Optimal Higher Order Delaunay Triangulations of Polygons*

Rodrigo I. Silveira and Marc van Kreveld

Department of Information and Computing Sciences
Utrecht University, 3508 TB Utrecht, The Netherlands
{rodrigo,marc}@cs.uu.nl

Abstract. This paper presents an algorithm to triangulate polygons optimally using order-k Delaunay triangulations, for a number of quality measures. The algorithm uses properties of higher order Delaunay triangulations to improve the $O(n^3)$ running time required for normal triangulations to $O(k^2 n \log k + kn \log n)$ expected time, where n is the number of vertices of the polygon. An extension to polygons with points inside is also presented, allowing to compute an optimal triangulation of a polygon with $h \geq 1$ components inside in $O(kn \log n) + O(k)^{h+2} n$ expected time. Furthermore, through experimental results we show that, in practice, it can be used to triangulate point sets optimally for small values of k. This represents the first practical result on optimization of higher order Delaunay triangulations for $k > 1$.

1 Introduction

One of the best studied topics in computational geometry is the triangulation. When the input is a point set P, it is defined as a subdivision of the plane whose bounded faces are triangles and whose vertices are the points of P. When the input is a polygon, the goal is to decompose it into triangles by drawing diagonals.

Triangulations have applications in a large number of fields, including computer graphics, multivariate analysis, mesh generation, and terrain modeling. Since for a given point set or polygon, many triangulations exist, it is possible to try to find one that is the best according to some criterion that measures some property of the triangulation.

The properties of interest are application-dependent, but are generally either local properties of the triangles (like area, height or minimum angle) or global properties of the triangulation (such as total edge length). For example, in automatic mesh generation for finite element methods, criteria like minimizing the minimum/maximum angle and height of the triangles are related to the error of the finite element approximation [3,21]. In the context of terrain modeling, terrains are many times represented by *triangulated irregular networks*, which are

* This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under the project GOGO.

E.S. Laber et al. (Eds.): LATIN 2008, LNCS 4957, pp. 133–145, 2008.

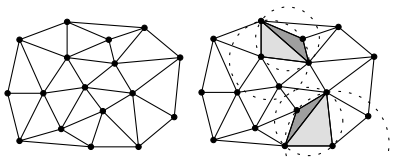

Fig. 1. A Delaunay triangulation ($k = 0$) (left), and an order-2 triangulation (right). Light gray triangles are first order, the medium grey ones are second order.

triangulations where each point has an elevation. When terrain models need to be *realistic*, for example in visualization or terrain analysis for hydrology, criteria like avoiding *ill-shaped* triangles and having few local minima are particularly relevant [18,6].

For a given set of points P, a well-known triangulation is the Delaunay triangulation. It is defined as a triangulation where the circumcircle of the three vertices of any triangle does not contain any other point of P. It is unique when no four points are cocircular, and can be computed in $O(n \log n)$ time for n points. The Delaunay triangulation has a large number of known properties, and it optimizes several measures, like max min angle or min max smallest enclosing circle, among others. This is the reason why its triangles are said to be well-shaped. However, for many applications, the Delaunay triangulation is not flexible enough. For example, when used for terrain modeling, the triangulation does not take the third dimension into account, which may result in artifacts like interrupted valley lines or artificial local minima.

To overcome this limitation, Gudmundsson et al. [11] define *higher order Delaunay triangulations*, a class of well-shaped triangulations where a few points are allowed inside the circumcircles of the triangles. A triangle is said to be *order-k Delaunay* if its circumcircle contains at most k points. A triangulation is *order-k Delaunay* if all its triangles are order-k Delaunay (see Figure 1). For $k = 0$, each non-degenerate point set has only one higher order Delaunay triangulation, equal to the Delaunay one. As the parameter k is increased, more points inside the circumcircles imply a reduction in the shape quality of the triangles, but also an increase in the number of triangulations that are considered, and hence greater flexibility to optimize some other criterion, while limiting the badness of the shape of the triangles. The concept of higher order Delaunay triangulation has been successfully applied to several areas, including terrain modeling [7], minimum interference networks [1] and multivariate splines [20].

