Construction Of The Constrained Delaunay Triangulation Of A Polygonal Domain

Reinhard Klein [1]

A fast and easy to implement divide-and-conquer algorithm is presented for the construction of the Constrained Delaunay triangulation of a polygonal domain. The algorithm simplifies the complicated merging step inherent to divide-and-conquer algorithms for the computation of triangulations. Furthermore, no triangles are computed outside the valid region of the domain. A grid structure accelerates the computation of the visibility among vertices with respect to the boundary polygons as well as the computation of Constrained Delaunay triangles.

Introduction and previous work

This algorithm was motivated by the development of an adaptive triangulation algorithm for trimmed parametric surface patches [Kle94], [Ke95]. In the first step of this algorithm an initial Constrained Delaunay triangulation of the polygonal domain of the trimmed parametric surface has to be computed. Later on further vertices are inserted into this triangulation until the corresponding triangulation of the surface approximates the surface itself with sufficient precision. This is not the only application of the algorithm. Further examples are terrain modeling [DFP85], surface interpolation [Rip92] and finite element analysis [HL88].
Polygonal domains in two dimensions can also be considered as planar straight line graphs (PSLG). Many algorithms have been developed to compute the Constrained Delaunay triangulation of PSLGs. Lee and Lin [LL86] published in 1986 a divide-and-conquer algorithm based on a cutting theorem of Chazelle. They proofed a worst case time complexity for their algorithm of $O(nlogn)$ for polygons, where n is the number of vertices of the polygons. De Floriani and Puppo developed in 1988 an incremental algorithm with a worst case complexity $O(mn^2)$ for PSLGs, where m is the number of constrained edges and n is the number of vertices.

[1] Wilhelm-Schickard-Institut, GRIS, Universität Tübingen, Germany

In 1989 Chew [Che89] showed that using a divide-and-conquer algorithm the Constrained Delaunay triangulation of a PSLG can be computed in $O(nlogn)$ time. In 1993 Piegl and Richard [PR93] used a shelling technique and a uniform grid as acceleration structure to compute the Constrained Delaunay triangulation of polygonal domains. Excess triangles generated during the triangulation process contained in the convex hull of the polygonal domain but not in the domain itself are detected by a special labeling procedure and removed from the output.
Examples of algorithms specialized to compute the Constrained Delaunay triangulation of polygons are described in [AGSS89], [Cha91], [KKT90], [CTV89]. All these algorithms take advantage of the special structure of polygons and are therefore even more efficient than the algorithms described above, which also handle the general case of a PSLG.
The general idea of the algorithm presented here is also to exploit the special structure of polygonal domains. The computation of Constrained Delaunay triangles contained in the convex hull of the domain but not in the domain itself is avoided. Further a divide and conquer paradigma in combination with a uniform grid data structure accelerates the computation of Delaunay triangles and reduces the number of visibility tests between vertices as well as the number of tests to determine if an edge is contained in the domain or not. Instead of performing the test against all edges of the bounding polygons, the tests have to be performed only for the edges of the subpolygons due to the divide and conquer approach.
The algorithm is very fast and easy to implement.

Basic definitions

Definition 1 (Planar domain) *A* planar domain $\Omega \subset \mathbb{R}^2$ *is defined by a set* $B = \{b_0, \cdots, b_N\}$ *of* boundary polygons. *The set* B *includes an exterior boundary* b_0 *and interior boundaries (holes)* $b_1, \cdots b_N$. *Each boundary* b_i *consists of a finite number* (≥ 3) *of oriented line segments, the* domain edges $e_{i0}, \cdots, e_{in_i}$ *and is defined by a set of* boundary nodes $V \supset V_i = \{v_{i0}, \cdots, v_{in_i}\}$. *Every boundary polygon* b_i *defines a region* Ω_i. *The interior of these regions* $\mathring{\Omega}_i$ *is located on the left side of the oriented boundary polygons. In order for* Ω *to be a well-defined finite domain, the following conditions must hold true (by* $\complement\,\mathring{\Omega}_i$ *we denote the complement of* $\mathring{\Omega}_i$*):*

$$
\begin{aligned}
\complement\,\mathring{\Omega}_i \quad &\subset \quad \mathring{\Omega}_0,\ 1 \leq i \leq N, \\
\complement\,\mathring{\Omega}_i \quad &\cap \quad \complement\,\mathring{\Omega}_j = \emptyset,\ 1 \leq i, j \leq N,\ i \neq j, \\
b_i \quad &\cap \quad b_j \subset V,\ 0 \leq i, j \leq N,\ i \neq j.
\end{aligned}
$$

