Straight Skeleton Implementation

Petr Felkel and Štěpán Obdržálek
*felkel|xobdrzal@fel.cvut.cz**

Abstract

Straight skeleton (Angular Bisector Network, ABN) of a planar polygon, which can be grasped as a modification of a planar Voronoi diagram without parabolic arcs, has been successfully used by Oliva et al. [5] as a part of a system for three dimensional reconstruction of objects from a given set of 2D contours in parallel cross sections. The algorithm itself is used for the construction of intermediate contour layers during the reconstruction process, in order not to create self intersected surface triangles or a surface with holes.

But – Oliva's algorithm is not publicly available and we have not found any other useful code on the net. We have followed our older ideas [4] and implemented our version of straight skeleton. Our algorithm runs in $O(nm + n \log n)$ time, where n denotes the total number of polygon vertices and m the number of reflex ones.

1 Introduction

Skeleton like structure is often used for the description of basic topological characteristics of a 2D object. In the image processing and the computer vision fields, *skeleton*, informally defined as a set of discrete points in raster environment located in the centers of circles included in the object that touch the object boundary in at least two distinct points [7], is used. Very similar goal can be achieved in the continuous vector environment by a *medial axis* of a polygon (a subset of *generalized Voronoi diagram*), which consists of line segments and parabolic arcs.

Aichholzer [1] resp. Oliva et al. [5] introduced a new internal structure for simple planar polygon called *straight skeleton* resp. *Angular Bisector Network, ABN*, which is made up of straight line segments, which are pieces of angular bisectors of polygon edges.

There is a lot of literature about construction of the Voronoi Diagram, eg. [6, 2] and publicly available algorithms for its computation, but, as we know, the idea of straight skeleton is mentioned very rarely and there is no publicly available implementation of the algorithm.

In this article we describe our implementation of the straight skeleton algorithm. We exploit it in segmentation of the capillary bed of the human placenta for a 3D reconstruction from contours in parallel cross sections. The idea of reconstruction itself was described by Oliva et al. [5]. We apply the reconstructed surface constructed from easy segmentable slices for prediction of the contour shape in the slices in between, where the boundaries of object of interest are not sharp [3].

2 Straight Skeleton Computation

We can imagine the principle of the algorithm as a construction of a roof with constant slope from a given shape of the walls [1, 4]. It can be done by a sweep algorithm, which simulates cutting of the roof by parallel planes and checks local changes of the polygonal base in the cross sections. In a 2D view, it appears as shrinking of the polygon. The polygon edges are moving in constant speed inward the polygon and they are changing their lengths. The polygon vertices move along the angular bisectors as long as the polygon changes its topology. Aichholzer [1] described two possible types of changes:

*CTU Prague, Department of Computer Science & Engineering

- *Edge event*: An edge shrinks to zero, making its neighboring edges adjacent.
- *Split event*: A reflex vertex runs to the edge and splits it, thus splits the whole polygon. New adjacencies occur between the split edge and each of the two edges incident to the reflex vertex.

The straight skeleton $S(P)$ of the polygon P is defined as the union of the pieces of angular bisectors traced out by polygon *vertices* during the shrinking process. Each edge (a straight line segment) e sweeps out certain area which we call *face* of e. Bisector pieces are called *arcs*, and their endpoints which are not vertices of P are called *nodes* of $S(P)$. See an example in Fig. 1, where the straight skeleton arcs are drawn by a thick lines and the shape of intermediate levels by a thin line, and for more details refer to [1]. Our algorithm handles vertices and nodes the same way.