In this paper we focus mainly on triangulations of polygons. Optimal polygon triangulation has been a subject of study for a long time, both because it has applications of its own, like in finite element methods, and also because it gives insight into the generally harder problem of optimal point set triangulation. When the goal is to optimize only one criterion, optimal polygon triangulations can be computed in polynomial time for many measures. The constrained Delaunay triangulation [5], which generalizes the standard definition in order to force certain edges into the triangulation, can be used to triangulate polygons,

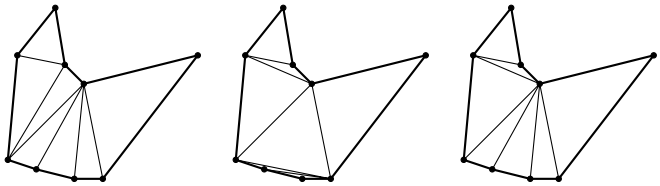

Fig. 2. Three different triangulations of a polygon: the constrained Delaunay triangulation (left), the minimum weight triangulation (center), and the minimum weight triangulation constrained to order-2 Delaunay triangulations (right). The third one combines nicely shaped triangles with the minimization of the weight.

with triangle shape properties similar to the ones of the Delaunay triangulation. Many other measures can be optimized using a dynamic programming algorithm attributed to Klincsek [15], and also, independently, proposed by Gilbert [10]. This approach allows to find in $O(n^3)$ time an optimal triangulation of a simple polygon for any *decomposable* measure. Intuitively, a measure is decomposable if the measure of the whole triangulation can be computed efficiently from the measures of two pieces, together with the information on how the pieces are glued together. See [3] for a formal definition. Decomposable measures include the following: min / max angle, min / max circumcircle, min / max length of an edge, min / max area of triangle, and the sum of the edge lengths. The algorithm by Klincsek can be extended to other measures that are not decomposable, like maximum vertex degree. For convex polygons, the min/max area of triangle measures can be optimized even faster, in $O(n^2 \log n)$ time [14].

Triangulating point sets optimally is in general more difficult than triangulating polygons. For example, the minimum weight triangulation can be computed for polygons in cubic time using Klincsek's algorithm, but is NP-hard for point sets [19]. Only a few methods exist for optimal triangulations of point sets. The edge insertion paradigm [2] can be used to optimize several measures in $O(n^2 \log n)$ or $O(n^3)$ time (depending on the measure). A triangulation of a point set minimizing the maximum edge length can be computed in $O(n^2)$ time [8]. The *greedy triangulation*, which lexicographically minimizes the sorted vector of length edges, can be constructed in $O(n \log n)$ time [17].

Our problem is more involved, since we aim at optimizing a measure over higher order Delaunay triangulations, therefore enforcing well-shaped triangles at the same time as optimizing some other measure. Figure 2 shows an example. There are not many results on optimal higher order Delaunay triangulations. For the case $k = 1$, the triangulations have a special structure that allows a number of measures (for example max/min area triangle, total edge length, number of local minima in a terrain) to be optimized in $O(n \log n)$ time [11]. A few other measures, like minimizing the maximum area ratio of edge-adjacent or vertex-adjacent triangles, can also be optimized efficiently [16]. Other measures like maximizing the number of convex edges or minimizing the maximum vertex degree have been shown to be NP-hard [16]. For $k > 1$, fewer results are known.

Minimizing local minima in a terrain is NP-hard for orders at least n^{ε}, where ε is any positive constant [7].

In this paper we present an extension of the algorithm by Klincsek [15] that allows to optimize a decomposable measure for a simple polygon over order-k Delaunay triangulations. A straightforward extension of Klincsek's algorithm leads to $O(kn^3 + n^3 \log n)$ running time. Our main contribution is improving this to $O(k^2 n \log k + kn \log n)$, by exploiting properties of this special class of triangulations. This represents an important improvement, given that small values of k are most important [7].

We also explain how to extend our algorithm to triangulate polygons with points inside, and present experimental results on the structure of order-k Delaunay triangulations that suggest that in practice, the same approach can be used to triangulate point sets for small values of k optimally. This constitutes the first practical result on optimization of higher order Delaunay triangulations for $k > 1$.

Note that in this paper we use the standard definition of *order* of a triangle, as in [11,20], which does not take the boundary edges of the polygon into account. This implies that for a given polygon and value k, our algorithm may find that no order-k triangulation of that polygon exists. In such a case, it is always possible to increase k until a triangulation is found. The authors recently studied possible definitions of the order of a triangle that take a set of constraining edges into account [22], which guarantee that a polygon can always be triangulated for any k. However, although seven different definitions for this notion of *constrained order* were proposed in [22], no single definition can be regarded as the *natural* or *right* one.

2 Higher Order Delaunay Triangulations

In this section we present some basic concepts on higher order Delaunay triangulations, together with some results that will be needed later. From now on we assume non-degeneracy of the input set P: no four points are cocircular.