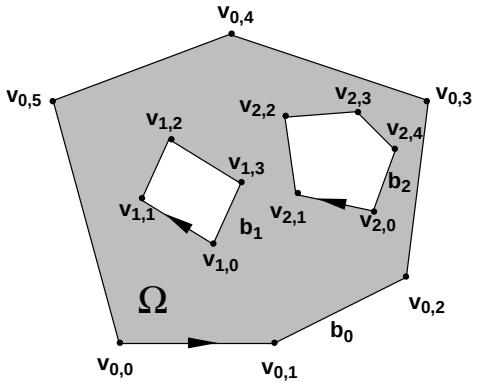

Figure 1: A simple planar domain Ω, described by the outer boundary b_0 and two inner boundaries b_1 and b_2. The edges of the boundaries are oriented in such a way, that the interior of the domain is located on their left.

Remark 2 *A planar domain is a planar straight line graph (PSLG) $G = (V, E)$, where V is the set of boundary vertices and E is the set of boundary edges.*

Definition 3 (Triangulation of a planar straight line graph) *For any PSLG $G = (V, E)$, a triangulation $T(G)$ of G is a PSLG $G' = (V, E')$, where $E \subset E'$, such that no edges can be added to E' without intersecting an existing edge $e \in E$.*

Definition 4 (Constrained Delaunay triangulation) *For any PSLG $G = (V, E)$ the Constrained Delaunay triangulation (CDT) of G, denoted by $CDT(G)$ is a triangulation $T(G) = (V, E')$, where $E \subset E'$, in which the circumcircle of each face or triangle $\Delta(v_i v_j v_k)$ denoted by $C(v_i, v_j, v_k)$ does not contain in its interior any other vertex of V which is visible from the vertices v_i, v_j, v_k of the triangle. The vertices u and v, $u, v \in V$ are visible from each other with respect to E if the line segment uv does not intersect an edge of E. The edges of the set $E' - E$ are called Delaunay edges.*

Definition 5 (Local Delaunay property) *Two triangles $\Delta(v_0, v_1, v_2)$ and $\Delta(v_1, v_3, v_2)$ adjacent along the nonconstrained edge (v_1, v_2) are locally Delaunay, iff the circumcircle $C(v_0, v_1, v_2)$ does not contain the vertex v_3. This is aquivalent to the condition that the circumcircle $C(v_1, v_2, v_3)$ does not contain the vertex v_0. If all pairs of adjacent triangles of a triangulation T are locally Delaunay, the triangulation is said to be locally Delaunay.*

Theorem 6 *A constrained triangulation $T(G)$ of a PSLG $G = (V, E)$ is a Constrained Delaunay triangulation if it is locally Delaunay.*

The algorithm

Given a planar domain Ω with border polygons $B = \{b_0, \cdots, b_N\}$ and vertices $V = \{v_{00}, \cdots, v_{0n_0}, \cdots, v_{N0}, \cdots v_{Nn_N}\}$, where no four vertices lie on a circle, then there is exactly one CDT $\mathcal{T}(V, B)$ of the inner of Ω. Look at this triangulation of Ω: Exactly one triangle $\Delta(v_i v_j v_k)$ of this triangulation in the inner of Ω corresponds to each edge $v_i v_j$ on the border. Therefore, there is exactly one vertex v_k to each edge $v_i v_j$ on the border such that $\Delta(v_i v_j v_k) \in \mathcal{T}(V, B)$.
The idea of the algorithm is, to find this third vertex for every edge on the boundary of Ω. The search for the third vertex is based on the following characterization of triangles in the inner of Ω.

Lemma 7 *Let Ω be a polygonal domain with border polygons $B = \{b_0, \cdots, b_N\}$ and vertices $V = \{v_{00}, \cdots, v_{0n_0}, \cdots, v_{N0}, \cdots v_{Nn_N}\}$ on the border and let $v_i v_j$ be an edge of some of the border polygons. Assume that no four vertices lie on a common circle. In this situation a triangle $\Delta(v_i, v_j, v_k) \subset \Omega,\ v_i, v_j, v_k \in V$ belongs to the Constrained Delaunay triangulation $\mathcal{T}(V, B)$ if and only if*

- *(i)* $v_k \in V_{ij}$, *where* $V_{ij} := \{v \in V \mid v_i v \subset \Omega,\ v_j v \subset \Omega\}$,
- *(ii)* $\overset{\circ}{C}(v_i, v_j, v_k) \cap V_{ij} = \emptyset$, *i. e. in the inner of the circumcircle* $\overset{\circ}{C}(v_i, v_j, v_k)$ *there is no vertex* $v_l \in V_{ij}$.