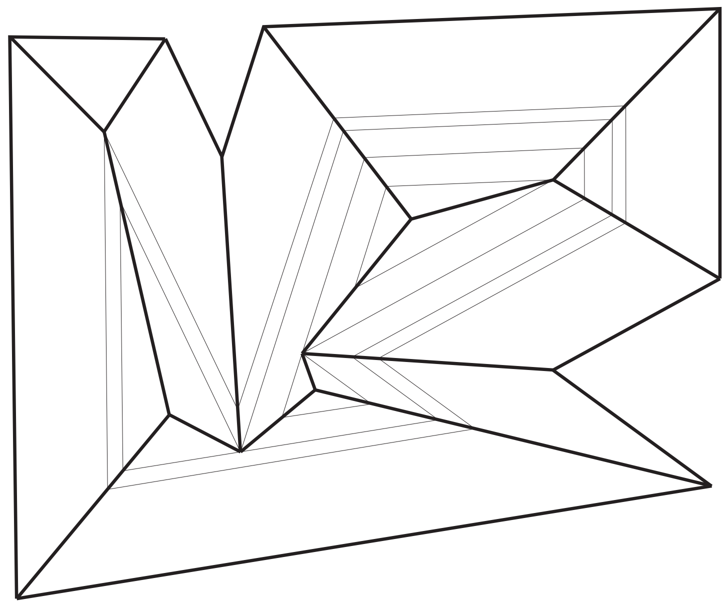

Figure 1: Polygon hierarchy (thin line) and straight skeleton – by [1]

We will describe the method for straight skeleton computation in two steps. At first for a convex polygon, then for a non-convex one. Our algorithm applies the principle of the roof construction by sweeping, but instead of constructing the polygonal base in the cross sections, it manages only the pointers to the edges of the original polygon.

The basic data structure used by the algorithm is a *Set of circular Lists of Active Vertices (SLAV)*. This structure stores a loop of vertices for outer boundary and for all holes and sub-polygons created during the straight skeleton computation. In the case of convex polygon, it always contains only one list (LAV). In the case of a simple non-convex polygon, it stores a list for every sub-polygon (as described later) and in the case of polygons with holes also a list for each hole.

All the vertices in the SLAV have references to both neighbors (vertices of the polygon) in the circular lists (LAVs) stored in the SLAV.

2.1 Convex Polygon Skeleton Computation

Given a simple convex polygon P, only the *edge events* occur and the straight skeleton $S(P)$ is computed in the following steps (We suppose the polygon vertices and edges are oriented counter-clockwise and the polygon interior is on the left-hand side of its boundary):

1. Initialization:

(a) Organize given vertices $V_1, V_2 \dots, V_n$ into one *double connected circular List of Active Vertices* (LAV) stored in the SLAV. The vertices are all active at this moment,

(b) for each vertex V_i in the LAV add the pointers to two incident edges $e_{i-1} = V_{i-1}V_i$ and $e_i = V_iV_{i+1}$, and compute the vertex angle bisector (ray) b_i,

(c) for each vertex V_i compute the nearer intersection of the bisector b_i with adjacent vertex bisectors b_{i-1} and b_{i+1} starting at the neighboring vertices V_{i-1}, V_{i+1} and (if it exists) store it into a priority queue according to the distance to the line $L(e_i)$ supporting the edge e_i. For each such intersection point I_i store also two pointers to the vertices V_a, V_b, that means two origins of bisectors which have created the intersection point I. They are necessary for the identification of appropriate edges (e_a and e_b in Fig. 2) during the bisector computation in later steps of the algorithm.

2. While the priority queue with the intersection points is not empty do:

(a) Pop the intersection point I from the front of the priority queue,

(b) if the vertices/nodes V_a and V_b, pointed to by I, are marked as processed
then continue on the step 2,
else the edge e between the vertices/nodes V_a, V_b shrinks to zero (*edge event* - this edge is in Fig. 2 marked by a cross),

(c) if the predecessor of the predecessor of V_a is equal to V_b (peak of the roof)
then output three straight skeleton arcs V_aI, V_bI and V_cI, where V_c is the predecessor of V_a and the successor of V_b in the LAV simultaneously, and continue on the step 2,

(d) output two skeleton arcs of the straight skeleton V_aI and V_bI,

(e) modify the list of active vertices/nodes (see Fig. 2 for details):

- Mark the vertices/nodes V_a, V_b (pointed to by I) as processed (marked by a cross in Fig. 2),
- create a new node V with the coordinates of the intersection I (a square mark in Fig. 2),
- insert this new node V into the LAV. That means connect it with the predecessor of V_a and the successor of V_b in the LAV (thick arrows in Fig. 2),
- link the new node V with appropriate edges e_a and e_b (pointed to by the vertices V_a and V_b),

(f) for the new node V, created from I:

- compute a new angle bisector b between the line segments e_a and e_b,
- compute the intersections of this bisector with the bisectors starting from the neighboring vertices in the LAV in the same way as in the step 1c,
- store the nearer intersection (if it exists) to the priority queue.