Definition 1. *(from [11]) A triangle $\triangle uvw$ in a point set P is* order-k Delaunay *if its circumcircle $C(u, v, w)$ contains at most k points of P. A triangulation of a set P of points is an* order-k Delaunay triangulation *if every triangle of the triangulation is order-k.*

Definition 2. *(from [11]) Let P be a set of points, and let $\overline{pq}$ be an edge between two points p, $q \in P$. $\overline{pq}$ is an* order-k Delaunay edge *if there exists a circle that passes through p and q that has at most k points of P inside. The* useful order *of an edge is the lowest order of a triangulation that includes that edge.*

For brevity, we will sometimes write *order-k* instead of *order-k Delaunay*, and *k-OD edge* instead of *order-k Delaunay edge*. We also assume that $k \geq 1$ is a given integer, and write *useful edge* instead of *useful k-OD edge*. It is worth mentioning that the order and the useful order of an edge can differ arbitrarily much.

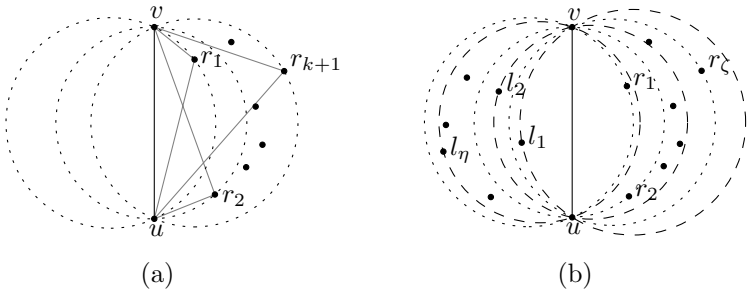

Fig. 3. (a) At most $k+1$ *third points*. (b) Finding the k-*OD* triangles incident to $\overline{uv}$.

Lemma 1. *(from [11]) Let $\overline{uv}$ be a k-OD edge, let s_1 be the point to the left of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left of $\overrightarrow{vu}$. Let s_2 be defined similarly but to the right of $\overrightarrow{vu}$. Edge $\overline{uv}$ is a useful k-OD edge if and only if $\triangle uvs_1$ and $\triangle uvs_2$ are k-OD triangles.*

We extend the basic definitions and lemma above with one more lemma.

Lemma 2. *Let $\overline{uv}$ be a useful k-OD edge. There are $O(k)$ order-k triangles that have $\overline{uv}$ as one of their edges.*

Proof. We will show that $\overrightarrow{uv}$ can be part of at most $k+1$ triangles on each side. Imagine we slide a circle in contact with u and v until it touches a first point r_1 to the right of the edge (see Figure 3(a)). This could potentially be a *third point* of a triangle that includes $\overrightarrow{uv}$ because it is possible for the circumcircle of triangle $\triangle ur_1v$ to contain less than $k+1$ points. If we slide the circle again until it touches a second point r_2, we now know that the circle contains at least one point (r_1). Continuing in this way it can be seen that the circle defined by u, v and the $(k+1)$-th touching point, r_{k+1}, contains at least k points, hence no further point can be a *third point* because then the circle would contain $k+1$ points. Since an identical argument can be applied to the left side, we conclude that at most $O(k)$ triangles can have $\overrightarrow{uv}$ as one of its edges. ⊠

Next we show that all the order-k triangles formed by useful k-*OD* edges can be computed efficiently.

Lemma 3. *Let P be a set of n points in the plane. In $O(k^2 n \log k + kn \log n)$ expected time one can compute all order-k triangles of P that are incident to at least one useful k-OD edge.*

We provide a sketch of the algorithm. An order-k triangle must meet two conditions: its circumcircle must contain at most k points and it must be empty. For a set of n points there are $O(kn)$ useful k-*OD* edges [11], and by Lemma 2, a given useful edge can be part of $O(k)$ order-k triangles. This makes the total number of order-k triangles $O(k^2 n)$.

All the useful edges can be computed in $O(k^2 n + kn \log n)$ expected time [11]. Moreover, without increasing the previous asymptotic running time, we can store

for every useful edge $\overrightarrow{uv}$, the two sets of points that are contained in the two circles that determine its usefulness (see Lemma 1). There are at most k of these points on each side. For each side, we will sort the points according to the order in which they are touched when sliding a circle in contact with u and v (in the same way as in the proof of Lemma 2). This can be done in $O(k \log k)$ time by sorting the centers of the circles.