Remark: Let H_{ij}^l be the left halfspace defined by the edge $v_i v_j$. Then $V_{ij} = W_i \cap W_j \cap H_{ij}^l$, where W_i, W_j are the set of vertices $v \in V$ that are visible from v_i, v_j respectively with respect to the border B.

Proof: Let $\Delta(v_i, v_j, v_k) \in \mathcal{T}(B, V)$. We show the properties (i) and (ii). Because of $\Delta(v_i, v_j, v_k) \subset \Omega$ property (i) follows directly.

According to definition 4 we have $\overset{\circ}{C}(v_i, v_j, v_k) \cap W_i \cap W_j \cap W_k = \emptyset$. We show property (ii) by contradiction.

Suppose that $\overset{\circ}{C}(v_i, v_j, v_k) \cap V_{ij} = \overset{\circ}{C}(v_i, v_j, v_k) \cap W_i \cap W_j \cap H_{ij}^l \neq \emptyset$. Then there are vertices that either belong to one of the subsets C_α or C_β of $\overset{\circ}{C}(v_i, v_j, v_k)$, see figure 2. Let us suppose that the vertices belong to C_β. The other case can be handled similarly. Let $w \in V_{ij} \cap \overset{\circ}{C}_\beta$ be the vertex with smallest distance to the edge $v_i v_k$, that is $d(w, v_i v_k) \leq d(v, v_i v_k)\ \forall v \in V_{ij} \cap \overset{\circ}{C}_\beta$. Then w is visible from v_k w. r. t. B and therefore $W_i \cap W_j \cap W_k \cap \overset{\circ}{C}$. This is a contradiction to the assumption, that $\Delta(v_i, v_j, v_k) \in \mathcal{T}(V, B)$.
The other direction of the proof can be obtained as follows. From $v \in V_{ij}$ it follows that $\Delta(v_i, v_j, v_k) \in \Omega$. It remains to show that $\overset{\circ}{C}(v_i, v_j, v_k) \cap W_i \cap$

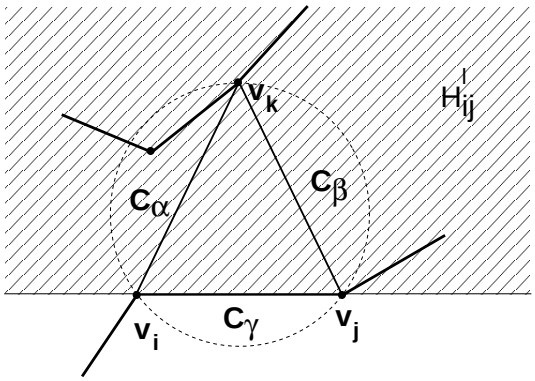

Figure 2: In triangle $\Delta(v_i, v_j, v_k)$ there are no vertices of V. Therefore, if there are vertices in the inner $\overset{\circ}{C}(v_i, v_j, v_k)$ of the circumcircle $C(v_i, v_j, v_k)$ on the left side of the oriented edge $v_i v_j$, these vertices belong either to C_α or to C_β.

$W_j \cap W_k = \emptyset$. Because of $\overset{\circ}{C}(v_i, v_j, v_k) \cap V_{ij} = \emptyset$ all vertices belonging to $\overset{\circ}{C}(v_i, v_j, v_k) \cap W_i \cap W_j \cap W_k$ must be contained in C_γ, see figure 2. But $C_\gamma \cap W_k = \emptyset$ and therefore, the assertion follows. ∎

Description of the algorithm

To describe the algorithm we suppose first, that the border of the domain consist of one single polygon P and discuss the general case later on.
The algorithm works recursively. While the polygon P contains more than three vertices, we choose an edge of P and search the corresponding third vertex with the properties (i) and (ii) of lemma 7.
According to the characterization of the lemma the triangle $\Delta(v_i, v_j, v_k)$ is a Constrained Delaunay triangle and subdivides the actual polygon P into two subpolygons P_1 and P_2. These subpolygons can be subdivided further independently.
In detail the following steps are performed:

