ways to adapt this algorithm to other distributions? Ideally the algorithm would adapt to any distribution and run in $O(n \log^k n)$ time for some small k as long as the distribution had some nice properties. One possible property is that the probability density is a quasi-concave function, i. e., along any segment it is minimized at an endpoint. The symmetric two-dimensional normal distribution is a good candidate. The algorithm at least can be applied almost without change; only the sorting step may have to be adapted. The analysis will be different, since there is no clear distinction between the cases M and B.

Problem 2. It would be more elegant and probably faster to use a single-stage bucket sort. We would need to know some approximation of the cumulative distribution function of the edge lengths of the *generated* edges. We were able to compute this for the edges generated in case M, whereas for the points close to the boundary this seems very difficult to do. We would need some estimations which would show that the influence of the "boundary edges", which tend to be longer than the other edges, is limited. For example, if we could show that the number of edges generated under case B were bounded by $O(\sqrt{n})$, they could not foul up our analysis even if they fell all in the same bucket.

Problem 3. What is the true worst-case ratio for δ in Lemma 1? We have bounded it between $|p,q|/(2\sqrt{5})$ and $\sqrt{2/17}\ |p,q|$. Note that in this paper we only use a circular sector with radius $|p,q|/2$ centered at p as an exclusion region. Therefore we can't apply the example for the upper bound mentioned after Lemma 1 for our exclusion region. But a very similar example gives approximately $43.87°$ for the angle apq of the circular sector (cf. Corollary 1) showing that we need at least 17 wedges. A more accurate computation for Corollary 1 gives approximately $25.84°$ for the angle apq, thus 28 wedges are enough (rather than 30). What is the true worst case for the angle of the circular sector, i.e., the lowest number of wedges?

References

[1] A. Aggarwal and L. J. Guibas and J. Saxe and P. Shor, "A linear time algorithm for computing the Voronoi diagram of a convex polygon ." *Discrete Comput. Geom.* **4** (1989) 591–604.

[2] R. C. Chang and R. C. T. Lee, "On the average length of Delaunay triangulations." *BIT* **24** (1984) 269–213.

[3] P. L. Chew, "Constrained Delaunay triangulations." *Proceedings of the Third Annual Symposium on Computational Geometry* (1987) 215–222.

[4] G. Das and D. Joseph, "Which Triangulations Approximate the Complete Graph?", *Optimal Algorithms, International Symposium Proceedings*, Lecture Notes in Computer Science **401**, Springer-Verlag, Berlin 1989, pp. 168–192.

[5] F. Dehne, B. Flach, J.-R. Sack, and N. Valiveti, "Analog Parallel Computational Geometry." *Proceedings of the 4th Canadian Conference on Computational Geometry* (1992) 143–153.

[6] M. Dickerson and R. S. Drysdale, "Fixed-radius near neighbors search algorithms for points and segments." *Information Processing Letters* **35** (1990) 269–273.

[7] M. Dickerson, R. L. Drysdale, S. McElfresh, and E. Welzl, "Fast greedy triangulation algorithms." *Proc. 10th Annual Symposium on Computational Geometry* (1994) 211-220.

[8] M. Dickerson, R. L. Drysdale, and J.-R. Sack, "Simple Algorithms for enumerating interpoint distances and finding k nearest neighbors." *International Journal of Computational Geometry and Applications* **3** (1992) 221–239.

[9] R. D. Düppe and H. J. Gottschalk, "Automatische Interpolation von Isolinien bei willkürlich verteilten Stützpunkten." *Allgemeine Vermessungsnachrichten* **77** (1970) 423–426.

[10] P. Gilbert, "New results in planar triangulations." MS Thesis, University of Illinois, Urbana, IL, 1979.

[11] S. Goldman, "A Space Efficient Greedy Triangulation Algorithm." *Information Processing Letters* **31** (1989) 191–196.