To find out which *third points* can make an order-k triangle, we need to count the number of points inside each circumcircle. This can be done using the two sorted lists of points as follows. We explain how to do it for the right side, the left side is identical. Let $L = \{l_\eta, ..., l_1\}$ be the sorted points to the left of $\overrightarrow{uv}$, and $R = \{r_1, ..., r_\zeta\}$ the ones to its right. See Figure 3(b). The circumcircle of $\triangle uvr_1$, denoted $C(u, v, r_1)$, contains by definition no points to the right of $\overrightarrow{uv}$ and η points to its left. For the second point, r_2, we know that $C(u, v, r_2)$ contains exactly one point to the right of $\overrightarrow{uv}$ (namely, r_1). To find out how many points it contains to the left of $\overrightarrow{uv}$, we check whether l_η is inside $C(u, v, r_2)$ or not. If it is, then $C(u, v, r_2)$ contains exactly η points to the left of $\overrightarrow{uv}$. If it is outside, we go through L until we find the first l_j that is inside $C(u, v, r_2)$. That implies $C(u, v, r_2)$ contains exactly j points to the left of $\overrightarrow{uv}$. This is then repeated for $r_3, r_4, ..., r_\zeta$. The running time is linear in k because both lists are scanned only once, from left to right. Hence for each useful edge $\overrightarrow{uv}$, we can find part of the triangles incident to $\overrightarrow{uv}$ whose circumcircles contain at most k points in $O(k \log k)$ time.

Notice that this algorithm, when applied to one edge $\overrightarrow{uv}$, does not necessarily find all the order-k triangles adjacent to $\overrightarrow{uv}$. It may happen that some of the order-k triangles adjacent to it have a third point that is not among the points included in the two circles defining the usefulness of $\overrightarrow{uv}$, because these circles contain *at most* k points, but may contain less. However, we show in the full paper that any triangle composed of three useful order-k edges that is missing will be considered when processing one of the other two useful edges that make the triangle, hence no relevant triangle will be missed at the end.

Still, some of these triangles may contain points inside, so we need to discard the ones that are not empty. Let $\triangle uvx$ and $\triangle uvy$ be two triangles, and let α_u (α_v) denote the angle of $\triangle uvx$ at u (at v), and β_u (β_v) the same for $\triangle uvy$. It is easy to see that $\triangle uvx$ contains point y if and only if $\beta_u < \alpha_u$ and $\beta_v < \alpha_v$. Each triangle can be represented by a point in the plane using its angles at u and at v. The empty triangles are the ones lying on the lower-left staircase of the point set, and can be found in $O(k \log k)$ time by a sweep line algorithm.

The total time needed to find the triangles for one useful edge is $O(k \log k)$. Since there are $O(kn)$ useful edges, all the useful edges and order-k triangles can be computed in $O(k^2 n \log k + kn \log n)$ expected time, proving Lemma 3.

3 Triangulating Polygons

As mentioned in the introduction, Klincsek's algorithm allows to triangulate polygons optimally for a large number of measures using dynamic programming.

In this paper we have the additional requirement that the resulting triangulation must be order-k, therefore the classical algorithm must be adapted to include only order-k triangles.

The input of the algorithm is a polygon P, defined by its vertices in clockwise order: $p_0, p_1, \ldots, p_{n-1}$. The output is a k-OD triangulation of optimum cost, if it exists. It may be that the useful order of some of the polygon edges is such that no order-k triangulation of P exists at all.

The dynamic programming algorithm finds an optimal solution by combining solutions of smaller problems in a systematic way. The typical algorithms have $O(n^3)$ running time and use an $n \times n$ matrix L, which in our problem has the following meaning: $L[i, i+j]$ is the cost of the optimal k-OD triangulation of the polygon $P_{i,i+j}$, defined by the edges of P between p_i and p_{i+j}, plus edge $\overline{p_{i+j}p_i}$.

The matrix can be filled in a recursive way. The simplest entries are the ones of the form $L[i, i+1]$, which have cost 0. The recursive formula for $L[i, i+j]$ is:

$$L[i, i+j] = \min_{q=1,2,\ldots,j-1} (Cost(p_i, p_{i+q}, p_{i+j}) \oplus L[i, i+q] \oplus L[i+q, i+j]) \quad (1)$$

The expression $Cost(p_i, p_{i+q}, p_{i+j})$ denotes the cost of triangle $\triangle p_i p_{i+q} p_{i+j}$, and the operator $\oplus$ represents a way to combine the values of the subproblems. Their precise meaning depends on the measure being optimized. Edges $\overline{p_i p_{i+q}}$ and $\overline{p_{i+q} p_{i+j}}$ must be diagonals. Triangles that are not contained entirely inside P or are not k-OD have cost $+\infty$. Checking the latter (verifying that there are no more than k points inside the circumcircle of the triangle) would take $O(\log n + k)$ time [11], but if we precompute all the order-k triangles for each useful edge (see Section 2) and store them in a perfect hashing table [9], we can find out in $O(1)$ time if the triangle is among the order-k triangles. Note that measures of the type $\min \max(\ldots)$ can also be optimized with the same method, by using a recursive formula similar to (1).

