Optimal Higher Order Delaunay Triangulations of Polygons *

Rodrigo I. Silveira† Marc van Kreveld‡

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.

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 points of P. In this paper we focus mainly on triangulations of polygons. The goal is to decompose the polygon into triangles by drawing diagonals. For a given point set or polygon, many triangulations exist, so it is possible to try to find one that is the best according to some criterion that measures some property of the triangulation. Typical properties are local properties of the triangles (like area, height or minimum angle) or global properties of the triangulation (such as total edge length or maximum vertex degree).

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 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. Gudmundsson et al. [6] define *higher order Delaunay triangulations*, a class of well-shaped triangulations where a few points are allowed inside the circumcircles of the triangles. A triangulation is *order-k Delaunay* if the circumcircle of the three vertices of any triangle contains at most k other points (see Figure 1).

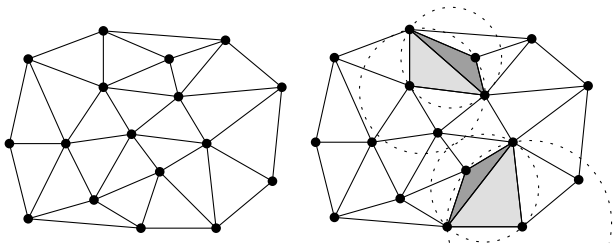

Figure 1: A Delaunay triangulation ($k = 0$) (left), and an order-2 triangulation (right). Light gray triangles are first order, the medium grey ones are second order.

When the goal is to optimize only one criterion, optimal polygon triangulations can be computed in polynomial time for many measures. The main tool for this is a dynamic programming algorithm attributed to Klincsek [9], that 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. 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 faster, in $O(n^2)$ time [8]. A few methods exist for optimally triangulating point sets, and can be applied also to polygons. The edge insertion paradigm [1] can be used to optimize several (decomposable) 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 [4]. The *greedy triangulation*, which lexicographically minimizes the sorted vector of length edges, can be constructed in $O(n^2)$ time [10].

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. There are not many results on optimal higher order Delaunay triangulations. For the case $k = 1$, the

*This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503 (GADGET) and under the project GOGO.

†Department of Information and Computing Science, Utrecht University, the Netherlands, rodrigo@cs.uu.nl

‡Department of Information and Computing Science, Utrecht University, the Netherlands, marc@cs.uu.nl

triangulations have a special structure that allows a number of measures (for example min / max area triangle, total edge length, number of local minima in a terrain) to be optimized in $O(n \log n)$ time [6]. When $k > 1$, 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 [3].

In this paper we extend the algorithm of Klincsek [9] to allow to optimize a decomposable measure for a simple polygon over order-k Delaunay triangulations. A straightforward extension of Klincsek's algorithm leads to $O(n^3 \log n)$ running time. Our main contribution is improving this time 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 the small values of k are the ones that are most interesting [3]. An extension to triangulate polygons with points inside is also presented.

2 Higher Order Delaunay Triangulations

We begin by presenting some basic concepts on order-k Delaunay triangulations, together with some results that will be needed later. We assume non-degeneracy of the input set P: no four points are cocircular.

Definition 1 *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 *For a set P of points, the* order *of an edge $\overline{pq}$ between two points $p, q \in P$ is the minimum number of points inside any circle that passes through p and q. 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*.

Lemma 1 *(from [6]) 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.*

Lemma 2 *(from [6]) Let $\overline{uv}$ be any Delaunay edge. The number of useful k-OD edges in a triangulation T that intersect $\overline{uv}$ is $O(k)$.*

We extend these results with one more lemma.

Lemma 3 *Let $\overline{uv}$ be a useful k-OD edge. Then there are at most $O(k)$ order-k triangles that have $\overline{uv}$ as one of their edges.*

We provide only a sketch of the proof here. We slide a circle in contact with u and v until it touches a first point r_1 to the right of edge $\overrightarrow{uv}$. This could be a *third point* of a triangle incident to $\overrightarrow{uv}$ because it is possible for $C(u, v, r_1)$ to contain less than $k + 1$ points. We slide the circle again until it touches a second point r_2. Now $C(u, v, r_2)$ contains at least one point (r_1). Continuing in this way it can be seen that $C(u, v, r_{k+1})$ contains at least k points, hence no further point can be a *third point*. An identical argument applies to the left side.

Next we show that all order-k triangles can be computed efficiently.

Lemma 4 *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.*

We provide a sketch of the algorithm. There are $O(kn)$ useful k-OD edges [6], with $O(k)$ incident order-k triangles each (Lemma 3), therefore the total number of order-k triangles is $O(k^2 n)$.