[12] D. Kirkpatrick, "A note on Delaunay and optimal triangulations." *Information Processing Letters* **10** (1980) 127–128.

[13] G. Klincsek, "Minimal triangulations of polygonal domains." *Ann. Discrete Math.* **9** 121–123.

[14] D. T. Lee and A. K. Lin, "Generalized Delaunay triangulations for planar graphs." *Discrete Comput. Geom.* **1** (1986) 201–217.

[15] C. Levcopoulos, "An $\Omega(\sqrt{n})$ Lower Bound for the Nonoptimality of the Greedy Triangulation." *Information Processing Letters* **25** (1987) 247–251.

[16] C. Levcopoulos and A. Lingas, "On Approximating Behavior of the Greedy Triangulation for Convex Polygons." *Algorithmica* **2** (1987) 175–193.

[17] C. Levcopoulos and A. Lingas, "Fast Algorithms for Greedy Triangulation." *BIT* **32** (1992) 280–296.

[18] C. Levcopoulos and A. Lingas, "Greedy Triangulation Approximates the Minimum Weight Triangulation and Can be Computed in Linear Time in the Average Case." Tech. Report LU-CS-TR:92-105, Dept. of Computer Science, Lund University, 1992.
A preliminary version of this report appeared in *Advances in Computing and Information* - Proceedings of the 3rd International Conference on Computing and Information, (ICCI'91), Ottawa, May 1991, ed. F. Dehne, F. Fiala, W. W. Koczkodaj, Lecture Notes in Computer Science 497, Springer-Verlag (1991) 139-148.

[19] A. Lingas, "On Approximating Behavior and Implementation of the Greedy Triangulation for Convex Planar Point Sets." *Proceedings of the Second Annual Symposium on Computational Geometry* (1986) 72–79.

[20] A. Lingas, "Greedy triangulation can be efficiently implemented in the average case." Proceedings of the International Workshop on *Graph-Theoretic Concepts in Computer Science* (WG'88), Amsterdam, June 1988, ed. J. van Leeuwen. Lecture Notes in Computer Science 344, Springer-Verlag (1989) 253–261.

[21] E. L. Lloyd, "On triangulations of a set of points in the plane." *Proceedings of the 18th IEEE Symp. Foundations of Computer Science* (1977) 228–240.

[22] G. Manacher and A. Zobrist, "Neither the greedy nor the Delaunay triangulation of the planar set approximates the optimal triangulation." *Information Processing Letters* **9** (1979) 31–34.

[23] G. Manacher and A. Zobrist, "Probabilistic methods with heaps for fast-average-case greedy algorithms." *Advances in Computing Research* vol. 1 (1983) 261–278.

[24] F. Preparata and M. Shamos, "Computational Geometry: an Introduction." Springer-Verlag, 1985.

[25] C. A. Wang, "Efficiently updating the constrained Delaunay triangulations." *BIT* **33** (1993) 238–252.

[26] Y. F. Wang and J. K. Aggarwal, "Surface reconstruction and representation of 3-D scenes." *Pattern Recognition* **19** (1986) 197–207.

[27] C. A. Wang and L. Schubert, "An optimal algorithm for constructing the Delaunay triangulation of a set of line segment." *Proceedings of the Third Annual Symposium on Computational Geometry* (1987) 223–232.

[28] P. Yoeli, "Compilation of data for computer-assisted relief cartography." In *Display and Analysis of Spatial Data*, J. C. Davis and M. J. McCullagh, editors, John Wiley & Sons, NY (1975).

Appendix: Proof of Lemma 1

For notational convenience, rotate the plane so that segment pq is horizontal, with p on the left side. (For the remainder of the proof, refer to Figure 4.) Let a be the point in S closest to pq in the upper exclusion region and let b be the point in S closest to pq in the lower exclusion region. Let C be the circle with pq as a diameter, C_p be the circle of radius $|p,q|$ centered at p, and C_q be the circle of radius $|p,q|$ centered at q. Let $\alpha = \sqrt{|p,q|^2 - 4\delta^2}$. The quantity α was chosen to make the following true:

Observation 1 Any segment that intersects ab, does not intersect the interior of pq, and has both endpoints on or outside of C is at least α long.