We can take advantage of the properties of higher order Delaunay triangulation to reduce the running time significantly. The main steps of the algorithm are the following (details are given below). In the preprocessing phase we compute all the useful edges and filter out the ones that are not fully contained inside the polygon. The order-k triangles adjacent to each useful edge are precomputed. The triangulation algorithm proceeds by applying Equation (1), using a perfect hashing table to store the solutions to the subproblems already computed.

For a given edge $\overline{p_i p_j}$, the number of possible *third points* to form a triangle is not $O(n)$, as in the normal triangulation problem, but $O(k)$ (see Lemma 2). If for every edge we precompute these $O(k)$ points, we can improve the $O(n^3)$ dynamic programming running time to $O(kn^2)$, after spending $O(k^2 n \log k + kn \log n)$ time in the precomputation of the useful edges and the order-k triangles (see Lemma 3).

Every time an edge is considered as a candidate to be in an optimal triangulation, we must also check that it does not intersect the polygon boundary and that it does not lie outside the polygon. This check can be done in $O(\log n)$

time per edge using an algorithm for ray shooting in polygons [13]. This adds an $O(kn \log n)$ term to the preprocessing time, which does not increase the previous asymptotic running time.

Finally, the matrix L has $O(n^2)$ cells, each corresponding to one potential edge. However, we know that only $O(kn)$ edges will be useful, so it is not necessary to keep a data structure of quadratic size. In order to avoid wasting time and space on edges that are not useful, we will not use the standard matrix-based dynamic programming algorithm, but we will use a *memoized* version instead. The idea is to use Equation (1) to compute $L[0, n-1]$, and maintain a perfect hashing table where we will store all the subproblems already solved. Notice that each subproblem $L[i, j]$ is associated with the insertion of an edge $\overline{p_i p_j}$, which must be useful k-OD. Hence, only $O(kn)$ subproblems will be computed and stored. The same table used to store the order-k triangles incident to an edge can be extended to also store the value of the subproblem associated with that edge. To solve one particular problem $O(k)$ time is needed, yielding a total running time of $O(k^2 n)$, plus $O(k^2 n \log k + kn \log n)$ preprocessing time.

Theorem 1. *In $O(k^2 n \log k + kn \log n)$ expected time one can compute an optimal order-k Delaunay triangulation of a simple polygon with n vertices that optimizes a decomposable measure.*

4 Triangulating Polygons with Points Inside

In this section we consider the more general problem of finding an optimal triangulation of a simple polygon that contains h components in its interior. A component can be either a point or a connected component made of several points connected by edges. We will denote the polygon with the components by P. We can reuse the algorithm from the previous section if we connect one vertex of each component to some other vertex in order to remove all the loose parts. To find the optimal triangulation we must try, in principle, all the possible ways to make these connections. The number of them depends on h and on the order k. In principle, there are $O(n)$ ways to connect each component. However, since we need only one edge that connects the component to the outer boundary of the polygon, we don't need to try $O(n)$ but only $O(k)$ edges.

Lemma 4. *Let P be a polygon with h components inside. There is a collection of $O(k)^h$ sets, of h edges each, such that: (i) for every set in the collection, the edges in the set connect all the components in P to the outer boundary; (ii) any order-k Delaunay triangulation includes the edges of some set in the collection.*

Proof. First we show that for a given component in P, any order-k triangulation T of P must connect the component to the rest of the polygon by one of $O(k)$ edges. Let u be the topmost point among the boundary points of the components inside the polygon. Everything above u is part of the outer polygon boundary. Let $\overline{uv}$ be an edge of the Delaunay triangulation of the *point set* induced by P

and its components (ignoring the edges), with v higher than u. Since $\overline{uv}$ is a Delaunay edge, we know from [11] that the useful k-OD edges that cross it have $O(k)$ endpoints on each side of $\overline{uv}$. If $\overline{uv}$ is not part of T, at least one of the useful edges that cross it must be. Let $\overline{xy}$ be the first of these edges (the first one encountered when going from u to v along $\overline{uv}$) in T, then triangle $\triangle uxy$ must be part of T. This implies that edges $\overline{ux}$ and $\overline{uy}$ are part of it as well. Hence, either $\overline{uv}$ or one of the $O(k)$ possible edges of type $\overline{uz}$ (for $z = x$ or $z = y$) must be in T, and connects v to a higher point of the outer boundary of P.