As you can see in the steps 1c and 2f, there are duplicities among the intersection points in the priority queue. The algorithm always computes one intersection for one vertex. Step 2b removes these duplicities.

2.2 Non-convex Polygon Skeleton Computation

The principle of the method for the straight skeleton computation is in the case of non-convex polygons similar. New intersection points (one for each vertex/node in all LAVs in the SLAV) are after their computation stored into the priority queue as in the section 2.1, but they have a new attribute indicating their event type: *edge event* or *split event*.

At first, we discuss the circumstances which we have to take into account.

Presence of a reflex vertex may (but may not) lead into a polygon splitting (see Fig. 3). In case of the point A a standard *edge event* occurs which is handled the same way as the edge event of non-reflex

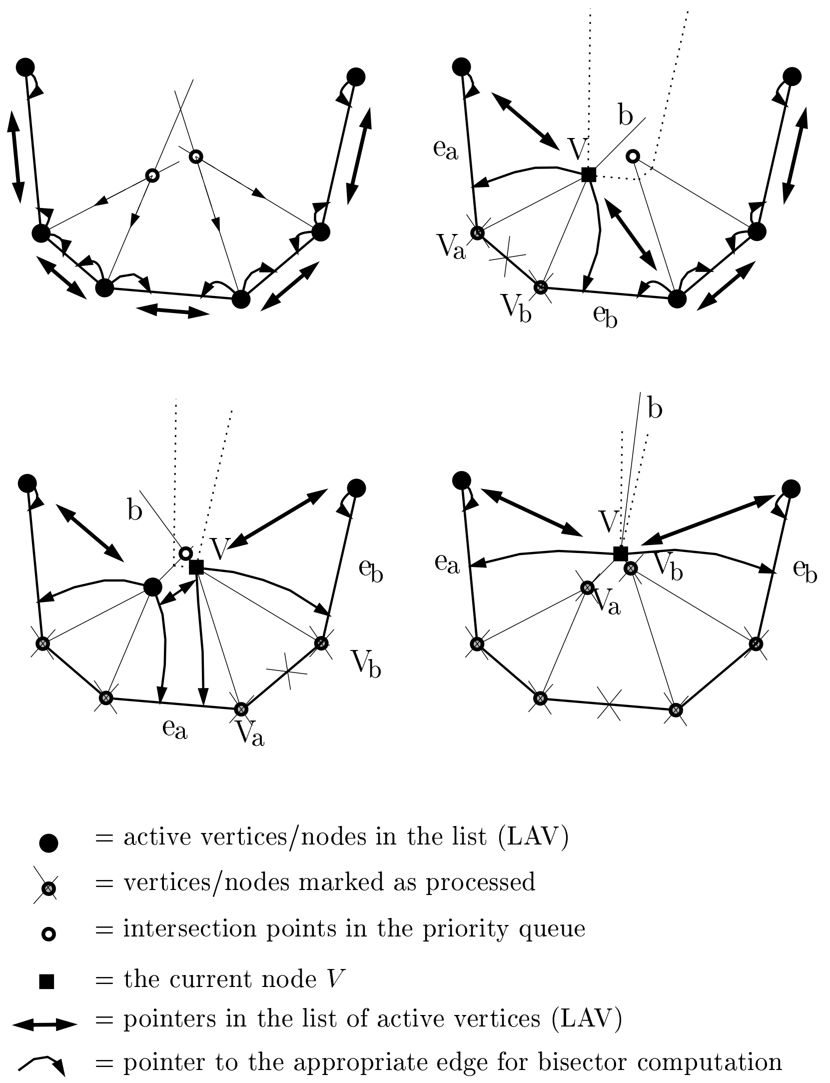

Figure 2: Initialization and the first three steps of the skeleton algorithm for convex vertices

vertex (as described in the section 2.1). We will concentrate on the case of the point B, where a *split event* occurs.