1. Choose a starting edge $v_i v_j$.
2. Find the third vertex v_k with the following properties:
   (i) $v_i v_k \subset \Omega$ and $v_j v_k \subset \Omega$
   (ii) $\overset{\circ}{C}(v_i, v_j, v_k) \cap V_{ij} = \emptyset$.
3. Subdivide the actual polygon P into two subpolygons $P_1 = (v_i, v_k, v_{k+1}, \cdots, v_{i-1}, v_i)$ and $P_2 = (v_j, v_{j+1}, \cdots, v_k, v_j)$. (The vertices are indexed modulo the number of vertices of polygon P.)

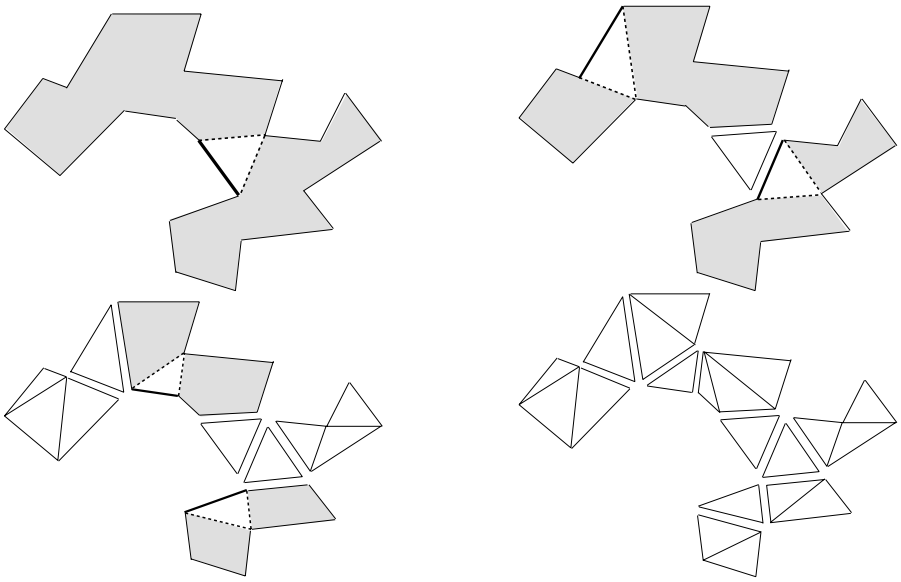

Figure 3: Functioning of the algorithm for a simple polygon.

4. Choose new starting edges in the subpolygons and repeat steps 2-4 until the actual polygon consists of only one triangle.

Remark 8 *The merging steps consist only in the unification of the triangulations of the subpolygons.*

The correctness of the algorithm for polygons follows from the following theorem.

Theorem 9 *Let P_1 and P_2 be the subpolygons of one recursion step of the algorithm and $\mathcal{T}_1, \mathcal{T}_2$ the corresponding CDTs of the inner of P_1 and P_2. Let $\Delta(v_i, v_j, v_k)$ be the triangle determined in this recursion step dividing the polygons P_1 and P_2. Then the Constrained Delaunay triangulation consists of the union $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Delta(v_i, v_j, v_k)$.*

Proof: For every inner edge of $\mathcal{T}_1$ and $\mathcal{T}_2$ the local Delaunay property is fulfilled. Therefore, we have to show the local Delaunay property only for the edges v_iv_k and v_jv_k. But for both of these edges the local Delaunay property follows directly from Lemma 7. ∎

Data structures

Given a polygon P and an edge v_iv_j for the search for the third vertex the following three steps are repeated until the correct vertex is found:

- Choose a vertex v_k of P on the left side of the edge v_iv_j.
- Determine the visibility between v_i, v_j and v_k.
- Verify that in $\overset{\circ}{C}\ (v_i, v_j, v_k)$ there are no further vertices on the left side of the edge v_iv_j which are visible from v_i, v_j, respectively, w. r. t. the border B.

Note that only for a valid third vertex all three steps are performed. These steps are accelerated by the use of a uniform grid data structure, see [PR93].