Following the same idea, for each of the h components we can find a set of $O(k)$ useful k-OD edges such that any triangulation T connects each component using one of these $O(k)$ edges. The result follows. ⊠

Using the previous result, our algorithm will try the $O(k)^h$ different ways to connect the loose components in P. Let $P_1, \cdots, P_\eta$ be the $O(k)^h$ different polygons that are tried, and let H_i be the set of new boundary edges of P_i. For each P_i, besides computing the boundary, we must compute the intersections between the $O(kn)$ useful edges and the new h edges in H_i, which were added to connect the loose components. This is because during the triangulation we need to consider only edges that make the polygon simply-connected.

The computation of these intersections can be done once and maintained between successive polygons without increasing the asymptotic running time. During the preprocessing phase, we will compute all the intersections between useful k-OD edges. A useful k-OD edge can intersect $O(k^2)$ other useful k-OD edges [11]. Therefore the total number of intersections is $O(k^3n)$, and they can be computed in time $O(kn \log kn + k^3n) = O(kn \log n + k^3n)$ [4]. We store for each useful edge all the other useful edges that it intersects and in addition we keep a counter. The counter will be used to keep track of how many edges in H_i each useful edge intersects.

The algorithm will iterate through the polygons in such a way that two consecutive polygons P_i and P_{i+1} differ only in the edge chosen for one of the components. Then during the $(i+1)$-th step the counters for H_i are already computed, and one can compute the counters for H_{i+1} very easily, as follows: let e_{out} be the edge that is removed and e_{in} the new edge (that is, $H_{i+1} = H_i \backslash \{e_{out}\} \cup \{e_{in}\}$). Firstly, all the $O(k^2)$ useful edges that intersect e_{out} must have their counters decreased by one. Secondly, all counters of the $O(k^2)$ edges that intersect e_{in} are incremented by one. Both sets of edges were previously computed during the preprocessing phase and can be accessed in constant time. Hence the time needed to update the intersection information from one polygon to the next one is $O(k^2)$.

We conclude that the total time required to compute all the new intersections is $O(kn \log n + k^3n)$ for the preprocessing and $O(k^2)$ per polygon. Note that the useful edges and order-k triangles do not need to be recomputed, since they only depend on the point set, which has not changed.

Triangulating each generated polygon using the algorithm from the previous section takes $O(k^2n)$ time, yielding a total time of $O(k^2n \log k + nk \log n + kn \log n + k^3n) + O(k)^h(k^2 + k^2n) = O(kn \log n) + O(k)^{h+2}n$ (because $h \geq 1$).

Theorem 2. *An optimal order-k Delaunay triangulation of a simple polygon with n boundary vertices and $h \geq 1$ components inside that optimizes a decomposable measure can be computed in $O(kn \log n) + O(k)^{h+2}n$ expected time.*

5 Some Other Measures

The approach described above can also be used to optimize some other, non-decomposable, measures. The challenging part is adapting the recursive formula to make the subproblems independent. In the full paper we show how this can be done for minimizing the maximum vertex degree, minimizing the number of local minima (if the points have an elevation), and even optimizing functions of quadrilaterals, such as minimizing the maximum area ratio of triangles sharing an edge. In general these variations involve more complicated algorithms and have higher running time.

Interestingly, a similar approach also allows to find the lowest order completion of a polygon and a set of diagonals, that is, finding a higher order Delaunay triangulation, with the lowest possible order, which contains the given diagonals. The more general problem of finding a lowest order completion of a point set with respect to a set of edges is still open; polynomial time algorithms are known only for $k \leq 3$ [12].

6 Application to Point Sets

Any point set can be optimally triangulated using the results from Section 4 if it is seen as a polygon made of its convex hull with points inside. In general, this will lead to a running time exponential in n, so this is of no practical use. For low order Delaunay triangulations, the situation is better. Given a point set and an order k, there are *fixed edges* that are present in any order-k triangulation, and partition the convex hull of the point set into a number of polygons with components inside. For $k = 1$, it is known that these polygons are always empty triangles or quadrilaterals [11], which simplifies the optimization of several measures. As k increases, the number of fixed edges decreases until it is reduced to little more than the convex hull. Moreover, for $k > 1$ the polygons may be larger and may have many components inside. However, our experiments on the structure of higher order Delaunay triangulations suggest that in practice, for small values of k, the appearance of such polygons is rather unlikely.