All the useful edges can be computed in $O(k^2 n + kn \log n)$ expected time [6]. Moreover, within the same 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 (as in the proof of Lemma 3). This can be done in $O(k \log k)$ time by sorting the centers of the circles. By going through these sorted lists from left to right, we can count the number of points inside all the circumcircles in $O(k)$ time.

Still, some of these triangles may contain points inside, so we need to discard them. 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 laying in the lower-left staircase of the point set, and can be found in $O(k \log k)$ time by a sweep line algorithm.

The time needed to find the triangles for one useful edge is $O(k \log k)$. Hence the useful edges and order-k triangles can be found in $O(k^2 n \log k + kn \log n)$ time.

3 Triangulating polygons

As mentioned in the introduction, Klincsek's algorithm allows to triangulate polygons in an optimal fashion for a large number of measures using dynamic programming. In this paper we have the additional requirement that the triangulation must be order-k, therefore only order-k triangles can be used.

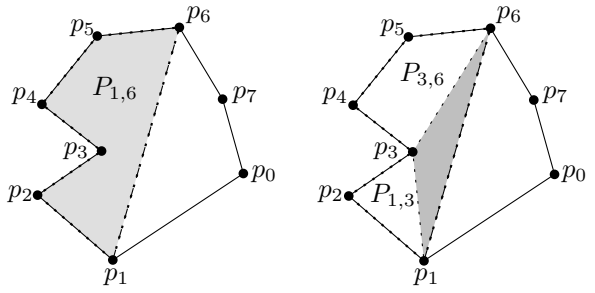

Figure 2: Dynamic programming method. The value of an optimal triangulation of $P_{1,6}$ is stored in $L[1,6]$.

The input is a polygon P defined by its vertices in clockwise order $\{p_0, p_1, ..., p_{n-1}\}$. The output is a k-OD triangulation of optimum cost.

The dynamic programming algorithm finds an optimal solution by combining solutions of smaller problems in a systematic way. The typical algorithms use an $n \times n$ matrix L, which in our problem would have the following meaning: $L[i, i+j]$ contains the cost of the optimal k-OD triangulation of the polygon $P_{i,i+j}$, defined by the edges between p_i and p_{i+j}, plus edge $\overline{p_{i+j}p_i}$. See Figure 2 for an example.

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 formula for $L[i, i+j]$ is:

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

Where $\overline{p_i p_{i+q}}$ and $\overline{p_{i+q} p_{i+j}}$ must not intersect the boundary of P. The expression $Cost(p_i, p_{i+q}, p_{i+j})$ is the cost of the triangle $\triangle p_i p_{i+q} p_{i+j}$. Triangles that are not k-OD have cost $+\infty$. Checking the order of the triangle would take $O(\log n + k)$ time [6], but if we precompute all the order-k triangles for each useful edge and store them in a perfect hashing table [5], we can answer the question in $O(1)$ time.

The idea behind the formula is that if edge $\overline{p_i p_{i+j}}$ is part of an optimal triangulation, then it must also contain some triangle incident to it, dividing the polygon into two subpolygons that can be solved independently, see Figure 2. Note that measures of the type $\min\max(\ldots)$ can also be optimized with the same method by a recursive formula similar to (1).

This algorithm has $O(n^3)$ running time because if the matrix is computed in the right way, we always have the required elements of L calculated when computing $L[i, i+j]$. Filling in one cell takes $O(n)$ time, leading to $O(n^3)$ time for the whole matrix.

The special properties of order-k Delaunay triangulation allow to reduce the running time significantly.

First of all, for a given edge $\overline{p_i p_j}$, the number of possible *third points* is not $O(n)$ but $O(k)$ (see Lemma 3). If for every edge we precompute these $O(k)$ points, we can improve the running time to $O(kn^2)$, after spending $O(k^2 n \log k + kn \log n)$ in the precomputation of the useful edges and the order-k triangles (Lemma 4).

Secondly, matrix L has $O(n^2)$ cells, each corresponding to one edge. However, only $O(kn)$ can be useful, so it is not necessary to keep a data structure of quadratic size. We will use *memoization* instead. We will apply the recursive formula (1) to compute $L[0, n-1]$, but to avoid solving the same subproblem $L[i, j]$ more than once, we will use a perfect hashing table to store all the subproblems already solved. Since each subproblem $L[i, j]$ is associated with a k-OD edge $\overline{p_i p_j}$, only $O(kn)$ subproblems will be computed and stored. Each particular problem is solved in $O(k)$ time, yielding a total running time of $O(k^2 n)$, plus $O(k^2 n \log k + kn \log n)$ preprocessing time.

Finally, every edge considered must not intersect the polygon boundary and must not lie outside the polygon. This can be checked in $O(\log n)$ time per edge using an algorithm for ray shooting in polygons [7]. This adds an $O(kn \log n)$ preprocessing term, which does not increase the asymptotic running time.

