How To Place a Point to Maximize Angles

Boris Aronov* aronov@poly.edu
Mark V. Yagnatinsky† myag@cis.poly.edu

Polytechnic Institute of NYU, Brooklyn, New York

Abstract

We describe a randomized algorithm that, given a set of points in the plane, computes the best location to insert a new point, such that the Delaunay triangulation of the resulting point set has the largest possible minimum angle. The expected running time of our algorithm is at most cubic on any input, improving the roughly quartic time of the best previously known algorithm.

1 Introduction

The subject of meshing and specifically constructing "well behaved" triangulations has been researched extensively [B04]. One of the problems extensively addressed in the literature is that of refining or improving an existing mesh by incremental means. Motivated by this, Aronov *et al.* [AAF10] considered the following problem: Given a set of points in the plane, where would you place one additional point, so as to maximize the smallest angle in a good triangulation of the point set. Since Delaunay triangulations are known to maximize the smallest angle over all possible triangulations with a given vertex set [S78], the question can be rephrased as: "Given a point set, where do we place an additional point, so as to maximize the minimum angle in the Delaunay triangulation of the resulting set?" In the rest of the paper we always picture the new point as lying within the convex hull of the existing points, but the algorithm is essentially the same without this simplification. (Another variant of the problem mentioned in [AAF10] involved incrementally improving an existing triangulation by "tweaking" the position of an existing interior vertex, one at a time, so that, again, the smallest angle is maximized.) They also discuss the more challenging question of how to position several points in the best possible coordinated way; we do not address this variant of the problem in the current paper.

The previous algorithm [AAF10] for placing an additional point runs in worst-case $O(n^{4+\varepsilon})$ time, for any $\varepsilon > 0$, with the constant of proportionality depending on ε. We propose a randomized algorithm whose expected running time is roughly an order of magnitude lower. Somewhat surprisingly, Aronov *et al.* considered and rejected the approach we use in this paper [AAF10, page 96].

We present our algorithm in the following section. The analyses of this and the precursor algorithm [AAF10] are misleading in that they reflect situations unlikely to happen for "reasonable" inputs. We discuss how to quantify reasonableness of the inputs and the resulting behavior of both algorithms in section 3 and conclude in section 4.

2 The Algorithm

Our algorithm takes a set P of n points in the plane and computes the best location for a new point p, such that the Delaunay triangulation of $P \cup \{p\}$ has the largest possible minimum angle; for ease of presentation we will assume that the points of P are in *general position*, that is no three points of P lie on a line and no four on a circle. We start by recalling an argument detailed in [AAF10] which duplicates the insertion step of the standard incremental Delaunay triangulation algorithm [GS85]. Let T be the Delaunay triangulation of P. We begin by computing the arrangement $\mathcal{A}$ induced by the Delaunay circles of P, i.e., of the circumcircles of the triangles of T. Although there are only a linear number of such circles, in the worst case every pair of them intersect, so that $\mathcal{A}$ has quadratic complexity. We examine how the Delaunay triangulation T_p of $P \cup \{p\}$ differs from T. Let c be the face of $\mathcal{A}$ containing p. Recall that a triangle is present in a Delaunay triangulation if and only if its Delaunay disk is empty of vertices. Point p *invalidates* some triangles of T by appearing in the interior of the corresponding disks. After we have inserted p, we no longer have a triangulation; instead we have a star-shaped polygonal hole H in T containing p; see Figure 1. Since the insertion of p only invalidates previously valid triangles, but cannot make an invalid triangle valid (since insertion of p can not turn nonempty disks into empty ones), new edges of T_p must have p as an endpoint. So, connecting p to all vertices of H (Figure 1, right) is the way to complete T_p. This suggests this algorithm outline:

1. Compute the Delaunay triangulation T.
2. Build the arrangement $\mathcal{A}$ of Delaunay circles of T.

*Supported by NSF grants CCF-11-17336 and CCF-12-18791.
†Supported by NSF grant CCF-11-17336.

