Towards a Definition of Higher Order Constrained Delaunay Triangulations

Rodrigo I. Silveira

Marc van Kreveld

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2008-007

www.cs.uu.nl

ISSN: 0924-3275

Towards a Definition of Higher Order Constrained Delaunay Triangulations *

Rodrigo I. Silveira† Marc van Kreveld†

Abstract

When a triangulation of a set of points and edges is required, the constrained Delaunay triangulation is often the preferred choice because of its well-shaped triangles. However, in applications like terrain modeling, it is sometimes necessary to have flexibility to optimize some other aspect of the triangulation, while still having nicely-shaped triangles and including a set of constraints.

Higher order Delaunay triangulations were introduced to provide a class of well-shaped triangulations, flexible enough to allow the optimization of some extra criterion. But they are not able to handle constraints: a single constraining edge may cause that all triangulations with that edge have high order, allowing ill-shaped triangles at any part of the triangulation.

In this paper we generalize the concept of the constrained Delaunay triangulation to higher order constrained Delaunay triangulations. We study several possible definitions that assure that an order-k constrained Delaunay triangulation exists for any $k \geq 0$, while maintaining the character of higher order Delaunay triangulations of point sets.

Several properties of these definitions are studied, and efficient algorithms to support computations with order-k constrained Delaunay triangulations are also discussed. For the special case of $k = 1$, we show that many measures can be optimized efficiently in the presence of constraints.

1 Introduction

Higher order Delaunay triangulations [11] are a generalization of the Delaunay triangulation. They provide a class of triangulations that are reasonably well-shaped, depending on a parameter k. A triangulation is *order-k Delaunay* if the circumcircle of the three vertices of any triangle contains at most k other points. If no four points are cocircular, for $k = 0$ there is only one higher order Delaunay triangulation, equal to the Delaunay triangulation. As k is increased, the shape quality of the triangles may decrease, but the number of triangulations generally increases, and hence there is more flexibility to optimize some other criterion. The concept of higher order Delaunay triangulation has been successfully applied to several areas, including terrain modeling [7], minimum interference networks [2] and multivariate splines [16].

When working with triangulations it is often the case that a given set of edges must be included in the triangulation. We refer to these edges as constraints, or constraining edges. For

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

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

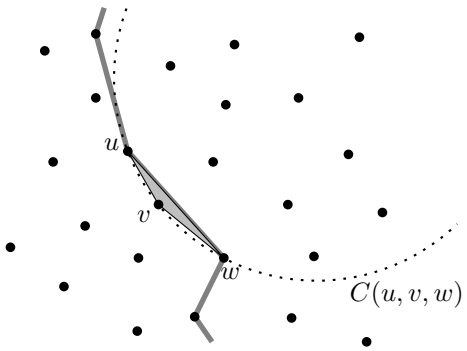


Figure 1: A point set is augmented with some constraining edges (in gray), which must be incorporated into the triangulation. Any triangulation of this point set that includes the gray edges must include triangle $\triangle uvw$, which has a very high order. This allows all the other triangles to have high order too.

example, in mesh generation, the mesh must respect the boundary edges of the components. When working with polyhedral terrains for hydrologic applications, it is common to augment the terrain with the edges representing the drainage network [6]. Other uses of constrained triangulations include hierarchical surface models [5], terrain data integration [14] and map generalization [22].

Regardless of the reason for including a set of edges, in many cases it is important that the triangulation containing them has nicely-shaped triangles. The *constrained Delaunay triangulation* (CDT) [4] includes a given set of edges and is "as close as possible to the Delaunay triangulation". This is achieved by relaxing the empty-circle property of the Delaunay triangulation: points are allowed inside the circumcircles of the triangles if they are separated from the triangle by some constraining edge.

The only previous work on order-k Delaunay triangulations with constraints focuses on finding a lowest-order Delaunay triangulation that includes a given set of edges [12], but the definition of order used does not depend on the constraints. Until now, there was no concept equivalent to the CDT for higher order Delaunay triangulations. That implies that if one of the constraints causes the inclusion of a triangle of very high order, then the whole triangulation will have at least that order (see Figure 1). Therefore, all the triangles, even the ones far away from these constraints, will be allowed to have that very high order, and, more important, a not very good shape.

The need for a higher order version of the constrained Delaunay triangulation arises in situations where three requirements need to be met: (i) the triangulation must include a set of constraints, (ii) the triangulation must be *well-shaped* (that is, triangles should be close to equilateral) and (iii) some extra quality criterion must be optimized. The constrained Delaunay triangulation addresses only the first two requirements, whereas higher order Delaunay triangulations address only the last two.

Situations in which these three requirements are present arise, for example, in terrain modeling. For certain uses of terrain models, such as for intervisibility (determining whether an observer at a given point can see another point on the surface) or drainage analysis (for example, computing basin boundaries), it is common to augment the existing terrain model (based solely on elevation data) with a set of surface-specific features like peaks, pits,

passes, (point features), together with ridges and valleys (edge features) that connect them. These topographic features play an important role in these types of application, and their incorporation into the terrain model improves the consistency between the model and the real terrain [1, 10, 19]. When the terrain is represented by a polyhedral terrain (also known as *triangulated irregular network* or TIN), the edge features (ridge and valley edges) act as constraints that must be part of the triangulation. The triangle shape is also important, in particular, for both the analysis and visualization of the terrain, long and thin triangles should be avoided [19]. Finally, there are several extra criteria relevant to terrain analysis that can be optimized, like minimizing the number of local minima or minimizing the angle between surface normals [8, 15, 20, 24].

In this paper we address the problem of defining higher order constrained Delaunay triangulations. We achieve this by proposing several definitions of the *constrained order* of a triangle, which take the constraints into account when counting the number of points inside the circumcircles of the triangles. In the standard definition, the *order* of a triangle is defined as the number of points inside its circumcircle (every point is counted). The new definitions proposed try to reflect that some triangles may have a bad shape because of the constraints, thus their order should be defined in a different way. By defining these triangles to have a lower order than by the standard definition, the order of the whole triangulation is also kept lower.

This paper is structured as follows. We start by proposing several definitions for the notion of *constrained order* of a triangle in Section 2. The next two sections study two algorithmic problems that are important for dealing with higher order Delaunay triangulations: computing the order of one triangle (Section 3) and computing all order-k constrained Delaunay triangles (Section 4). Section 5 studies the situation in which the constraints define a simple polygon, and presents more efficient algorithms for computing the order of a triangle in this case. In Section 6 we study the particular structure of first order constrained Delaunay triangulations (that is, $k = 1$), and show that several measures can be efficiently optimized over this class of well-shaped constrained triangulations. Finally, in Section 7, we provide some concluding remarks.

From now on, we assume non-degeneracy of the input set $\mathcal{P}$: no four points are cocircular. For brevity, we will sometimes write *order* instead of *constrained order*.

2 Proposed definitions

Any suitable definition of order-k constrained Delaunay (k-OCD) triangulations must be in line with the idea of the CDT and, at the same time, be consistent with the spirit of higher order Delaunay triangulations of point sets. We summarize this by establishing a list of properties that a suitable definition should satisfy:

1. For $k = 0$ there is only one constrained triangulation, which is the CDT.
2. If there are no constraining edges, any k-OCD triangulation is a k-OD triangulation, and any k-OD triangulation is a k-OCD triangulation.
3. As k increases, the number of k-OCD triangulations also increases or stays the same.
4. If a point or endpoint of a constraint moves slightly, the constrained order changes only slightly (this is made more precise below).
5. The definition is *intuitive* for triangulations of polygons.

6. The definition is *intuitive* for triangulations of points with constraining edges.

The last two properties are subjective and will not be explicitly addressed, although all the definitions proposed here are meant to be intuitive generalizations of the standard one. The important part in any definition of higher order constrained Delaunay triangulations is defining when a point inside the circumcircle of a triangle must be counted. We propose seven different definitions, where what varies is when a point is counted in the constrained order. Recall that according to the standard definition, which we will denote `STD`, all points are counted.

Let $u, v, w \in \mathcal{P}$ and let $C = C(u, v, w)$ be the circle through u, v and w. Suppose we want to compute the order of $\triangle uvw$. No other point can be in $\triangle uvw$, otherwise it is not a triangle in the triangulation. In the definitions below, points p, r and s lie inside C, and belong to $\mathcal{P}$. Note that $\overline{uv}$, $\overline{vw}$ and $\overline{wu}$ can be constraints.

PATHCON (Path connected). A point p is counted if and only if there is a constraint-free path contained in C that connects p to some point interior to $\triangle uvw$.

SEESTRIANG (Sees triangle). A point is counted if and only if it can see some point in the interior of $\triangle uvw$.

SEESVTX (Sees vertex of triangle). A point p is counted if and only if it can see some vertex of $\triangle uvw$ and some point in its interior.

CONFEDGE (Conflicting edge). A point p is counted if and only if there is a point r, in C or in $\{u, v, w\}$, such that $\overline{pr}$ intersects the interior of $\triangle uvw$, and does not intersect any constraint.

SEESOPP (Sees opposite). A point p is counted if and only if it sees the *opposite* vertex of the triangle, that is, the vertex $x \in \{u, v, w\}$ such that $\overline{px}$ intersects the interior of $\triangle uvw$.