Maintainance of the grid-structure

All vertices and edges of the polygon are sorted into the corresponding grid cells. The vertices in the grid cells are administrated by double linked lists. Because of the divide and conquer approach we have to deal with several independent polygons during the algorithm. Therefore each grid cell contains a list of several linked lists of vertices, one for each subpolygon whose vertices belong to the grid cell itself, see for example figure 4. In addition each polygon holds a pointer to its linked list and each vertex of the polygon contains a pointer to its element in the corresponding linked list. Because of the number of vertices remains constant during the algorithm no more storage is neccessary to store the vertices in several lists instead of one list. The grid structure accelerates the search for the third vertex. Given an edge v_iv_j the grid cells intersecting with the bounding box of v_iv_j are determined. These cells defines a search area for the third vertex, which is increased stepwise until the third vertex is found, see [PR93]. A similar procedure using the uniform grid is performed to verify that for a choosen vertex $v_k \in V$ there are no vertices $v \in \overset{\circ}{C}\ (v_i, v_j, v_k) \cap V_{ij}$. The same is done for each edge of the polygon. Every grid cell contains a list of linked lists of edges, one for each subpolygon whose edges intersect the grid cell. To store the edges in their corresponding linked lists of the grid cells we use a modification of the algorithm of Amanatides and Woo [AW87]. The algorithm determines all grid cells intersected by a given edge. To determine the visibility between two vertices v_i and v_k the grid cells intersected by the edge v_iv_k are computed with the same algorithm and the intersections of the edge v_iv_k with all edges contained in the linked lists of these grid cells are computed.

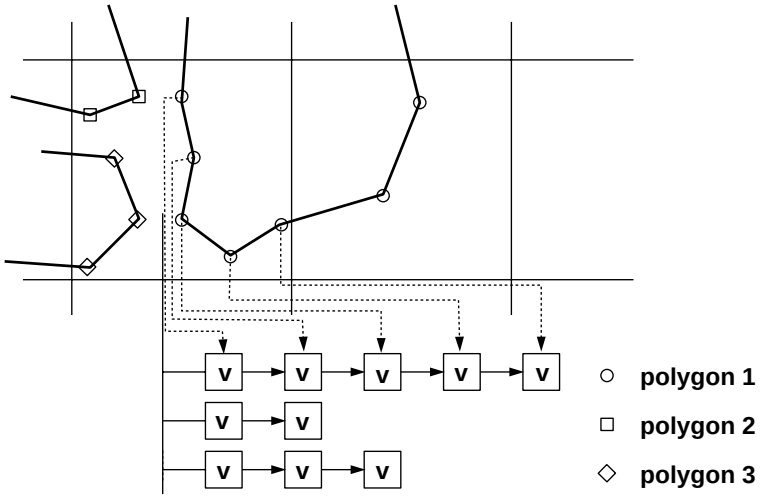

Figure 4: All vertices are sorted into grid cells. The administration of the vertices in the cells is done by linked lists. Providing that a grid cell contains vertices of a polygon, it has a vertex list in the cell.

The update of the linked lists is performed as follows. After a polygon $P = (v_0, \cdots, v_n)$ is divided by a triangle $\Delta(v_i, v_j, v_k)$ into two subpolygons $P_1 = (v_0, v_k, \cdots, v_n)$, $P_2 = (v_j, \cdots, v_k)$ the elements of the first polygon in the linked list corresponding to polygon P are copied into a new linked list and removed from the original one. The pointer of the second polygon is set to the original linked list, see for example figure 5.

Choosing a starting edge

The algorithm has optimal performance only if the polygon considered in the actual recursion step is divided in two subpolygons of about the same size. This can only be achieved by the correct choice of the starting edge, but in general this cannot be done in constant time. In the worst case a polygon with k vertices is divided into one triangle and a restpolygon with $k-1$ vertices. This leeds to a recursion depth of $O(n)$, where in the optimal case we obtain a recursion depth of $O(logn)$, i.e only $O(n)$ different starting edges are considered. A simple but efficient possibility to choose an starting edge is to consider the angles between consecutive edges of the polygons. If two consecutive angles in the polygon are both greater than 180^0 we can be sure that non of the neighbouring vertices of the corresponding edge is the third vertex to that edge. Choosing such an edge the worst case is avoided. In the algorithm for every edge v_iv_j a functional $f(v_iv_j)$ is evaluated and the edge with maximum functional value is choosen as