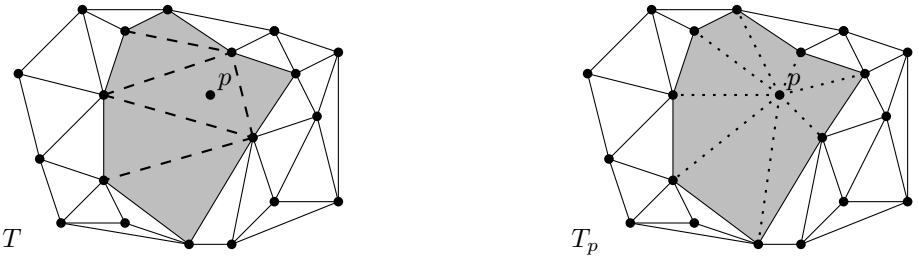

Figure 1: The new point p is in the kernel of the shaded star-shaped polygonal hole H. Removed edges of T are shown dashed (left) and added edges are dotted (right).

3. For each of the $O(n^2)$ cells $c \in \mathcal{A}$:
   (a) Find the set of $O(n)$ triangles invalidated by placing p in c, the union of which forms the hole H.
   (b) Optimize the placement of p in c.
4. Return the best placement of p found.

This outline was in fact used in [AAF10]. The main contribution of this paper is to use a different approach for step 3b. Specifically, in [MSW96], it was shown that the following is an LP-type problem.[1]

> Given a star-shaped polygon H, find the point p in its kernel that maximizes the smallest angle in the triangulation that results by connecting p to all vertices of H.

Being an LP-type problem, it can be solved in expected time linear in the number of polygon vertices, while the approach from [AAF10], based on explicitly computing lower envelopes of bivariate functions, takes time roughly quadratic in the number of vertices. However, this LP-type problem is not quite the problem we actually wish to solve, as we need the optimal placement of p *within the current cell* c, which is why this idea was rejected in [AAF10]. Fortunately, there is a conceptually simple fix. In the *region search* stage of our procedure, for each cell c, we run the algorithm from [MSW96] discarding the result if the returned optimum lies outside c. A simple argument (see Lemma 1 below) shows that if the solution to the unconstrained problem results in a point not in c, then the optimum within c must lie on its boundary. So in a separate *boundary search* step detailed below, we find the best placement of p on any cell boundary. Combining the results from the two steps we obtain the globally optimal placement for p.

Lemma 1 *If the optimum solution to the unconstrained LP-type problem corresponding to cell c is not in c, then the optimum solution for c lies on its boundary.*

Proof. Consider the locus $R(x)$ of points p such that the smallest angle in the new triangulation of H is at least x. It was shown in [MSW96] that $R(x)$ is a convex region; it is easy to see that it varies continuously with x, when non-empty. Clearly, $R(x) \subset R(y)$ for $y < x$. As x decreases from its optimum unconstrained value, $R(x)$ will gradually grow from a single point outside c and eventually intersect c; as it is connected and changes continuously with x, the first intersection must occur along the boundary of c. □

It remains to find the best placement for p on each cell boundary. A cell boundary has two sides, and we process each separately. First consider each edge of $\mathcal{A}$ separately. For a fixed side of a fixed edge e, we know which cell of $\mathcal{A}$ we are in, and thus the hole H. If H has h vertices, the triangulation has $3h$ angles. The measure of each of these angles is a function of the position of p. To maximize the smallest of these functions, find the maximum of their lower envelope by computing the envelope explicitly. We will show that the graphs of any pair of these functions intersect at most 16 times. A well-known result from the theory of Davenport-Schinzel sequences immediately implies that the maximum complexity $E(n)$ of the lower envelope is $O(\lambda_{16}(n))$, which is $o(n \log^* n)$, where $\lambda_s(n)$ is the maximum length of a $\mathrm{DS}(s, n)$ sequence [AS00]. If the worst-case complexity of the lower envelope of h functions from some class is $E(h)$, then we can compute the lower envelope of n functions from that class in $O(E(n) \log n)$ time using a simple divide-and-conquer algorithm [AS00].