SEES3VTX (Sees 3 vertices). A point p is counted if and only if it can see u, v and w.

EMPTYQUAD (Empty quadrilateral). A point p is counted if and only if the quadrilateral formed by the three vertices of $\triangle uvw$ and p is empty, and the edge of $\triangle uvw$ that is a diagonal of the quadrilateral is not a constraint. This corresponds to the idea of being able to flip one edge of $\triangle uvw$.

Figure 2 shows an example where the different definitions can be compared. The strongest (most restrictive) definition is the standard one, where every point in the circumcircle is counted. As the requirements for a point to be counted increase, the definitions become weaker (less restrictive).

We now make Property 4 more precise. Assume that in some triangulation, a vertex $p \in \mathcal{P}$ moves, without changing the structure of the triangulation. Then a change in the order of the triangulation in all definitions can only occur if p becomes collinear with two points or cocircular with three points. We call this a *criticality* during the move of p. Property 4 should be interpreted such that if any point moves through only one criticality, then the order of the triangulation changes by at most one.

We make the following simple observation.

Observation 1 *All the previous definitions satisfy Properties 1 to 6, except for* `PATHCON`*, which does not satisfy Property 4, and for* `EMPTYQUAD`*, which does not satisfy Property 2.*

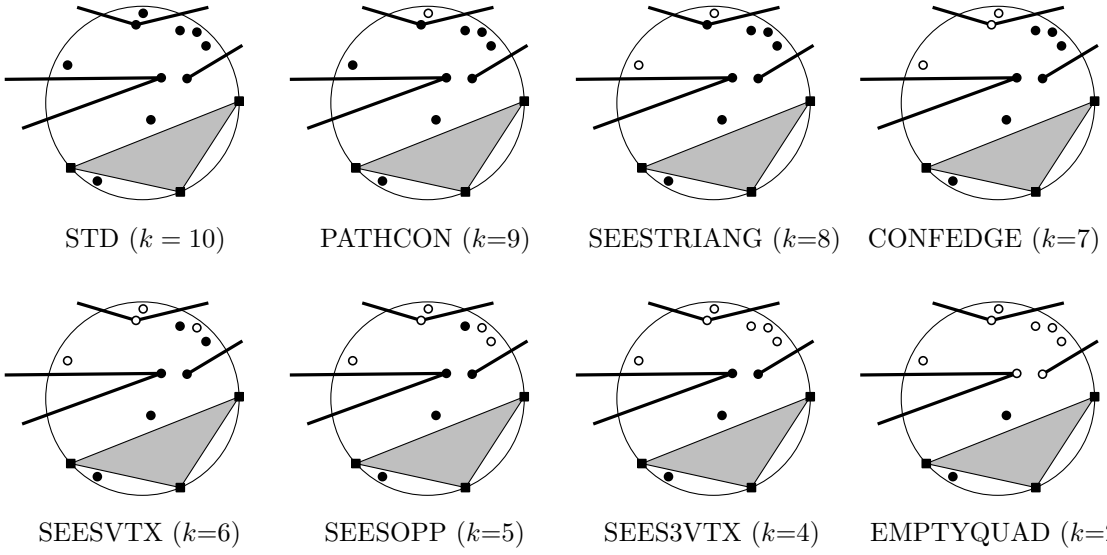


Figure 2: What is the order (k) of the gray triangle? Different definitions yield different orders. For each definition, the points that are counted are drawn as discs and the ones that are not counted as empty circles. Constraints are drawn with thick edges.

Figure 3 gives the examples of this observation for both definitions. On the left, the order of a triangle under PATHCON can increase by an arbitrary number if w is moved in such a way that the shaded region becomes connected to $\triangle uvw$. On the right, an example shows that the order according to EMPTYQUAD can be arbitrarily much smaller than the standard order, even when there are no constraints.

Even though some of the definitions do not satisfy all the desired properties, this does not mean they cannot be of use. Therefore in the next sections we still consider them as possible definitions that one may want to use. In what follows we analyze the way the different definitions relate to each other.

Definition 1 *Given a point set $\mathcal{P}$ and set of constraints $\mathcal{C}$, we define $T_k(\mathcal{P}, \mathcal{C}, \texttt{DEF})$, where* DEF *is one of the definitions from above, as the set of all order-k constrained Delaunay triangulations of $\mathcal{P}$ and $\mathcal{C}$, using the definition* DEF *to compute the order of the triangles.*

Lemma 1 *For higher order constrained Delaunay triangulations of a point set $\mathcal{P}$ with constraining edges $\mathcal{C}$ (which can define a polygon), the following inclusion relations hold:*

- $T_k(\mathcal{P}, \mathcal{C}, \texttt{STD}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{PATHCON}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESOPP}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEES3VTX}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD})$
- $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESOPP})$

Proof: We begin with the first series of inclusions. We prove the pairs of consecutive relations from left to right. Let T be a constrained triangulation of $\mathcal{P}$ and $\mathcal{C}$. Denote the order of the highest order triangle in T, using definition DEF, by $\theta(T, \texttt{DEF})$. Clearly, if $\theta(T, \texttt{DEF}) = k$, then $T \in T_j(\mathcal{P}, \mathcal{C}, \texttt{DEF})$ for any $j \geq k$.

The first inclusion follows immediately because PATHCON counts only points that are inside the circumcircle of the triangle, hence any point counted under PATHCON will be counted under the standard definition too.

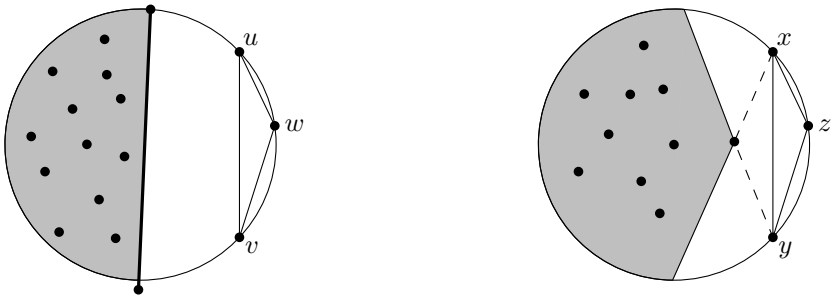


Figure 3: Left: with PATHCON, the points in the shaded area will not be counted for $\triangle uvw$. However, if w is moved an arbitrarily small distance towards $\overline{uv}$, they will all be counted. Right: the order of a triangle, using EMPTYQUAD definition, can be smaller than with respect to the standard definition even when no constraints are present. Triangle $\triangle xyz$ has constrained order 1, no matter how many points lie in the shaded region, and despite the fact that no constraints are involved.

Assume now that $\theta(T, \texttt{PATHCON}) = k$. Any point that is counted under SEESTRIANG can be connected by a line segment to the triangle, therefore will also be counted under the PATHCON definition. It follows that $\theta(T, \texttt{SEESTRIANG}) \leq \theta(T, \texttt{PATHCON}) = k$, hence $T \in T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG})$.

The inclusion $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX})$ follows immediately by definition.

The next two inclusions follow from the basic fact that seeing the opposite vertex implies seeing a vertex, and seeing the three vertices implies seeing, in particular, the opposite one. Finally, if a point can be used to create a (flippable) empty quadrilateral, it must see the three vertices of the triangle, hence $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEES3VTX}) \subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD})$.

We now prove the second series of inclusions. For the leftmost inclusion note that conflicting edges always intersect the triangle in question, hence if a point is counted under CONFEDGE, it can also see the triangle and will be counted under SEESTRIANG. The rightmost inclusion follows from the fact that if a point can see the opposite vertex, then it can use that vertex to create a conflicting edge, hence that point is also counted for CONFEDGE. ⊠

Figure 4 shows two examples where it can be seen that $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX}) \nsubseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE})$ and $T_k(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE}) \nsubseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX})$. The relation between the different definitions is illustrated in Figure 5.

Lemma 2 *For higher order constrained Delaunay triangulations of a simple polygon defined by a point set $\mathcal{P}$ and constraints $\mathcal{C}$, $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEES3VTX}) = T_k(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD})$*

Proof: Let $t = \triangle uvw$ be a triangle in a triangulation $T \in T_k(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD})$. Let $p \in \mathcal{P}$ be a point in $C(u, v, w)$ that can see the three vertices of t. Line segments $\overline{pu}$, $\overline{pv}$ and $\overline{pw}$ do not intersect any constraint. There cannot be any point $q \in \mathcal{P}$ inside the quadrilateral defined by $\{p, u, v, w\}$, because some edge should intersect one of the three segments mentioned above since q must be a vertex of the polygon boundary. Hence $\{p, u, v, w\}$ is an empty quadrilateral and p counts for the order of t under EMPTYQUAD. Therefore all the points counted for SEES3VTX are also counted for EMPTYQUAD, hence $T_k(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD}) \subseteq$

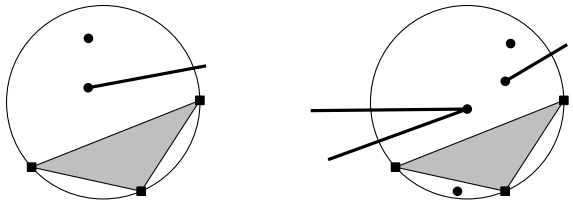


Figure 4: Examples showing that $T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) \not\subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE})$ and $T_k(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE}) \not\subseteq T_k(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG})$.