Theorem 5 *An optimal order-k Delaunay triangulation of a simple polygon with n vertices that optimizes a decomposable measure can be computed in* $O(k^2 n \log k + kn \log n)$ *expected time.*

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 $h \geq 1$ components in its interior. A component can be either a point or a connected component made of several points connected by edges. We can reuse the algorithm from the previous section if we connect 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.

In principle, there are $O(n)$ ways to connect each component. However, since what we need is to find only one edge to connect the component to the polygon, it is enough to try only $O(k)$ edges. The reason is as follows. Let u be the top-most point inside the polygon. Everything above u is part of the polygon boundary. Let $\overline{uv}$ be a Delaunay edge of the constrained Delaunay triangulation of the polygon and its components, with v above u. By Lemma 2, a Delaunay edge can be crossed by only $O(k)$ useful k-OD edges. If $\overline{uv}$ is not part of the optimal triangulation, at least one of the $O(k)$ 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 $\overline{uv}$), then triangle $\triangle uxy$ must be part of the optimal triangulation. This implies that edges $\overline{ux}$ and $\overline{uy}$ are part of it as well. Then in our algorithm we can try $\overline{uv}$ and

for each of the $O(k)$ possible edges of the type $\overline{uz}$, for $z = x$ or $z = y$ (any of them as long as it connects u to a higher point). At least one of these edges must be part of any optimal triangulation,[1] and connects v to the triangulated part.

Trying every possible way to connect each inner component to the boundary of the polygon involves trying $O(k)^h$ cases. 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 remove the loose components. The computation of these intersections can be done once and maintained between successive polygons without increasing the asymptotic running time. Though we omit the details here, we can compute during the preprocessing phase an intersection graph for the useful k-OD edges, in $O(kn \log n + k^3 n)$ time, and then update it in $O(k^2)$ time per polygon by iterating through the polygons in a certain order.

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

Theorem 6 *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 Application to point sets

Any point set can be triangulated using the results from the previous sections 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 higher order Delaunay triangulations, the situation might be better. Given a point set and an order k, there is always a set of *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 components are always empty triangles or quadrilaterals [6]. For $k > 1$ this is not true anymore, but preliminary experimental results suggest that in practice, for small values of k, the polygons created contain only a few components (Figure 3). Therefore it is possible that the algorithms presented here can be used in practice to triangulate point sets for small values of k.

[1] Actually, it may happen that none of these edges are in any optimal triangulation. That can occur only if $\overline{uv}$ crosses a boundary edge of the polygon that is not useful order-k, which implies that no order-k triangulation exists.

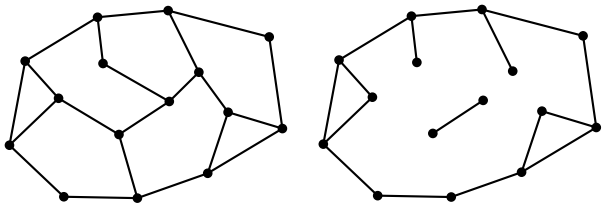

Figure 3: Fixed edges: $k = 2$ (left) and $k = 4$ (right).

6 Discussion

We studied algorithms to find higher order Delaunay triangulations for polygons that optimize a decomposable measure. An existing algorithm for polygon triangulation was extended to optimize over this special class of triangulations. Their specific properties allowed to improve the running time substantially, reducing a $O(n^2)$ factor to $O(k^2)$, a considerable improvement since k will be, in general, significantly smaller than n [3]. Even though not discussed here, other measures, like maximum vertex degree or number of local minima in a terrain, can also be optimized using a similar approach.

References

[1] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T. S. Tan. Edge insertion for optimal triangulations. *Discrete Comput. Geom.*, 10(1):47–65, 1993.

[2] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 590–600, 1988.

[3] T. de Kok, M. van Kreveld, and M. Löffler. Generating realistic terrains with higher-order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 36:52–65, 2007.

[4] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. *SIAM J. Comput.*, 22:527–551, 1993.

[5] M. L. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, July 1984.

[6] J. Gudmundsson, M. Hammar, and M. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 23:85–98, 2002.

[7] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[8] J. M. Keil and T. S. Vassilev. Algorithms for optimal area triangulations of a convex polygon. *Comput. Geom. Theory Appl.*, 35(3):173–187, 2006.

[9] G. T. Klincsek. Minimal triangulations of polygonal domains. *Discrete Math.*, 9:121–123, 1980.

[10] C. Levcopoulos and A. Lingas. Fast algorithms for greedy triangulation. *BIT*, 32(2):280–296, 1992.