Lemma 2 *The complexity of the lower envelope of n angle functions is $O(\lambda_{16}(n))$.*

Proof. There are two kinds of angles to consider: angles at the boundary of H, and angles at the new point p. We consider first angles at p. Let $p = (x, y)$, and let q and r be two consecutive vertices of H. Note that the coordinates of q and r are known constants. We are interested in the angle $\angle qpr$ at which p sees the segment qr; see Figure 2 (left). Let s, t be another pair of consecutive vertices. The angle that p makes with the segment st is $\angle spt$. Now consider the locus of points p specified by the equation $\angle qpr = \angle spt$; a point p

[1] In [ABE99], this and related problems are presented in a unified framework.

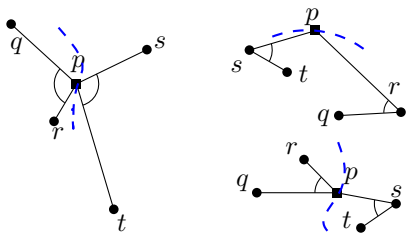

Figure 2: "Artist's impression" of the curves defined by the three types of angle equality constraints.

satisfying this equation will see qr and st at the same angle; refer to Figure 2 (left). An intersection between this curve and an edge of $\mathcal{A}$ corresponds precisely to an intersection of the graphs of two angle functions. Once we prove that there are at most 16 such intersections, we are done. For convenience, we will equate the cosines of the angles instead of the angles themselves. Using $|\cdot|$ to denote distances, the law of cosines gives $|qr|^2 = |pr|^2 + |pq|^2 - 2|pr||pq|\cos\angle qpr$. Solving for $\cos\angle qpr$ gives

$$\cos\angle qpr = \frac{|pr|^2 + |pq|^2 - |qr|^2}{2|pr||pq|}.$$

Setting $\cos\angle qpr$ equal to $\cos\angle spt$ produces

$$\frac{|pr|^2 + |pq|^2 - |qr|^2}{|pr||pq|} = \frac{|ps|^2 + |pt|^2 - |st|^2}{|ps||pt|}.$$

After squaring both sides and reshuffling, we get

$$(|pr|^2 + |pq|^2 - |qr|^2)^2|ps|^2|pt|^2 = (|ps|^2 + |pt|^2 - |st|^2)^2|pr|^2|pq|^2.$$

Now each side of the equation is a polynomial in x and y of total degree eight. So, the question now is: how many times can a curve of degree eight intersect an edge of $\mathcal{A}$? An edge is an arc of a circle, which is the zero set of a polynomial of degree two. According to Bézout's theorem [B1779], the number of intersection points is at most the product of the degrees, so there can be at most 16 intersection points. So, the complexity of the envelope is $O(\lambda_{16}(n))$, and we are done. A similar argument is needed for $\angle qpr = \angle pst$ and also $\angle pqr = \angle pst$ (refer to Figure 2 (right)), but they also result in polynomial equations of degree at most eight; we omit the entirely analogous calculation. □

The approach outlined above is inefficient in that there may be a quadratic number of arcs, and since we spend more than linear time on each, this would become this bottleneck of the algorithm. However, we are duplicating much work: if we follow a Delaunay circle as it crosses another circle, very little changes when we cross: either one triangle of T ceases to be valid, or else one triangle becomes valid. (This assumes that we only cross one circle at at time. At a vertex, we may cross many circles at once, so the total change is large, but it is still true that each circle we cross does only one triangle's worth of damage.) Suppose that a triangle becomes valid when we cross (the other case is symmetric). Then H loses a boundary vertex, and our triangulation of H loses two old triangles and gains one new one, which means our set of angle functions gains 3 new angles and loses 6 old ones. The other angle functions remain unchanged. Thus, we can define the functions over the an entire circle (provided we are consistent whether we are on the inside or outside of the circle.) On a given arc, there are at most $3n$ functions. If there are m circles, then the boundary of a fixed circle can only have $2(m-1) < 2m$ intersections with other circles, and for each of those intersections, at most 6 new functions appear. The number of circles equals the number of triangles, which is less than $2n$. Thus for the entire circle, there are at most a linear number of functions ($2n \times 2 \times 6 + 3n \leq 27n$). It is still the case that any pair of function graphs intersect at most 16 times, but because each is not defined over the entire circle, but only a contiguous arc on it, the complexity of the lower envelope can increase slightly, up to $\lambda_{18}(n)$ [AS00]. Thus, the running time of the boundary search stage is $O(n\lambda_{18}(n)\log n)$ and the total (expected) running time is dominated by the $O(n^3)$ region search time. (The boundary search can be sped up slightly to $O(n\lambda_{17}(n)\log n)$ by using the algorithm of Hershberger [H89].)