We summarize part of the results of experiments carried out on randomly generated point sets, which show that for small values of k, the polygons created contain only a few components. The experiments consisted in generating random point sets of between 1000 and 5000 points, and for different values of k, computing the partition into polygons with components inside given by the fixed edges. The size of the polygons and the number of components was registered. These are the two factors, besides k, involved in the running time of the algorithm of Section 4. Tables 1 and 2 show the results for point sets between 1000 and 5000 points, and $k = 1, \ldots, 10$. It is worth mentioning that since the

Table 1. Structure of order-k triangulations: average / maximum number of components per polygon for random point sets of n points, averaged over 200 runs

k	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$	$n = 5000$
1	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
2	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
3	0.00 / 0.03	0.00 / 0.03	0.00 / 0.04	0.00 / 0.06	0.00 / 0.07
4	0.00 / 0.44	0.00 / 0.68	0.00 / 0.80	0.00 / 0.94	0.00 / 0.90
5	0.01 / 1.15	0.01 / 1.35	0.01 / 1.50	0.01 / 1.53	0.01 / 1.57
6	0.05 / 2.22	0.05 / 2.63	0.04 / 2.77	0.04 / 2.91	0.04 / 3.13
7	0.18 / 5.52	0.17 / 6.77	0.16 / 6.82	0.16 / 7.38	0.16 / 7.87
8	0.55 / 18.95	0.53 / 24.19	0.52 / 29.85	0.52 / 31.28	0.52 / 35.85
9	1.45 / 61.05	1.47 / 108.70	1.47 / 150.83	1.52 / 205.30	1.53 / 242.75
10	3.08 / 115.55	3.33 / 225.36	3.44 / 332.13	3.62 / 443.35	3.72 / 552.41

Table 2. Structure of order-k triangulations: average / maximum size of polygons for random point sets of n points, averaged over 200 runs. Since the dynamic programming algorithm works by considering triangles, the size is measured as the number of Delaunay triangles that the polygon contains. More details in the full version.

k	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$	$n = 5000$
1	1.35 / 2.00	1.35 / 2.00	1.35 / 2.00	1.36 / 2.0	1.35 / 2.00
2	1.99 / 6.82	1.99 / 7.41	1.99 / 7.54	2.00 / 7.63	2.00 / 7.77
3	2.84 / 12.77	2.85 / 13.84	2.84 / 14.60	2.85 / 15.14	2.85 / 15.59
4	4.02 / 23.54	4.04 / 25.95	4.04 / 28.02	4.04 / 28.37	4.04 / 29.95
5	5.80 / 43.97	5.82 / 52.90	5.83 / 54.55	5.84 / 56.25	5.85 / 59.72
6	8.63 / 91.66	8.68 / 108.55	8.74 / 115.63	8.74 / 123.63	8.78 / 129.58
7	13.48 / 222.25	13.56 / 263.83	13.72 / 292.36	13.83 / 310.12	13.86 / 337.01
8	21.71 / 569.40	22.40 / 749.15	22.76 / 980.10	23.21 / 1038.78	23.41 / 1223.43
9	34.99 / 1294.76	37.26 / 2425.46	38.32 / 3434.07	39.89 / 4716.74	40.58 / 5661.38
10	49.91 / 1755.54	56.27 / 3566.14	59.04 / 5341.94	62.56 / 7195.49	64.77 / 9034.72

convex hull of the point set limits the growth of the polygons, the results may be influenced slightly by boundary effects.

A few observations are in order. The experiments confirm that for $k \leq 3$, it is very unlikely to find polygons with components inside. Even though for $k \geq 2$ one can build examples where that is the case, they hardly arise in random point sets. Even for orders up to 5 or 6, the size of the polygons and number of components are small enough to be useful for practical purposes. As a result, finding optimal triangulations that in general are NP-hard, like the minimum weight triangulation, can be done in practice if the Delaunay order is low enough. The small values of k are the most useful in practice, for several reasons. On the one hand, as k increases, the shape of the triangles deteriorates. On the other hand, previous experimental results [7], related to realistic terrain modeling, have shown that low values of k are enough to obtain important improvements

on several terrain measures (like the number of local minima), making small values of k particularly interesting for these applications.

7 Discussion

We studied algorithms to find higher order Delaunay triangulations of polygons that optimize a decomposable measure. Based on an existing technique for polygon triangulation, we proposed an algorithm to compute an optimal triangulation of a polygon restricted to order-k triangulations. Their specific properties allowed us to reduce an $O(n^2)$ factor to $O(k^2)$, a substantial improvement since k will be, in general, much smaller than n [7]. Our method can also be extended to some non-decomposable measures, like maximum vertex degree. For the more general problem of triangulating optimally a polygon with components inside, we presented an algorithm that is fixed-parameter tractable for $k = O(1)$.