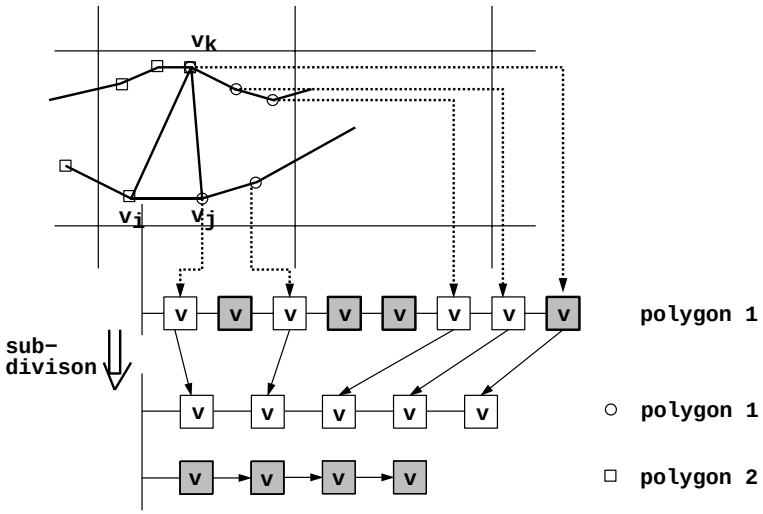

Figure 5: After the subdivision of a polygon the list in the grid cells are updated. If the polygon contains n vertices this can be done in $O(n)$ time using the pointers of the vertices to their list elements.

starting edge. We used different functionals which are motivated geometrically and constructed in such a way that the worst case is avoided:

$$\begin{aligned}
f_1(v_iv_{i+1}) &= \alpha_i + \alpha_{i+1} \\
f_2(v_iv_{i+1}) &= \max(\alpha_i, 180°) + \max(\alpha_{i+1}, 180°) \\
f_3(v_iv_{i+1}) &= (1-w)(\alpha_i + \alpha_{i+1}) + w(\alpha_{i-1} + \alpha_{i+2}) \\
f_4(v_iv_{i+1}) &= \max(\max(\alpha_i, 180°), \max(\alpha_{i+1}, 180°)) \\
f_5(v_iv_{i+1}) &= (1-w)(\max(\alpha_i, 180°) + \max(\alpha_{i+1}, 180°)) \\
&\quad + w(\max(\alpha_{i-1}, 180°) + \max(\alpha_{i+2}, 180°),
\end{aligned}$$

where $0 \leq w \leq 1$ is a weight factor. The indices are calculated modulo the size of the polygon.
At the beginning of the algorithm all angles between consecutive edges are computed. In each subdivision step we get four new angles and the functional must be updated accordingly. This can be done in constant time.
The different functionals were tested for various randomly generated polygons. An example of such a polygon is shown in figure 9. The results for different polygon sizes are shown in figure 6. In the functionals f_3 and f_5 we choose a weight factor $w = 2/3$. The functionals f_1 and f_2 delivered the best results for all example polygons. As in the optimal case the sum of the sizes of the subpolygons depends linearly of the size of the starting polygon. The functionals f_3 and f_5 shows that the consideration of more than two adjacent angles of an edge does not result in better subdivisions