3 Realistic inputs

In the long tradition in computational geometry, exemplified by [BKSV02], we would like to be able to analyze our problem in non-worst-case situations. To this end, we introduce several parameters, besides n that measures the number of input points, that quantify the "badness" of the input point set and express the running time of the algorithms in terms of them.

Consider the arrangement $\mathcal{A}$ of Delaunay disks of P and let k be its complexity, that is the total number of vertices, edges, and faces; let d be the maximum *depth* of the arrangement, that is the maximum, over all points in the plane, of the number of disks covering the point. In the worst case k is $\Theta(n^2)$ and d is $\Theta(n)$. In well-behaved point sets, such as those corresponding to uniformly distributed points, k is $\Theta(n)$; one would also expect d to be near-constant, however, somewhat surprisingly, an unfortunate, but arbitrarily small perturbation of the $\sqrt{n} \times \sqrt{n}$ grid can cause d to be $\Theta(\sqrt{n})$ (we omit the details in this version).

We now express the running times in terms of n, k, and d. Our algorithm starts by computing the Delaunay triangulation, which can be done in $O(n\log n)$ time. We then compute the arrangement of circles in $O(k\log n)$

time using a standard sweepline algorithm (better running times are possible using more involved techniques). Our algorithm and that of [AAF10] share the first two steps of the outline. Their analog of the region search runs in time $O(kd^{2+\varepsilon})$, for any positive ε, since for every cell $c \in \mathcal{A}$, it performs an independent bivariate lower envelope calculation on $O(d)$ functions, for a total time of $O(kd^{2+\varepsilon} + k \log n)$. We analyze the region search and the boundary search stages of our proposed algorithm separately. The region search runs in expected time $O(kd)$, as its bottleneck is solving k LP-type problems of size at most d each. (Note that this requires that we quickly determine the set of constraints that correspond to a cell. This is easy to arrange if we visit adjacent cells in order.)

We now turn our attention to the boundary search. Our analysis here needs stronger general position assumptions than the algorithm itself does. In particular, we require that if two circles intersect in some point not in P, no third circle goes through that point.

The running time for one circle is affected by how many functions we need to take the lower envelope of along that circle. We earlier derived a bound of $27n$ for the number of functions on a given circle. We now make this more precise. Let f_i denote the number of functions along circle C_i. If C_i intersects x_i other circles, and further is not adjacent to any cell having depth more than d_i, then by our previous analysis $f_i \leq 3(d_i+1)+12x_i$. Note that since these circles are Delaunay, no disk fully contains another. Hence, any circle containing a cell of large depth must intersect many other circles. In particular, $x_i \geq d_i - 1$. Thus, we have $f_i \leq 3(d_i + 1) + 12x_i \leq 3(x_i + 2) + 12x_i = 15x_i + 6$, which is $O(x_i)$.

We now show that the sum of x_i over all circles is at most proportional to the arrangement complexity k. Note first that this sum is simply twice the number of pairs of intersecting circles. Our approach will thus be to show that most pairs of intersecting circles contribute a vertex of degree 4 to the arrangement $\mathcal{A}$, that is, a vertex that no third circle goes through. Indeed, consider a pair of intersecting circles such that both intersection points, call them p and q, have degree at least 6 (in a circle arrangement, all vertices have even degree). By our stronger general position assumption, both p and q are from the original point set P. We now have a pair of points with two Delaunay circles passing through it: hence pq must be a Delaunay edge! But there are only a linear number of such edges, so we are done: all but $O(n)$ pairs of intersecting circles contribute a new vertex to the arrangement.