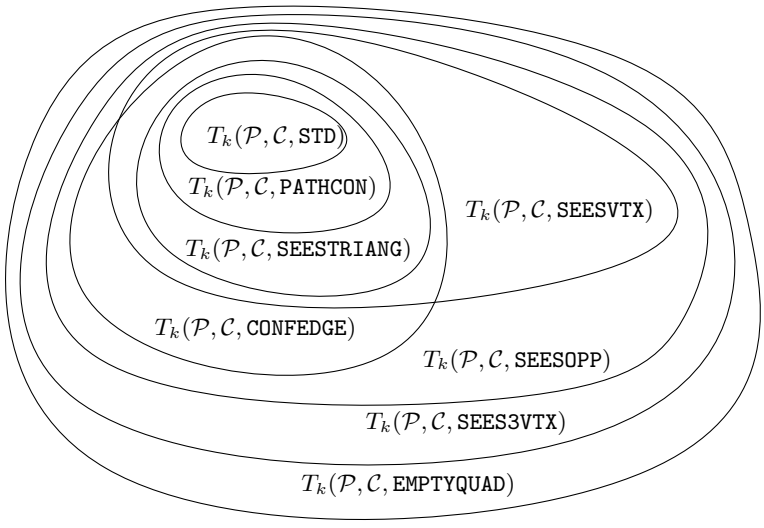


Figure 5: Inclusion relations between the different classes of triangulations.

$T_k(\mathcal{P}, \mathcal{C}, \texttt{SEES3VTX})$. The other inclusion follows directly from Lemma 1. ⊠

The notion of *useful edge* plays an important role in higher order Delaunay triangulations. We define the constrained version as follows.

Definition 2 *Let $\mathcal{P}$ be a set of points in the plane, and $\mathcal{C}$ a set of constraints. An edge $\overline{uv}$ with u,v $\in \mathcal{P}$ is* useful order *k, under definition* DEF*, if there exists an order-k constrained Delaunay triangulation of $\mathcal{P}$ (under* DEF*), with respect to $\mathcal{C}$, that includes $\overline{uv}$.*

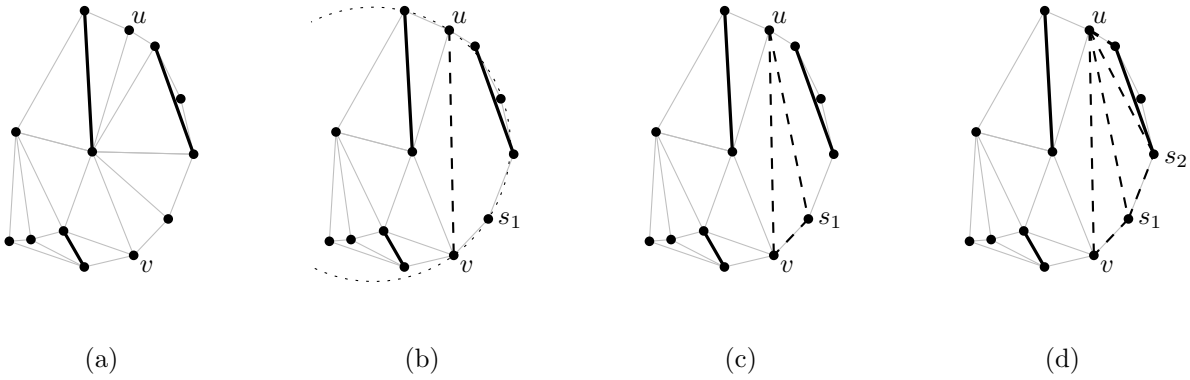
It is convenient to have efficient algorithms to test whether an edge is useful order-k, for example to speed up the computation of all the order-k triangles (see Section 4). In the case of (unconstrained) higher order Delaunay triangulations, this can be achieved by looking at the order of two specific triangles adjacent to the edge. Below we study the situation in the presence of constraints.

Lemma 3 *Let $\overline{uv}$ be an edge with $u, v \in \mathcal{P}$, let s_1 the point to the left (right) of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left (right) of $\overrightarrow{vu}$, in the constrained Delaunay sense, and such that $\overline{us_1}$ and $\overline{vs_1}$ do not intersect any constraint. If $\triangle uvs_1$ is not a k-COD triangle under definition* SEESOPP *or stronger, then $\overline{uv}$ is not a useful order-k edge under that definition.*

Proof: The proof follows the one for the unconstrained version (proof of Lemma 3 in [11]). Assume $\triangle us_1v$ is not a k-COD triangle under SEESOPP. It follows that $C(u, s_1, v)$ contains more than k points to the right of $\overrightarrow{vu}$ that can see the opposite vertex, namely s_1. Suppose that still a k-COD triangulation T exists that includes $\overline{uv}$. Let $\triangle us_iv$ be the triangle in T to the left of $\overrightarrow{vu}$. Assume w.l.o.g. that point s_i lies such that $\overline{us_i}$ intersects $\overline{vs_1}$. Let p_1 and p_2 be two points such that $\triangle s_1p_1p_2$ is in T and it intersects the triangle $\triangle uvs_1$ (possibly, $p_1 = u$ or $p_2 = s_i$). The circle $C(s_1, p_1, p_2)$ includes the whole part of $C(u, v, s_1)$ to the right of $\overrightarrow{vu}$ since p_1 or p_2 lie outside $C(u, v, s_1)$. Moreover, the opposite vertex of $\triangle s_1p_1p_2$, for the points to the right of $\overrightarrow{vu}$, is also s_1. Therefore the order of $\triangle s_1p_1p_2$, under SEESOPP, is at least the order of $\triangle us_1v$, hence cannot be order-k, contradicting the assumption that a k-COD triangulation exists with $\overline{uv}$. ⊠

In the unconstrained case [11], the previous result is complemented with a result stating that if both triangles $\triangle uvs_1$ and $\triangle uvs_2$ (the symmetric triangle to the right of $\overrightarrow{vu}$) are order-k, then $\overline{uv}$ can be completed to an order-k triangulation. The result is shown by presenting an algorithm to compute what the authors call the *greedy triangulation* of the edge.

In our context, the greedy triangulation of an edge $\overline{uv}$ can be defined constructively as follows. See Figure 6 for an example. Let $\overline{uv}$ be a k-COD edge. Take as initial triangulation the constrained Delaunay triangulation of $\mathcal{P}$ and $\mathcal{C}$, removing all the edges that intersect $\overline{uv}$. The remaining empty area is called the *hull* of $\overline{uv}$. We explain how to triangulate this hull. Let s_1 be the point to the right of $\overrightarrow{vu}$ such that the part of $C(u, v, s_1)$ to the right of $\overrightarrow{vu}$ is empty (in the constrained Delaunay sense), and such that $\overline{us_1}$ and $\overline{vs_1}$ do not intersect any constraint. Add the two edges $\overline{us_1}$ and $\overline{vs_1}$ to the triangulation (they may be constraints that are already present). Continue like this recursively for the two edges $\overline{us_1}$ and $\overline{vs_1}$ until the hull of $\overline{uv}$ to the right of $\overrightarrow{vu}$ is completely triangulated. The same procedure is then performed on the left side of $\overrightarrow{vu}$.



(a) (b) (c) (d)

Figure 6: (a) Constrained Delaunay triangulation. (b) Edge $\overline{uv}$ is inserted, the crossing edges from the CDT removed. (c) The first step of the greedy triangulation adds edge $\overline{us_1}$, (d) the second step adds $\overline{us_2}$, triangulating the right part of the *hull* of $\overline{uv}$.

This triangulation has the property that all the triangles used that are not Delaunay have a circumcircle that can only contain points that were already contained in $C(u, v, s_1)$. In the constrained case, this property is not always enough to guarantee that the order of the new triangles is never more than k. For example, in Figure 6, with definition `SEESOPP`, $\triangle uvs_1$ is order 4, whereas $\triangle us_1s_2$ is order 5. However, it is easy to observe that the property remains valid for definitions `PATHCON` and `SEESTRIANG`. This observation, together with Lemma 3, implies the following.

Corollary 1 *Let $\overline{uv}$ be an edge with $u, v \in \mathcal{P}$, let s_1 the point to the left of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left of $\overrightarrow{vu}$, in the constrained Delaunay sense, and such that $\overline{us_1}$ and $\overline{vs_1}$ do not intersect any constraint. Let s_2 be defined similarly but to the right of $\overrightarrow{vu}$. Edge $\overline{uv}$ is a useful k-COD edge under definition* `DEF` $\in \{$`PATHCON`, `SEESTRIANG`$\}$ *if and only if $\triangle uvs_1$ and $\triangle uvs_2$ are k-COD triangles, under definition* `DEF`*.*

Therefore for `PATHCON` and `SEESTRIANG` the same test used for the unconstrained version works in the presence of constraints. For two other definitions it is still possible to test whether an edge is useful or not, using the same idea. The following result is proved for `SEESOPP`, and automatically holds for `CONFEDGE` as well.