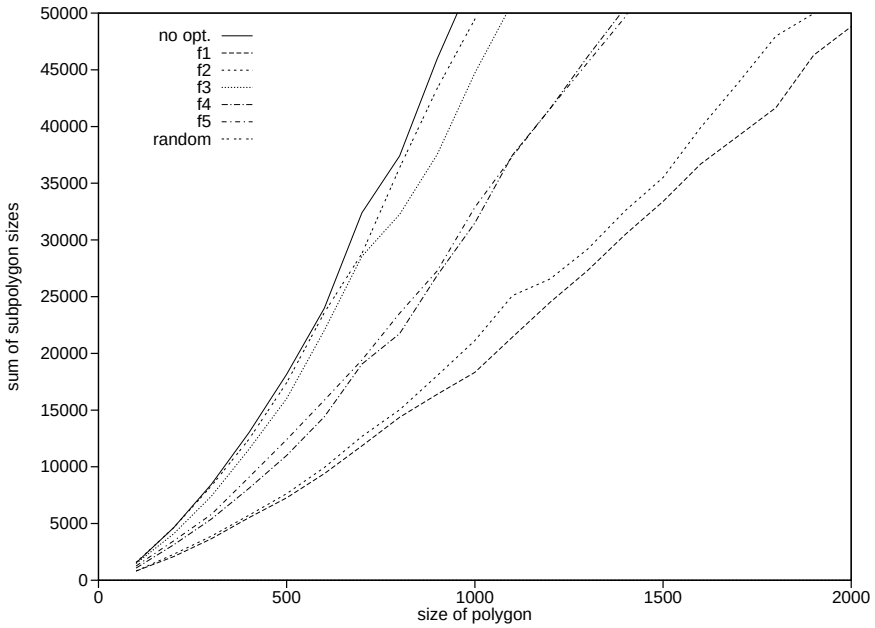

Figure 6: The sum of the sizes of all subpolygons generated during the algorithm dependent on the size of the starting polygon is shown without optimization, for different functionals $f_1, \cdots, f_5$ and for a random choice of the starting edge. If we choose the functionals f_1 and f_2 we obtain a much better dependence between the size of the starting polygon and this sum than $O(n^2)$, i.e. the polygons are subdivided in a nearly optimal way. In the optimal case we would get a dependence of $O(n \log n)$.

of the polygons. Both, choosing the edges randomly or without optimization leeds to nearly the same results and are not optimal.

Domains with more than one border polygon.

If the border B of the domain Ω consists of more than one polygon, inner polygons are processed first. In most cases the corresponding third vertex to an edge belongs to another polygon. To find the third vertex, we have to consider the edges of all other polygons. If the starting egde and the third vertex belong to different border polygons these polygons are joined, see figure 7.

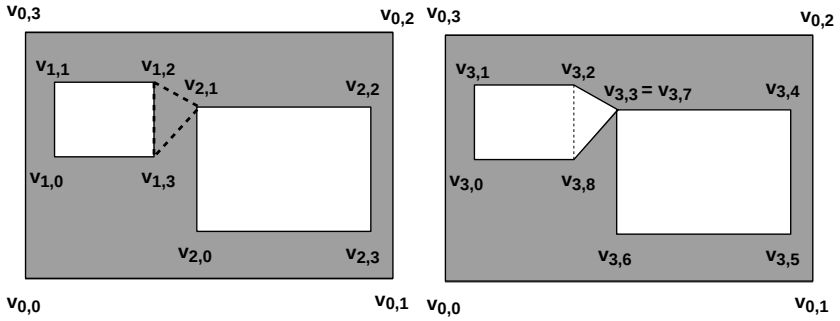

Figure 7: The corresponding vertex $v_{2,1}$ to edge $v_{1,2}v_{1,3}$ of polygon b_1 belong to an other polygon b_2. The polygons b_1 and b_2 are joined and we get polygon $v_{3,0}, \ldots, v_{3,8}$.

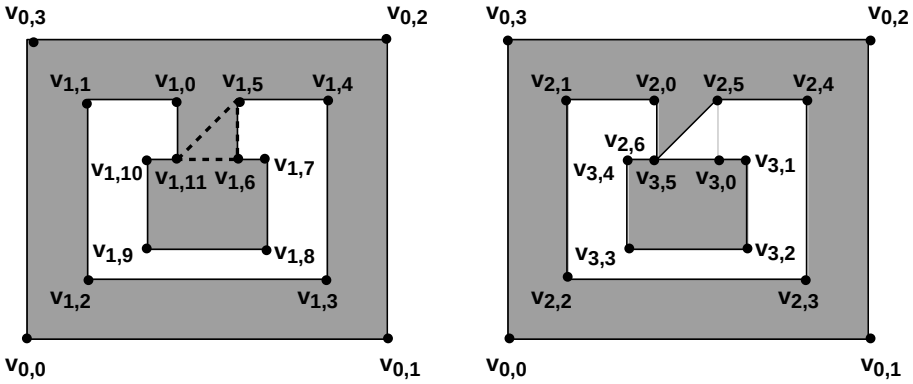

Figure 8: The triangle $\Delta(v_{1,5}, v_{1,11}, v_{1,6})$ subdivides the inner polygon $b_1\,(v_{1,0}, \ldots, v_{1,11})$ into two border polygons $b_2 = (v_{2,0} \ldots v_{2,6})$ and $b_3 = v_{3,0} \ldots v_{3,5}$. Remark that $v_{2,6} = v_{3,5}$. The Polygon b_3 lies outside the polygon b_2 and therefore b_3 can be triangulated without taking care of the other border polygons.

If the corresponding third vertex belongs to the same inner polygon, one of the two subpolygons belongs to the outer area of the other subpolygon, see figure 8. To triangulate this subpolygon the edges of the other border polygons must not be considered. If no inner border polygons are left, we are in the situation of one border polygon and can proceed as described above.