Finally, let m be the number of circles, X be the number of pairs of intersecting circles, u be the number of vertices of degree 4, and e be the number of edges of the Delaunay triangulation. We now bound the sum of x_i over all circles: $\sum_{i=1}^{m} x_i = 2X \leq 2(u + e) = 2u + 2e < 2k + 2e \leq 2k + 2(n + m - 2) \leq 2k + 2(m + 2 + m - 2) = 2k + 4m < 2k + 4k = 6k$, which is $O(k)$.

Lastly, the total running time of the boundary search stage is at most proportional to:

$$\begin{aligned}\textstyle\sum_{i=1}^{m} \lambda_{18}(x_i) \log x_i &\leq \textstyle\sum_{i=1}^{m} \lambda_{18}(x_i) \log m \\ &= \log m \cdot \textstyle\sum_{i=1}^{m} \lambda_{18}(x_i) \\ &\leq \log m \cdot \lambda_{18}(\textstyle\sum_{i=1}^{m} x_i) \\ &\leq \lambda_{18}(6k) \log m,\end{aligned}$$

which is $O(\lambda_{18}(k) \log n)$, and thus the running time of our entire algorithm is $O(kd + \lambda_{18}(k) \log n)$. Therefore, our algorithm outperforms (in expectation) that of [AAF10] for all values of k and d.

We can slightly refine the above analysis in another direction: Recall that we defined d to be the *maximum* depth of the arrangement $\mathcal{A}$. If we let $\bar{d}$ be the *average* depth, over all the cells, the running time of the region search can then be bounded by $O(k\bar{d})$, while the running time of the analogous part of algorithm of [AAF10] is $O(\sum_{c \in \mathcal{A}} d_c^{2+\varepsilon})$, where d_c is the depth of cell c; the latter quantity is, roughly, k times the average *squared* depth. The running time of the boundary search is not easily expressed in terms of $\bar{d}$, but it is less likely to dominate the running time of our algorithm.

It would be interesting to connect the parameters d and k (and ultimately the behavior of the algorithms) to the more commonly used measures of how well-behaved a point set in the plane is, such as its *spread*, which is the ratio between the largest and the smallest interpoint distances.

4 Conclusions and open problems

We believe our algorithm can be easily modified to work with constrained Delaunay triangulations, which was the original setting of [AAF10]; we omit the extension in this version. (The key differences are that the set of invalidated triangles depends on the constraints, and that the arrangement $\mathcal{A}$ is formed by circles and the constraining segments.)

It would be interesting to see if our algorithm can be derandomized using the results of Chazelle and Matoušek [CM93]; the LP-type problem needs to meet some technical requirements the discussion of which is omitted here.

Can the algorithm be sped up by roughly another order of magnitude by observing that there is generally very little difference between LP-type problems corresponding to adjacent cells of $\mathcal{A}$?

Is there any hope of generalizing our approach to multiple Steiner points as in [AAF10]?

References

[AS00] P.K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds., 1–47. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[ABE99] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms* 30(2), 302–322, 1999.

[AAF10] B. Aronov, T. Asano, and S. Funke. Optimal triangulations of points and segments with Steiner points. *Int. J. Comput. Geom. Appl.*, 20(1) 89–104, 2010.

[BKSV02] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.

[B04] M. Bern. Triangulations and mesh generation. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O'Rourke, Eds., 563–582. CRC Press LLC, Boca Raton, FL, April 2004.

[B1779] Bézout theorem. Wikipedia. From `http://en.wikipedia.org/wiki/B%C3%A9zout%27s_theorem`; retrieved 11 May 2013.

[CM93] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Proc. Fourth Annu. ACM-SIAM Symp. Discr. Algorithms*, pp. 281–290, 1993.

[GS85] L. 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, 1985.

[H89] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inf. Proc. Letters*, 33(4) 169–174, 1989.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4–5) 498–516, 1996.

[S78] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3) 243–245, 1978.