Lemma 4 *For definitions* `CONFEDGE` *and* `SEESOPP`*, if $\overline{uv}$ is useful order-k then the greedy triangulation has constrained order k.*

Proof: Let $\overline{uv}$ be a useful order-k edge under `SEESOPP`. Let s_1 be the point to the left of $\overrightarrow{vu}$ defined as in the previous lemmas. Since $\overline{uv}$ is useful order-k, it follows from Lemma 3 that triangle $\triangle uvs_1$ is order-k. Suppose the greedy triangulation of $\overline{uv}$ is not order k. Then there must be some triangle $\triangle p_1s_ip_2$ in the greedy triangulation with order higher than k. Assume w.l.o.g. $\triangle p_1s_ip_2$ lies to the left of $\overrightarrow{vu}$. See Figure 7. The properties of the greedy triangulation imply that the points inside $C(p_1, s_i, p_2)$ are also inside $C(u, v, s_1)$. Since the order of $\triangle p_1s_ip_2$ is higher than the one of $\triangle uvs_1$, the difference must be due to points inside $C(p_1, s_i, p_2)$ that cannot see s_1 but can see s_i. In any *k-COD* triangulation including $\overline{uv}$, there must be some other triangle with s_i as opposite vertex (with respect to points to the right of $\overrightarrow{vu}$). Because of the way the greedy triangulation is defined, any such triangle that

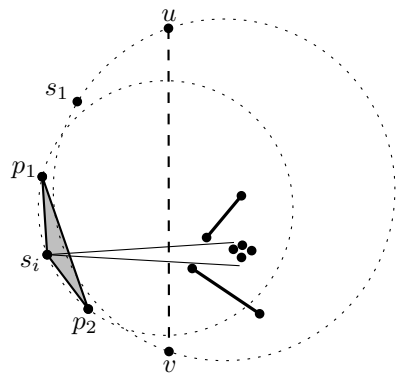


Figure 7: Proof of Lemma 4. If $\triangle p_1 s_i p_2$, part of the greedy triangulation of $\overline{uv}$, is not order-k, no other triangle incident to s_i that faces $\overline{uv}$ can be order-k.

Definition	Order of one triangle	All k-COD triangles
PATHCON	$O(n \log n)$	$O(n^3 \log n)$
SEESTRIANG	$O(n^2)$	$O(n^3 \log n)$
CONFEDGE	$O(n^2)$	$O(n^4)$
SEESVTX	$O(n \log n)$	$O(n^3 \log n)$
SEESOPP	$O(n \log n)$	$O(n^3 \log n)$
SEES3VTX	$O(n \log n)$	$O(n^3 \log n)$
EMPTYQUAD	$O(n \log n)$	$O(n^3 \log n)$

Table 1: Summary of the running times of computing the order of one triangle and computing all order-k triangles, for each of the definitions.

is not $C(p_1, s, p_2)$ has a circumcircle that includes all points inside $C(p_1, s, p_2)$, and since the opposite vertex is the same, it will see at least as many points as counted for $\triangle p_1 s_i p_2$. Therefore its order is at least the order of $\triangle p_1 s_i p_2$, leading to a contradiction. ⊠

3 Computing the order of a triangle

A basic operation when dealing with higher order Delaunay triangulations is determining the order of a triangle. In this section we analyze how efficiently this can be done for each of the proposed definitions. Table 1 contains a summary of the results.

Let $\triangle uvw$ be the triangle whose order should be computed, and let C be its circumcircle. Let P_C be the set of points inside C, and let E_C be the set of constraining edges that are contained in C or intersect its boundary. The running times in this section depend on $|P_C|$, but since $|P_C|$ can be linear in n, (where n is the total number of points and endpoints of constraints), for convenience we will express them in terms of n.

PATHCON We build a point location data structure for the edges in E_C, which allows us to determine for each point inside C, if it lies in the same face of the subdivision induced by the constraints and C as the interior of $\triangle uvw$. Those are the points that can be reached from the triangle, and must be counted. The subdivision is made of parts of constraining edges

and circular arcs (fragments of C). The point location structure can be built in $O(n \log n)$ time, and querying takes $O(\log n)$ time for each point, therefore the total running time is $O(n \log n)$.

SEESTRIANG The points inside C can be in one of three regions, bounded by C and the three edges of $\triangle uvw$. For a point in a given region, seeing $\triangle uvw$ is equivalent to seeing one of the edges of $\triangle uvw$. We process each region separately. Assume the current region is the one bounded by $\overline{uv}$. If $\overline{uv}$ is a constraint, no point is counted. Otherwise, let S be the set of points (including endpoints of constraints) inside that region. For each point in $S \cup \{u, v\}$, we sort the other points around it by angle. This can be done for all the points in $O(n^2)$ time [17]. For each point, we go through the sorted list of points around it and check if at any moment $\overline{uv}$ is visible. We can do this in linear time because we do not need to keep track of the order in which the constraints become visible. At any time we only need to know whether there is any constraint between the point and $\overline{uv}$, so the algorithm only needs to maintain a counter. Therefore the total running time is $O(n^2)$.

CONFEDGE We compute the visibility graph of the whole point set induced by $(P_C \cup E_C)$ in $O(n^2)$ time [17]. To determine if a point must be counted we check if it has a visible point in one of the other two regions of C or can see the opposite vertex of $\triangle uvw$. The total running time is $O(n^2)$.

SEESVTX For each of the vertices of the triangle we compute the visibility polygon, where the edges in E_C and the points in P_C are the obstacles. This can be done in $O(n \log n)$ time. Then we simply count the number of different points that can see some vertex. The total running time is $O(n \log n)$.

SEESOPP We apply the same method as before, but only for the opposite vertex. The total running time is $O(n \log n)$.

SEES3VTX Same as for `SEESVTX`, but we count only the points that see the three vertices of the triangle. The total running time is $O(n \log n)$.

EMPTYQUAD First we compute the points that see the three vertices. These points can be in one of three regions of C. For each region there is a vertex of the triangle that is the opposite vertex. We show how to proceed for the region where the opposite vertex is w, the other two cases are identical. We need to discard the points p such that triangle $\triangle uvp$ is not empty. Let $\triangle uvp$ and $\triangle uvq$ be two triangles, and let α_u (α_v) denote the angle of $\triangle uvp$ at u (at v), and β_u (β_v) the same for $\triangle uvq$. It is easy to see that $\triangle uvp$ contains point q if and only if $\beta_u < \alpha_u$ and $\beta_v < \alpha_v$. Each triangle with u and v as two of its vertices can be represented by a point in the plane using its angles at u and at v as its coordinates. The empty triangles, which are the ones that can create an empty quadrilateral, are the ones lying on the lower-left staircase of the point set. They can be computed in $O(n \log n)$ time by a sweep line algorithm. Therefore the total running time is $O(n \log n)$.

4 Computing all the k-OCD triangles

Another useful operation related to higher order Delaunay triangulations is computing all the order-k triangles. For example, this is a fundamental step when triangulating polygons optimally for order-k Delaunay triangulations [21]. Table 1 summarizes our results for the constrained order definitions.

The general approach will be to generate all candidate edges, and then, for each edge, we will find all the order-k triangles that are incident to it. In principle there are $O(n^2)$ candidate edges to test.

We explain how to compute all the order-k triangles adjacent to one edge $\overrightarrow{uv}$, which lie to the right of $\overrightarrow{uv}$, assumed not to be a constraint (if it is, the algorithms can be simplified). The triangles laying on the other side can be found in a symmetric way.

PATHCON We describe an algorithm to compute, for each third point s to the right of $\overrightarrow{uv}$, the number of points to the left of $\overrightarrow{uv}$ that can reach $\overline{uv}$. Then the same must be done with respect to the points to the right of $\overrightarrow{uv}$, and the two results must be combined. We only explain the first part, the rest is symmetric and straightforward.

Consider the connected components defined by the constraining edges (recall that they may share endpoints, so some components may be composed of several edges). In a preprocessing step, we will identify all these components and will keep only the ones with some endpoint in the halfplane to the left of $\overrightarrow{uv}$.

We will sweep a circle C through u and v that will start as the halfplane to the left of $\overrightarrow{uv}$ and will slide, always touching u and v, until it becomes the halfplane to the right of $\overrightarrow{uv}$. The event points will be the points and some of the endpoints of the components. During the sweep, points that were counted after the previous event may stop to be counted for the next one, but never the other way around. Note that there are two reasons for a point to stop to be counted. The first is that the new circle position does not include it. The second is that the region where the point lies is now disconnected from $\overline{uv}$ inside the circle. The first type is usual for the standard higher order Delaunay definition, therefore we concentrate on the second type, which is specific of this definition. Our goal is to compute, as we sweep the circle through u, v and a third point in S, all the points that stop to be reachable from $\overline{uv}$.