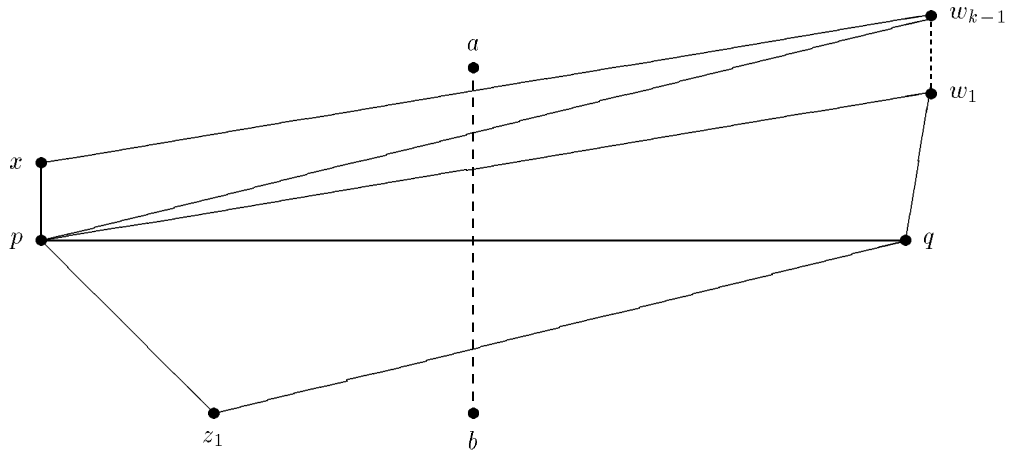

Figure 4: Edges intersecting ab

We will assume that pq is a GT edge and show that this leads to a contradiction. We will use the following observation.

Observation 2 Let p_1 and p_2 be any pair of points that are not connected by an edge in the GT. Then some GT edge intersecting segment p_1p_2 must have length $\leq |p_1, p_2|$.

If pq is a GT edge, then ab cannot be a GT edge because they would intersect. Therefore, by Observation 2 there must be some GT edge of length no longer than $|a, b|$ "cutting off" ab. We will show that that no such edge exists either above or below pq.

Since there are points on both sides of edge pq, it is not a CH edge, and therefore it must be an edge in two triangles – an upper and a lower. Let pw_1q be the upper triangle and let pz_1q be the lower triangle.

For notational convenience, we label all the GT edges intersecting segment ab. The GT edges above pq that intersect ab we label $e_1, \ldots, e_m$, with the edge whose intersection with ab lies nearest to pq labeled e_1, the next nearest labeled e_2, etc. The edges below pq that intersect ab we similarly label $f_1, \ldots, f_n$. Our proof will be a case analysis on the edges $e_1, \ldots, e_m$ and $f_1, \ldots, f_n$.

Since the GT is a triangulation, e_1 shares an endpoint with pq, each successive pair e_ie_{i+1} shares a common endpoint, the non-shared endpoints are connected by an edge, and a is connected to both endpoints of e_m. We label the new endpoint introduced by edge e_i as w_i. We similarly label the new endpoint introduced by f_i as z_i.

We first make a series of observations about the ways that these edges can be arranged. These observations will be stated in terms of the e_i and will assume without loss of generality that pw_1 (instead of qw_1) intersects ab. However, these observations are also true when e is replaced by f and a is replaced by b. They also remain true when p and q are interchanged.

Observation 3 If all of the e_i have both endpoints on or outside of C, then each e_i is of the form pw_i and $|a,q| \geq \alpha$.