Results

The subdivision of the polygon together with the grid data structure leeds to a nearly linear dependence between time and size of the polygon in practical examples. This is shown in table 1. To get these results we use func-

tional f_1 to optimize the selection of a starting edge and take the average of times to compute the Constrained Delaunay triangulation of 100 randomly generated polygons of the same size. Examples of test polygons are shown in figure 9.

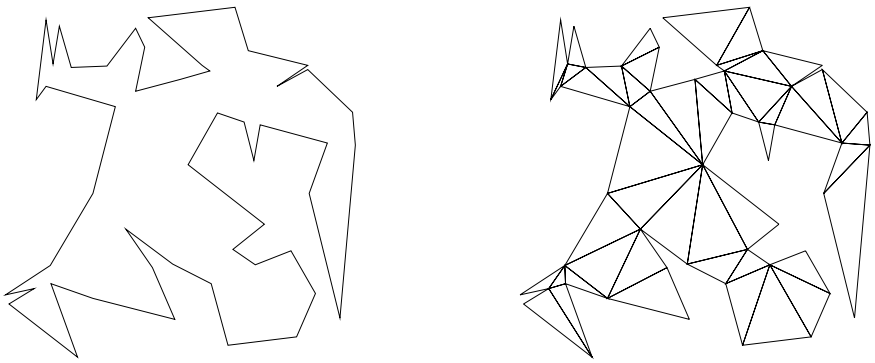

Figure 9: The run times of the algorithm was measured for randomly generated polygons. To get these polygons, we remove arbitrary triangles from a triangulation of randomly generated vertices. On the right side a Constrained Delaunay triangulation of the randomly generated polygon of the left side is shown.

size	200	400	600	800	1000	1200	1400	1600	1800	2000
time	20.6	41.2	61.9	82.5	103.2	124.0	144.6	165.4	186.1	206.9

Table 1: The run time of the algorithm for randomly generated polygons of different size. For every size the run time is averaged over 100 random generated test polygons.

Conclusions

A fast and easy to implement divide-and-conquer algorithm for the calculation of a Constrained Delaunay triangulation of planar domains was described. The algorithm combines the divide and conquer paradigma with a uniform grid as acceleration structure. In contrast to existing algorithms calculations of triangles not contained in the inner of the domain are avoided. For randomly generated polygons the run time of the algorithm depends nearly linear from the size of the border polygons of the domain.

Acknowledgement

I would like to thank J. Krämer for his efforts on all of the major implementation issues and the fruitful discussions that led to the realization of

this algorithm.

References

[AGSS89] A. Aggarwal, L. J. Guibas, J. Saxe, and P. Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete Computational Geometry*, 4:591, 1989.

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In G. Marechal, editor, *Eurographics '87*, pages 3–10. North-Holland, August 1987.

[Boi88] J. D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics and Image Processing*, 44, 1988.

[Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[Che89] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[CTV89] K. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[DFP85] L. DeFloriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics and Image Processing*, 32:127–140, 1985.

[FS91] D.A. Field and W.D. Smith. Graded tetrahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 31(3):413–425, 1991.

[HL88] K. Ho-Le. Finite element mesh generation methods: a review and classification. *Computer Aided Design*, 20:27–38, 1988.

[Ke95] R. Klein and W. Straßer. Mesh generation from boundary models. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 431–440. ACM Press, May 1995.

[KKT90] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log\log n)$ time with simple data structures. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 34–43, 1990.

[Kle94] Reinhard Klein. Linear approximation of trimmed surfaces. In R.R. Martin, editor, *The Mathematics Of Surfaces VI*, 1994.

[LL86] D. T. Lee and A. K. Lin. Generalized Delaunay triangulations for planar graphs. *Discrete Comput. Geom.*, 1:201–217, 1986.

[PR93] L. A. Piegl and A. M. Richard. Algorithm and data structure for triangulating multiply connected polygonal domains. *Computer & Graphics*, 17(5):563–574, 1993.

[Rip92] S. Rippa. Adaptive approximation by piecewise linear polynomials on triangulations of subsets of scatterd data. *SIAM Journal Sci. Stat. Comput.*, 13(5):1123–1141, September 1992.

[Wat81] D. F. Watson. Computing the n-dimensional delaunay tesselation with application to voronoi polytopes. *Comput. J.*, 24:167–172, 1981.