While sweeping the circle, the events (from now on we only consider the second type) will not be all the endpoints of the constraints but only the *spikes*. Let p be the endpoint of a constraint that belongs to some connected component. p is a *spike* if (i) p is *not* the first point of the component that is touched when sliding C until touching the component; (ii) all the edges incident to p have both endpoints to the same side of line ℓ, where ℓ is the line tangent to $C(u, v, p)$ at p. The second condition states that p must have, roughly, the shape of a *spike*, with respect to u and v.

Spikes have the property that part of $C(u, v, p)$ always defines a closed region R inside $C(u, v, p)$ such that once the circle being slid touches p, the interior of R gets disconnected from $\overline{uv}$ (unless the region of some previous spike of the same component contains R). In other words, the points inside that region do not have to be counted anymore once the circle reaches p. Each region is made of a polygonal chain and a circular arc, which is a part of $C(u, v, p)$. Figure 8 shows an example. A spike can define up to two regions, one on each side of p. In the way we will process the spikes, explained below, only one of them can become disconnected from $\overline{uv}$ at the event of that spike.

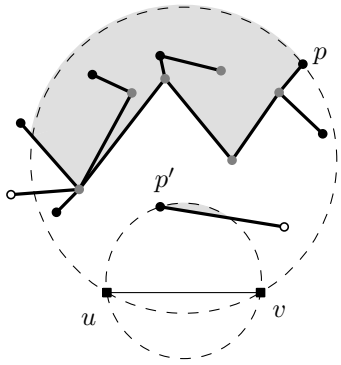


Figure 8: Example showing the regions defined by two spikes, p and p'. All the black, round, vertices are spikes.

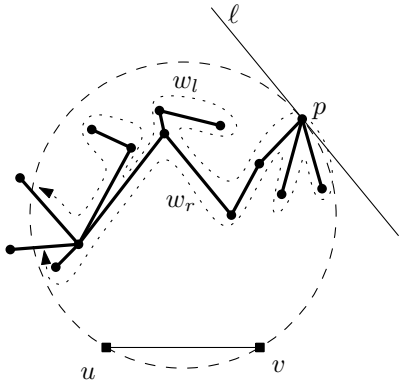


Figure 9: Tandem walk to identify the region to become disconnected, w_l in the example.

The sweep algorithm identifies for each circle $C(u, v, p)$, for a spike p, the region whose points must not be counted anymore. Every time an event occurs, we proceed as follows. We have to identify which of the two regions that the spike defines is the separated one, that is, the one that does not contain $\overline{uv}$. We can do this by walking around the edges, in both regions simultaneously, until we find the first edge on the boundary of the region that intersects $C(u, v, p)$.

We start by identifying the two initial edges of each region (see Figure 9). With some abuse of notation, we refer to the regions as *to the left* and *to the right* of p. We explain how to walk through the left region. The right one is symmetric. The starting edge is the first edge incident to p encountered when rotating ℓ around p in counterclockwise direction. The edge can be found by examining all edges incident to p. Every edge will be examined at most twice (once for each endpoint), because once the region is identified, p will not be processed again. The total time spent on this (for all spikes defined by $\overline{uv}$) is linear. Then we walk along the edges, always choosing as the next edge the one with smallest angle with the current one. This can be obtained in constant time by storing the components using some standard data structure like a doubly-connected edge list.

We do a walk in both regions at the same time, that is, we alternate between traversing one edge on each side of p. Every time we go to a new edge we check if it intersects C. When such an edge is found, we can check if the region found contains $\overline{uv}$ by checking if the circular

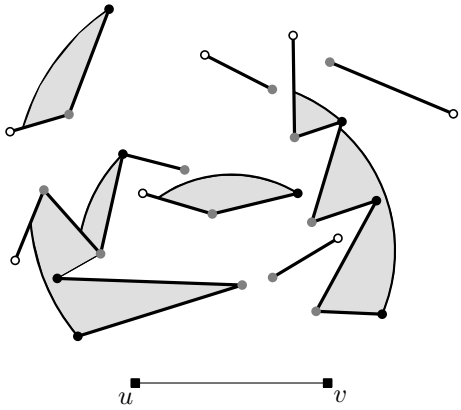


Figure 10: Example showing all the regions identified on one side of an edge $\overline{uv}$.

arc that bounds the region contains u and v. If it does not, we found the disconnected region, otherwise, the region that became disconnected is the other one. In that case we continue traversing the boundary of the other region until we find an edge intersecting the circle. At that point all the edges of the disconnected region have been identified.

Some of the vertices of the region that become disconnected can be spikes, but given the order in which points are processed (starting from a large circle that becomes smaller), their regions will be included in the current region, so they do not need to be considered and their events can be skipped. Hence we remove them from the event queue.

Regarding the running time of the sweep, there are $O(n)$ events. Handling each event involves traversing the edges that define a region. For every spike we process, the traversal takes time proportional to the number of edges bounding the region that becomes disconnected. The time spent on traversing the other region can be charged to the edges of the region that will become disconnected. Hence the overall time complexity of identifying all the regions is $O(n)$. The total running time of the sweep algorithm is therefore $O(n \log n)$.

Once the sweep is over we will have a region associated with some of the spikes (see Figure 10). Next all the points inside each region must be identified. The regions are disjoint and are made of line segments and circular arcs. The points can be found in total time $O(n \log n)$ by using a point location data structure able to handle circular arcs, for example the one from [18]. The overall running time for one edge $\overline{uv}$ is $O(n \log n)$, and $O(n^3 \log n)$ time is needed to identify all the order-k triangles.

SEESTRIANG, SEESVTX, SEESOPP, SEES3VTX and EMPTYQUAD We will sweep a circle in a way similar to the one used in the previous algorithm. All these definitions are based on visibility between the points inside the circle and some elements (an edge, a vertex, etc.) of the triangle of which the order is being computed. They have the property that once a vertex is counted (that is, it sees the part of the triangle in question), it will be counted until it stops being inside the current circle. Given the visibility graph of all points and constraint endpoints, a vertex can be checked to determine if it must be counted in $O(1)$ time. In the case of `EMPTYQUAD` we can first discard all the points that create a non-empty triangle with $\overline{uv}$, in $O(n \log n)$ time, as explained in the previous section. A simple circle sweep, where every point and endpoint of a constraint defines an event, is enough to keep track of the order of $\triangle uvw$, for each possible third point w to the right of $\overrightarrow{uv}$. After obtaining

Definition	Order of one triangle
PATHCON	$O(n)$
SEESTRIANG	$O(n)$
CONFEDGE	$O(n \log n)$
SEESVTX	$O(n)$
SEESOPP	$O(n)$
SEES3VTX = EMPTYQUAD	$O(n)$

Table 2: Summary of the running times for computing the order of one triangle for polygons.

the possible third points, only the ones that define empty triangles must be selected. Again, this can be done in $O(n \log n)$ time. Therefore the running time for one edge $\overline{uv}$ is $O(n \log n)$ (assuming the visibility graph is precomputed). It follows that all the order-k triangles can be found in $O(n^3 \log n)$ time.

CONFEDGE For this definition we can apply the algorithm used for the previous definitions, but in this case checking if a point must be counted takes more time. This is because every time a third point w to the right of $\overrightarrow{uv}$ is processed, many of the points inside C that can see w will be counted from the next step on. Hence linear time is required to find these points. The total running time, for one edge $\overline{uv}$, increases to $O(n^2)$, leading to $O(n^4)$ time to find all the order-k triangles.

5 Improved algorithms for polygons

When what needs to be triangulated is a simple polygon, instead of a point set, the computation of the order of one triangle can be done more efficiently. The results are summarized in Table 2.

In what follows, we denote by P the polygon defined by the constraints, $\triangle uvw$ is the triangle whose order should be computed, and $C = C(u, v, w)$ is its circumcircle.

PATHCON Assume that $\overline{uv}$ is horizontal, u is to the left of v, and w is below the line through u and v. We explain how to find the vertices of P that must be counted for the order of $\triangle uvw$ that are in the region of C to the left of $\overrightarrow{uv}$. The procedure for the other two regions is symmetric. The idea is to walk through the polygon boundary while keeping track of what part of the boundary of C can still be reached from $\triangle uvw$.

We begin by adding dummy vertices where the polygonal boundary from u to v intersects C. They divide the polygonal boundary into chains that are outside and inside C. We call them internal and external chains, respectively. See Figure 11 (a). Between two consecutive chains there is a dummy vertex. It is assumed to be both the end of a chain and the beginning of a chain. Vertices on external chains will never be counted, but vertices on internal chains may also not be counted.