To see this, we first note that the segment aq intersects precisely these e_i. Each edge is at least α long, by Observation 1. To prevent aq from being an edge that cuts these edges off, $|a,q| \geq \alpha$ by Observation 2. If some e_i has a left endpoint different than p, then by similar reasoning $|a,p| \geq \alpha$. But a cannot simultaneously lie at least α away from both p and q and remain in C.

Observation 4 If w_1 is on or outside of C and e_k is the lowest-numbered e_i with an endpoint x inside of C, then $|x,q| \geq \alpha$.

All of the e_i with $1 \leq i \leq k-1$ must have both endpoints on or outside of C, so must be at least α long by Observation 1. But these are precisely the GT edges intersecting xq (each completely crosses C). Therefore by Observation 2 one of them must be no longer than xq, implying that $|x,q| \geq \alpha$.

Observation 5 $|p,w_1| < |p,q|$.

Assume that $|p,w_1| \geq |p,q|$. Then w_1 lies on or outside of C_p, and therefore outside of C. Assume that there exists an e_i with either a left endpoint different from p or a right endpoint inside of C_p. Call the lowest numbered such edge e_k and let that endpoint be x. Note that this new point will connect to p and w_{k-1} to form a new triangle. All edges e_i with $1 \leq i \leq k-1$ must be at least $|p,q|$ long, so $|x,q| \geq |p,q|$ to prevent xq from cutting them all off. This is not possible if e_k is of the form px, so e_k must be of the form xw_{k-1} and x must lie outside of C to the left. This edge has both endpoints outside of C and has neither p nor q as one of its endpoints. But then no e_i can ever have an endpoint w_i in C. Otherwise the lowest-numbered such w_j would have to lie at least α away from both p and q to avoid cutting off edges e_i with $k \leq i \leq j-1$. No point in C is distance at least α away from both p and q.

But if no such e_k exists, all edges e_i have left endpoint p and right endpoint outside of C_p, so are at least $|p,q|$ long. But that is impossible, because $|a,q| < |p,q|$, so aq would cut them all off.

The Contradiction This structure is very constraining. If w_1 and z_1 are both in C, then $|w_1,z_1| < |p,q|$ and the GT has chosen the longer diagonal from a convex quadrilateral. This is impossible, so one of the points must lie on or outside of C. Without loss of generality we assume that w_1 is on or outside of C. It must lie on or below the upper tangent ray $p\vec{p}_u$. In addition, it must lie within C_p by Observation 5. This leaves two cases to consider. (See Figure 4 again.)

We first consider the case where none of the e_i has an endpoint within C. Then every e_i is at least α long by Observation 1. By Observation 3 a must lie at least α away from q.

In the second case some e_i has an point x inside of C. By Observation 4 it must lie at distance at least α from q. This point x must also lie below the upper tangent ray $q\vec{q}_u$. Note that the region consisting of possible locations for x contains the region consisting of possible locations for a in the previous case.

We now ask where z_1 can go. It must lie strictly inside the lune formed by the intersection of C_p and C_q by Observation 5. This also implies that segments z_1w_1, z_1x, and z_1a must all intersect pq. It must lie above at least one of the rays $p\vec{p}_l$ and $q\vec{q}_l$. It must lie at least $|p,q|$ away from w_1 (only pq can intervene). It must lie at least α away from either a (case 1) or x (case 2), because all intervening edges are of length at least α. (All have both endpoints on or outside of C.)

But this is not possible. If z_1 is in the right half of C, it can be at most α away from w_1. (The distance between the intersections of $p\vec{p}_u$ and $p\vec{p}_l$ with C_p is exactly α, but in this case we require the much larger separation of $|p,q|$.) If it lies in the left half of C, the actual maximum distance will be the distance between the intersection of $q\vec{q}_u$ with C and the intersection of $q\vec{q}_l$ with C_q, and this distance is about $.95\alpha$. Therefore we could increase the size of δ slightly, but only at the expense of greatly complicating its formula. $\square$