We also gave experimental evidence suggesting that the specific structure of order-k Delaunay triangulations, for small values of k, makes the algorithm presented here applicable to point sets. This constitutes the first practical result on optimal higher order Delaunay triangulations for $k > 1$, allowing to optimize any decomposable function over a class of well-shaped triangulations.

Acknowledgments. We thank Remco Burema for carrying out the experimental work in this paper.

References

1. Benkert, M., Gudmundsson, J., Haverkort, H.J., Wolff, A.: Constructing interference-minimal networks. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 166–176. Springer, Heidelberg (2006)
2. Bern, M., Edelsbrunner, H., Eppstein, D., Mitchell, S., Tan, T.S.: Edge insertion for optimal triangulations. Discrete Comput. Geom. 10(1), 47–65 (1993)
3. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In: Du, D.-Z., Hwang, F.K. (eds.) Computing in Euclidean Geometry, Lecture Notes Series on Computing, 2nd edn. vol. 4, pp. 47–123. World Scientific, Singapore (1995)
4. Chazelle, B., Edelsbrunner, H.: An optimal algorithm for intersecting line segments in the plane. J. ACM 39(1), 1–54 (1992)
5. Chew, L.P.: Constrained Delaunay triangulations. Algorithmica 4, 97–108 (1989)
6. De Floriani, L., Falcidieno, B., Pienovi, C.: Delaunay-based representation of surfaces defined over arbitrarily shaped domains. Comput. Vision Graph. Image Process. 32, 127–140 (1985)
7. de Kok, T., van Kreveld, M., Löffler, M.: Generating realistic terrains with higher-order Delaunay triangulations. Comp. Geom. Theory Appl. 36, 52–65 (2007)
8. Edelsbrunner, H., Tan, T.S.: A quadratic time algorithm for the minmax length triangulation. SIAM J. Comput. 22, 527–551 (1993)
9. Fredman, M.L., Komlos, J., Szemeredi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM 31(3), 538–544 (1984)

10. Gilbert, P.D.: New results in planar triangulations. Report R-850, Coordinated Sci. Lab., Univ. Illinois, Urbana, IL (1979)
11. Gudmundsson, J., Hammar, M., van Kreveld, M.: Higher order Delaunay triangulations. Comput. Geom. Theory Appl. 23, 85–98 (2002)
12. Gudmundsson, J., Haverkort, H., van Kreveld, M.: Constrained higher order Delaunay triangulations. Comput. Geom. Theory Appl. 30, 271–277 (2005)
13. Hershberger, J., Suri, S.: A pedestrian approach to ray shooting: Shoot a ray, take a walk. J. Algorithms 18, 403–431 (1995)
14. Keil, J.M., Vassilev, T.S.: Algorithms for optimal area triangulations of a convex polygon. Comput. Geom. Theory Appl. 35(3), 173–187 (2006)
15. Klincsek, G.T.: Minimal triangulations of polygonal domains. Discrete Math. 9, 121–123 (1980)
16. van Kreveld, M., Löffler, M., Silveira, R.I.: Optimization for first order Delaunay triangulations. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 175–187. Springer, Heidelberg (2007)
17. Levcopoulos, C., Krznaric, D.: The greedy triangulation can be computed from the Delaunay triangulation in linear time. Comput. Geom. Theory Appl. 14, 197–220 (1999)
18. Mark, D.: Network models in geomorphology. In: Anderson, M.G. (ed.) Modelling Geomorphological Systems, ch. 4, pp. 73–97. John Wiley & Sons, Chichester (1988)
19. Mulzer, W., Rote, G.: Minimum weight triangulation is NP-hard. In: Proc. 22nd Annu. ACM Sympos. Comput. Geom., pp. 1–10 (2006)
20. Neamtu, M.: Delaunay configurations and multivariate splines: a generalization of a result of B. N. Delaunay. Trans. Amer. Math. Soc. 359(7), 2993–3004 (2007)
21. Pebay, P.P., Baker, T.J.: A comparison of triangle quality measures. In: Proceedings of the 10th International Meshing Roundtable, pp. 327–340 (2001)
22. Silveira, R.I., van Kreveld, M.: Towards a Definition of Higher Order Constrained Delaunay Triangulations. In: Proceedings of the 19th Annual Canadian Conference on Computational Geometry (CCCG 2007), pp. 161–164 (2007)