The first step is to remove the chains of the polygon that wind around $\triangle uvw$, since they can never be reachable. See Figure 12 for an example. Consider a half-line that extends from some point of the triangle vertically downwards. Whenever a chain crosses this half-line from left to right, we add one to a counter that maintains the winding. If a chain crosses the

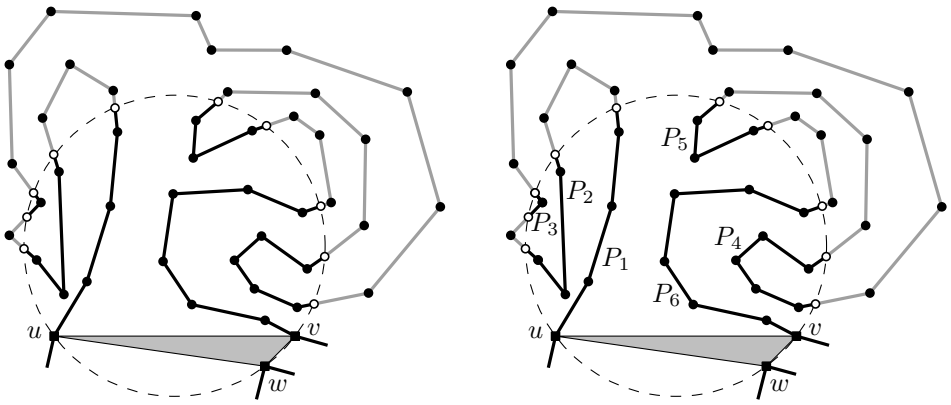


Figure 11: (a) Example with six internal chains (black edges) and five external chains (gray edges). Dummy vertices are shown white. (b) Internal chains numbered by occurrence along P.

half-line from right to left, we subtract one from this counter. Only chains that have winding count 0 and are interior to C can contain vertices that are counted for the order.

Denote by $P_1, \ldots, P_m$ the sequence of chains on the boundary of P from u to v that are interior to C and have a winding count of 0. Some of the chains of $P_1, \ldots, P_m$ have a clockwise orientation: when considering C clockwise from u to v, in a clockwise chain we first encounter its start dummy vertex and then its end dummy vertex (chains P_1, P_3, P_5, and P_6 in Figure 11 (b)). The other two chains have a counterclockwise orientation. It is easy to see that vertices on counterclockwise chains cannot be path-connected to $\triangle uvw$ in the interior of C and P, and hence their vertices will not be counted. But not all clockwise chains have vertices that are counted, like P_3 in the figure. We observe:

Observation 2 *For any prefix of chains $P_1, \ldots, P_i$ (see Figure 13):*

(i) *The clockwise chains that are path-connected to $\triangle uvw$ inside C with respect to $P_1, \ldots, P_i$ only, appear clockwise along C by increasing index.*

(ii) *The counterclockwise chains that are path-connected to $\triangle uvw$ inside C with respect to $P_1, \ldots, P_i$ only, appear counterclockwise along C by increasing index.*

(iii) *The clockwise chains from (i) are all counterclockwise with respect to the counterclockwise chains from (ii).*

We call the chains from (i) and (ii) *alive* for $P_1, \ldots, P_i$. In our algorithm, alive clockwise chains will be kept on a stack S_1 with the highest indexed one at the top. Similarly, alive counterclockwise chains are kept on another stack S_2 with the highest indexed one at the top.

Let the most clockwise dummy vertex of an alive clockwise chain be d_1, it is the last vertex of the chain on the top of S_1. Similarly, the most counterclockwise dummy vertex of an alive counterclockwise chain is d_2, see Figure 13. If S_1 is empty, then we define $d_1 = u$, and if S_2 is empty, then $d_2 = v$. The important observation is that the next alive chain P_j after P_i must appear between d_1 and d_2. So we can ignore any chain after P_i if it does not appear between d_1 and d_2. If the next chain to appear between d_1 and d_2, P_j, is clockwise, then it will extend S_1. Furthermore, it may kill some of the alive counterclockwise chains. These necessarily are

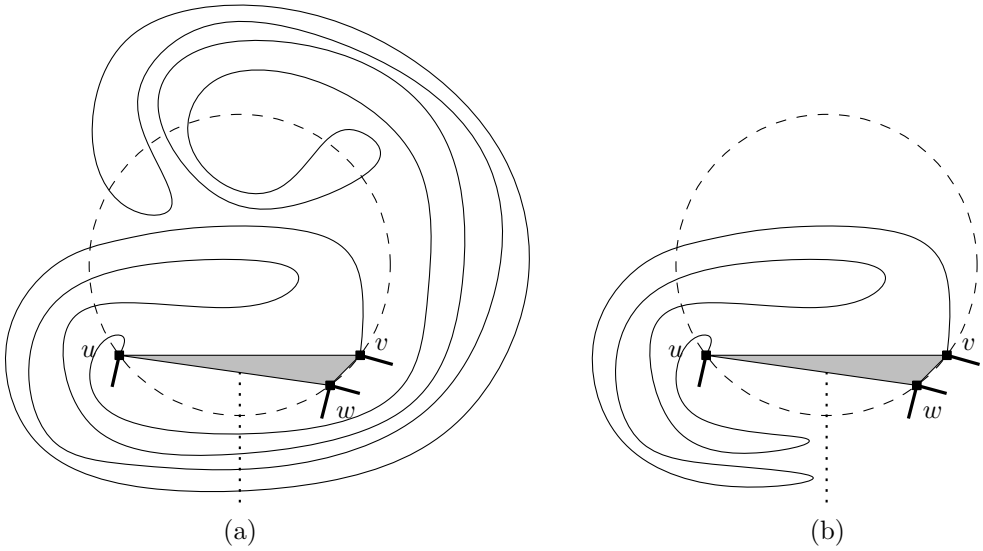


Figure 12: (a) Original polygon. (b) Polygon after removing parts that *wind around* $\triangle uvw$.

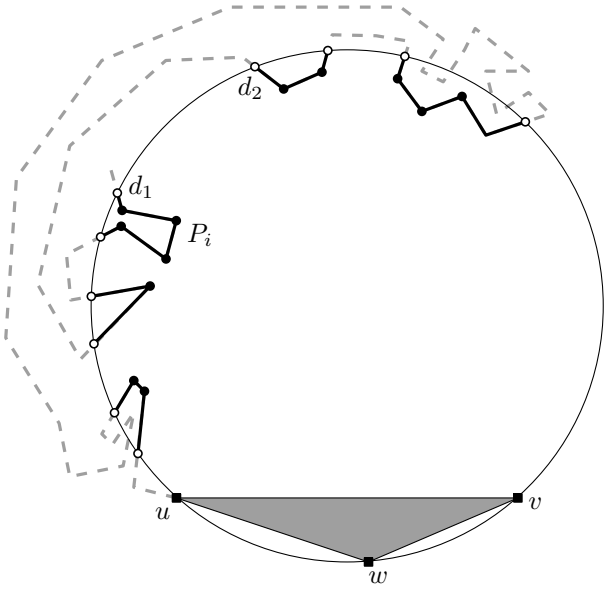
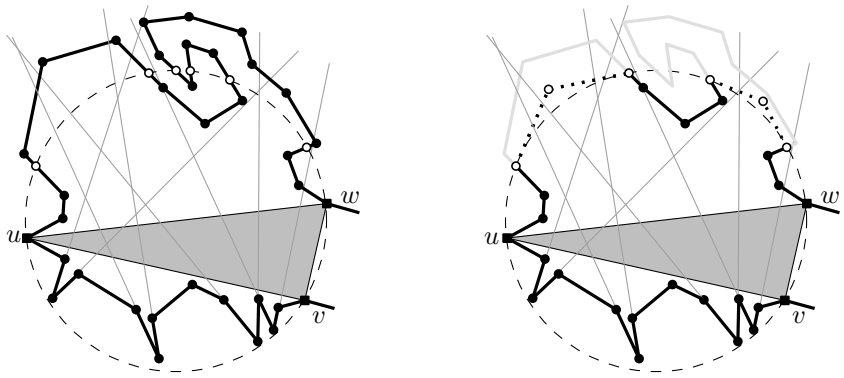


Figure 13: Three chains on the stack S_1 for clockwise chains and two chains on the stack S_2 for counterclockwise chains (all five black).



(a) (b)

Figure 14: (a) Part of the polygon, showing the visibility cones of some of the points that must be tested. (b) Polygon after simplification, unreachable chains were removed (shown in light gray) and dummy vertices and edges (dotted) were added to connect the remaining chains.

at the top of the stack S_2. Symmetrically, if P_j is counterclockwise, then it will extend S_2 and possibly kill some chains from S_1.

The implied incremental algorithm of treating the chains $P_1, \ldots, P_m$ in order eventually leads to a stack S_1 that contains exactly the chains whose vertices are counted. Stack S_2 will be empty. It is clear that this algorithm runs in linear time.

SEESTRIANG For each of the three regions (see Section 3) where a point can lie we must find the points that can see one of the edges of $\triangle uvw$. Given a triangulation of the polygon, the visibility polygon with respect to an edge can be computed in linear time [13]. Therefore if a linear-time algorithm is used for triangulating the polygon [3], the order of $\triangle uvw$ can be computed in linear time.

CONFEDGE First we simplify P as follows. We apply the algorithm for `PATHCON` to find the chains of P that are in the same face as $\triangle uvw$ inside C. We do this for all three regions inside C and outside $\triangle uvw$. If any chain in C does not contain any real vertices (it connects two dummy vertices), then we remove that chain. For the remaining chains, we connect the dummy vertices of two consecutive chains by two edges outside C; these two edges are tangent to C at the dummy vertices and have the intersection point of the tangents as one more dummy vertex, see Figure 14. This way, the reachable chains of P are separated by three dummy vertices (if the two dummy vertices where the parts end are too far apart, two new dummy vertices are needed in between). The resulting augmented polygon P' is simple and has $O(n)$ vertices.

For every non-dummy vertex of P', we need to know what it can see "through" $\triangle uvw$. Let s be a non-dummy vertex of P' between u and v (clockwise from u and counterclockwise from v). Then we compute the largest interval of $\overline{uv}$ that s can see. If it is non-empty, we perform ray shooting queries from s through the two endpoints of this interval, and find two edges of P' that lie clockwise from v and counterclockwise from u.

Observation 3 *(i) If the rays from s hit the same edge, then s is not involved in any conflicting edge.*

(ii) If the rays from s hit two different edges and in between there are only dummy vertices, then s is not involved in any conflicting edge.

(iii) In all other cases, s is involved in at least one conflicting edge.

By the observation above, and the fact that we never have more than four consecutive dummy vertices in P', we can test if s is involved in a conflicting edge in constant time after the ray shooting.

Computing the largest interval of $\overline{uv}$, $\overline{vw}$, and $\overline{uw}$ for all non-dummy vertices of P' can be done by first triangulating P', and then applying the algorithm in [13]. It allows us to find, for each vertex, the subsegment of $\overline{uv}$ that is visible in $O(\log n)$ time. Ray shooting also takes $O(\log n)$ time per query [13]. Hence, we can determine the number of vertices of P' (and therefore of P) that are involved in a conflicting edge in $O(n \log n)$ time in total.

SEESVTX, SEESOPP and SEES3VTX The visibility polygon of a point inside a simple polygon can be computed in linear time [9]. Therefore the order of a triangle, under these three definitions, can be found in linear time as well.

6 First order constrained Delaunay triangulations

The class of first order Delaunay triangulations (for the unconstrained case) has a special structure that allows many measures to be optimized efficiently [11, 23]. For a given point set, if we take all the edges that are present in any first order Delaunay triangulation of the point set (fixed edges), they form a subdivision of the convex hull into triangles and convex quadrilaterals. It follows that to generate any 1-OD triangulation of the point set it is enough to choose one of the two diagonals of each quadrilateral.

In this section we study the special structure of first order constrained Delaunay triangulations. For `SEESOPP` and the stronger definitions, we show that in the presence of constraints, 1-COD triangulations have the same structure as first order Delaunay triangulations, hence all the measures that can be optimized for non-constrained first order Delaunay triangulations can also be optimized when constraints are present. For the `SEES3VTX` definition we show that the structure becomes more complex, and the existing techniques cannot be applied. As mentioned in Observation 1, the constrained order under the `EMPTYQUAD` definition can be much larger than the order under the standard definition, so the first order Delaunay structure is not preserved either.

We begin by observing that when $k = 1$, two pairs of definitions become equivalent, whereas the other definitions remain different for $k = 1$, as illustrated by Figure 15.

Lemma 5 *For any point set $\mathcal{P}$ and set of constraints $\mathcal{C}$, we have $T_1(P, \mathcal{C}, \texttt{SEESVTX}) = T_1(P, \mathcal{C}, \texttt{SEESTRIANG})$*

Proof: The inclusion $T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX})$ follows from Lemma 1. For the other direction, let $t = \triangle uvw$ be an order-1 triangle under `SEESVTX`, and let C be its circumcircle. If t is not order-1 under `SEESTRIANG`, there must be a point $p \in \mathcal{P}$ inside C that sees t but does not see a vertex of t. Let q be a point on $\overline{uv}$ such that $\overline{pq}$ does not intersect any constraint. Such a point exists because p can see t. Assume w.l.o.g. that vertex

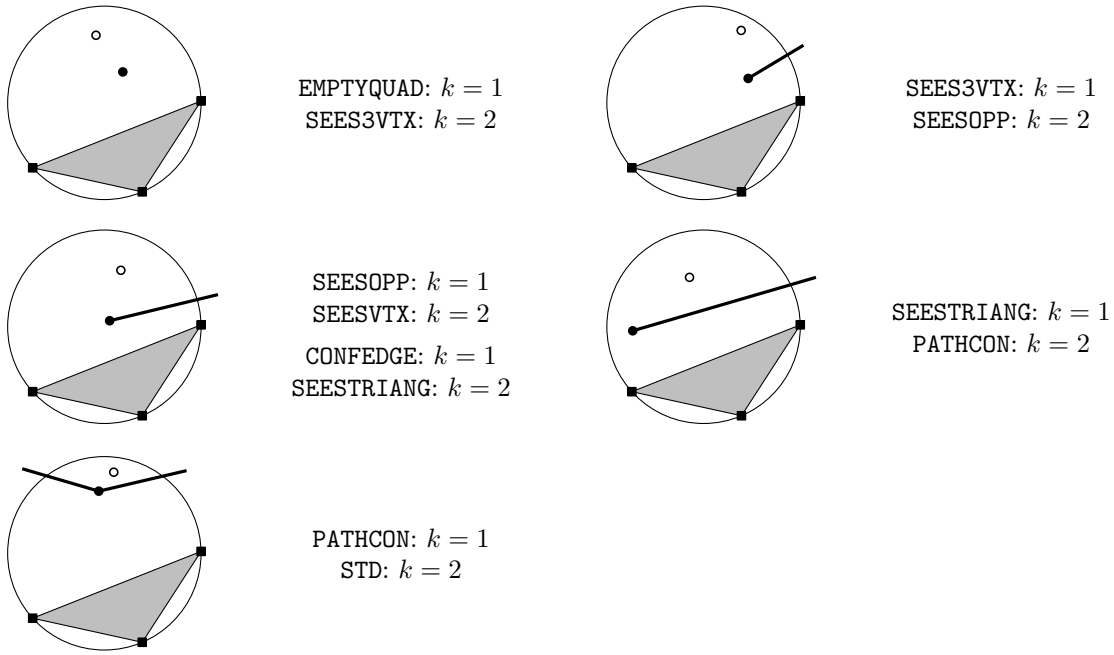


Figure 15: Except for {SEESTRIANG, SEESVTX} and {CONFEDGE, SEESOPP}, the other definitions remain different when $k = 1$. The examples shown here illustrate the pairs of definitions whose inclusions are proper for $k = 1$.

u lies to the right of $\overrightarrow{pq}$ and v to its left. Since p cannot see u, there must be at least one constraint intersecting $\overline{pu}$, and each such constraint has one endpoint in $\triangle upq$. Let c_1 be an endpoint in $\triangle upq$ of the constraint such that $\angle vuc$ is minimum. Then c_1 can see vertex u and is inside C, thus must be counted under SEESVTX. Proceeding analogously for the left side, there is a point c_2, endpoint of some constraint inside $\triangle pvq$ visible from vertex v, we observe that $c_2 \neq c_1$. Therefore the order of t under SEESVTX is at least two, giving a contradiction. ⊠

Lemma 6 *For any point set $\mathcal{P}$ and set of constraints $\mathcal{C}$, we have $T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESOPP}) = T_1(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE})$*

Proof: The result follows from the fact that if at most one point is allowed inside the circumcircle of each triangle, then any conflicting edge must be created with the opposite vertex. Hence the points that can see the opposite vertex and the ones that can create a conflicting edge are the same for $k = 1$. ⊠

The following corollary summarizes how the hierarchy of definitions looks when $k = 1$.

Corollary 2 *For first order constrained Delaunay triangulations of a point set $\mathcal{P}$ with constraining edges $\mathcal{C}$ (which can define a polygon), the following inclusion relations hold:*
$T_1(\mathcal{P}, \mathcal{C}, \texttt{STD}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{PATHCON}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESTRIANG}) = T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESVTX}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{CONFEDGE}) = T_1(\mathcal{P}, \mathcal{C}, \texttt{SEESOPP}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{SEES3VTX}) \subseteq T_1(\mathcal{P}, \mathcal{C}, \texttt{EMPTYQUAD})$

We now study the structure (in relation to fixed edges) of first order constrained Delaunay triangulations. We begin with the following definition.

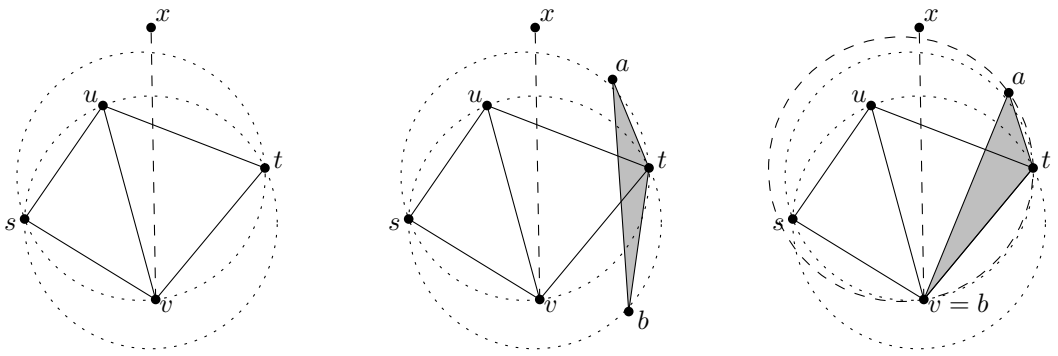


Figure 16: Left: no extra point x can lie inside the two dotted circles. Center and right: if some triangulation contains an edge $\overline{vx}$ that crosses $\overline{ut}$, there must be a triangle whose opposite vertex is t and whose circumcircle includes $\{u, s\}$, both visible from t.

Definition 3 *A triangle t in $T \in T_k(\mathcal{P}, \mathcal{C}, \texttt{DEF})$ is* single-flippable *if at least two of its edges are present in every triangulation of $T_k(\mathcal{P}, \mathcal{C}, \texttt{DEF})$.*

Lemma 7 *The triangles of any first order constrained Delaunay triangulation of a point set $\mathcal{P}$ and constraint set $\mathcal{C}$, under definition* SEESOPP *or stronger, are single-flippable.*

Proof: Suppose $\triangle usv$ and $\triangle uvt$ are triangles in the constrained Delaunay triangulation of $\mathcal{P}$, such that $\{u, s, v, t\}$ forms a flippable quadrilateral, that is, a quadrilateral of which both diagonals $\overline{uv}$ and $\overline{st}$ can be used to construct a 1-OCDT. We show that for SEESOPP (and hence also for all the stronger definitions), the four edges of the quadrilateral are fixed.

Since $\triangle usv$ and $\triangle uvt$ are constrained Delaunay triangles, the circles $C(u, s, v)$ and $C(u, v, t)$ must be empty (in the constrained Delaunay sense). Moreover, $\overline{st}$ is a useful order-1 edge, hence it follows from Lemma 3 that $\triangle ust$ and $\triangle vts$ are order-1 triangles. Since the circles $C(u, s, t)$ and $C(v, t, s)$ already contain one point each (in the SEESOPP sense), namely v and u, respectively, no other point that sees the opposite vertex can be inside them.

Suppose edge $\overline{ut}$ is not fixed (the other cases are symmetric). Then there is a 1-OCD triangulation that contains an edge crossing $\overline{ut}$.

Assume that edge is $\overline{vx}$, for x some vertex outside $C(u, s, v)$ and $C(u, v, t)$. In the triangulation that contains $\overline{vx}$ there must exist some triangle $\triangle abt$, such that $\overline{ab}$ intersects the interior of $\triangle uvt$. See Figure 16. Assume first that $b = v$. Then a must lie outside $C(v, t, s)$, otherwise $\triangle vts$ would not be order 1 (because v can see both u and a). But then $C(a, b, t)$ contains both u and s. Both u and s can see the opposite vertex t, hence the order of $\triangle abt$, under SEESOPP or any stronger definition, is at least two. If $b \neq v$, $C(a, b, t)$ is even larger, and must contain u and s as well. Therefore no edge like $\overline{vx}$ can exist in an order-1 triangulation. ⊠

The previous lemma implies that for the SEESOPP definition, all the optimization techniques that exist for first order Delaunay triangulations [11, 23] can be applied in the presence of constraints. Examples of measures that can be minimized efficiently include maximal area triangle, maximal triangle angle, total edge length, number of local minima, angle between triangle normals and number of convex vertices.

Corollary 3 *For definitions* SEESOPP *or stronger, all the existing optimization techniques*

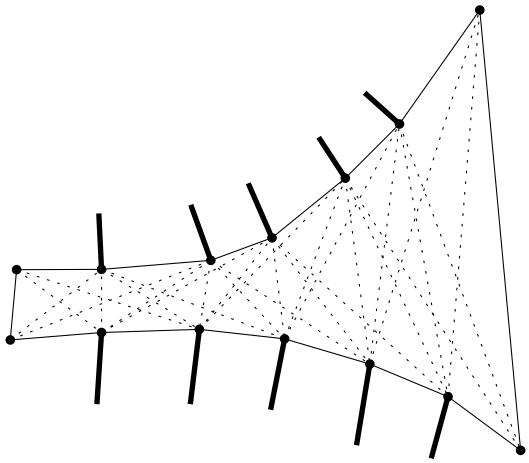


Figure 17: For the SEES3VTX definition, chains of overlapping flippable quadrilaterals can exist. Thick edges are constraints. All crossing-free combinations of the dotted edges can be completed to an order-1 constrained Delaunay triangulation.

for first order Delaunay triangulations can be applied to first order constrained Delaunay triangulations.

The structure of first order Delaunay triangulations is not preserved when the definition SEES3VTX is used. Figure 17 shows an example of the kind of structure one can get, comprised of a chain of overlapping flippable quadrilaterals. The quadrilaterals are not independent and the polygon of fixed edges can have linear size, hence the techniques of [11, 23] cannot be applied.

7 Concluding remarks

In the context of higher order Delaunay triangulations, we proposed seven different definitions of the order of a triangle that take into account a set of constraining edges. This constitutes an attempt to extend the concept of *constrained Delaunay triangulations* to higher order Delaunay triangulations. The proposed definitions can be seen as natural generalizations of the idea of *order* of a triangle. They define a hierarchy (with one exception) that goes from the standard order definition to a very permissive definition that counts much fewer points than the original one. In general it cannot be stated which definition is the best one, and which one to choose will probably depend on the application. Several theoretical properties of the different definitions were studied.

For each definition we presented algorithms to compute the order of one triangle and to find all the order-k triangles of a point set with constraining edges. These are basic problems that need to be solved for most implementations of higher order Delaunay triangulations. For the special case of triangulations of polygons we provided faster algorithms that allow to compute the order of a triangle in linear time for all but one definition.

Furthermore, we showed that for $k = 1$, several of the definitions preserve the structure present in (unconstrained) first order Delaunay triangulations. This is important from a practical point of view because it makes all the tools for optimizing first order Delaunay triangulations available for the constrained version as well.

One of the most interesting problems left open is analyzing the number of useful k-COD edges. For the unconstrained case it is known that this number is $O(kn)$, whereas for the constrained case we only have the trivial upper bound of $O(n^2)$. It is possible that the $O(nk)$ bound also holds in the presence of constraints, at least for some of the definitions, but it is unclear how to prove it. If such a result could be proven, this would imply, for the four definitions for which the useful order of an edge can be tested efficiently, that the asymptotic running time of the algorithms to find all the order-k triangles can be reduced considerably for small values of k. This would be an important improvement, given that the smallest values of k are most interesting [7].

References

[1] B. Ben-Moshe, M. J. Katz, J. S. B. Mitchell, and Y. Nira. Visibility preserving terrain simplification- an experimental study. *Comput. Geom. Theory Appl.*, 28:175–190, 2004.

[2] M. Benkert, J. Gudmundsson, H. Haverkort, and A. Wolff. Constructing interference-minimal networks. In J. Wiedermann, J. Stuller, G. Tel, J. Pokorný, and M. Bieliková, editors, *Proc. 32nd Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'06)*, volume 3831 of *Lecture Notes in Computer Science*, pages 166–176. Springer-Verlag, 2006.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[4] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[5] L. De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Comput. Graph. Appl.*, 9(2):67–78, Mar. 1989.

[6] L. de Floriani, P. Magillo, and E. Puppo. Applications of computational geometry to geographic information systems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 333–388. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[7] 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.

[8] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1990.

[9] H. ElGindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *J. Algorithms*, 2:186–197, 1981.

[10] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. In *Proc. 6th Annu. Conf. Computer graphics and Interactive Techniques*, pages 199–207, 1979.

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

[12] J. Gudmundsson, H. Haverkort, and M. van Kreveld. Constrained higher order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 30:271–277, 2005.

[13] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[14] A. Koch and C. Heipke. Semantically correct 2.5D GIS data – the integration of a DTM and topographic vector data. *ISPRS Journal of Photogrammetry & Remote Sensing*, 61:23–32, 2006.

[15] D. Mark. Network models in geomorphology. In M. G. Anderson, editor, *Modelling Geomorphological Systems*, chapter 4, pages 73–97. John Wiley & Sons, 1988.

[16] M. Neamtu. Delaunay configurations and multivariate splines: a generalization of a result of B. N. Delaunay. *Trans. Amer. Math. Soc.*, 359(7):2993–3004, 2007.

[17] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[18] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, July 1986.

[19] L. Scarlatos and T. Pavlidis. Hierarchical triangulation using terrain features. In *Proc. 1st Conference on Visualization*, pages 168–175, 1990.

[20] B. Schneider. Geomorphologically sound reconstruction of digital terrain surfaces from contours. In *Proc. 8th Int. Symp. on Spatial Data Handling*, pages 657–667, 1998.

[21] R. I. Silveira and M. van Kreveld. Optimal higher order Delaunay triangulations of polygons. *Accepted for publication, Comput. Geom. Theory Appl.*, 2007.

[22] P. M. van der Poorten and C. B. Jones. Characterisation and generalisation of cartographic lines using Delaunay triangulation. *Int. J. Geographical Information Science*, 16:773–794, 2002.

[23] M. van Kreveld, M. Löffler, and R. I. Silveira. Optimization for first order Delaunay triangulations. In F. Dehne, J.-R. Sack, and N. Zeh, editors, *Proc. 10th Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 175–187, 2007.

[24] Q. Zhu, Y. Tian, and J. Zhao. An efficient depression processing algorithm for hydrologic analysis. *Computers & Geosciences*, 32:615–623, 2006.