The first task is to determine the coordinates of the point B. Then we will discuss the insertion of B into the appropriate LAV in the SLAV and a special case of a multiple split edge. The name B represents the coordinates of the tested intersection point and also the coordinates of the new vertex inserted into LAV. We will distinguish them in the algorithm.

Determination of the coordinates of the point B

Point B can be characterized as having the same perpendicular distance to the straight line carrying the "opposite" line segment to the vertex V and from both straight lines containing the line segments starting at the vertex V. We have to find such an "opposite" line segment.

We traverse all line segments e of the original polygon and test them whether they can be the opposite line segments. Unfortunately a simple test of the intersection between a bisector starting at V

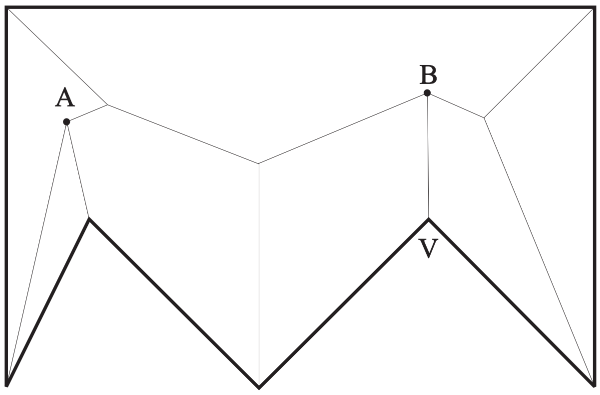

Figure 3: Reflex vertex can yield to both cases: an edge event (point A) or a split event (point B)

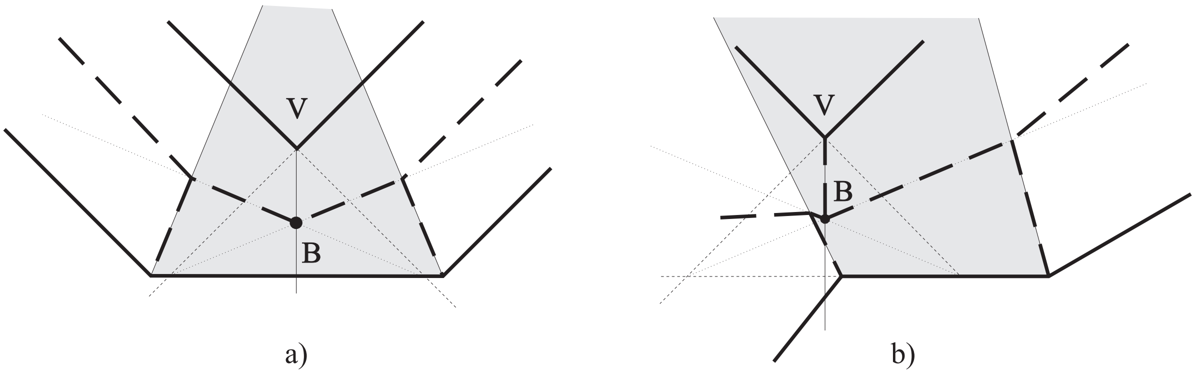

Figure 4: Intersection point computation for a reflex vertex V which yields a split event (point B)

and the currently tested line segment e_i cannot be used (see Fig. 4b). We have to test the intersection with the line supporting the edge e_i and to test whether the candidate point B_i lays in the area limited by the currently tested edge e_i and by the bisectors b_i and b_{i+1} leading from the vertices at both ends of this line segment (see Fig. 4a,b).

Simple intersection test between the bisector starting at V and the (whole) line containing the currently tested line segment e_i rejects the line segments laying "behind" the vertex V. We then compute the coordinates of the candidate point B_i as the intersection between the bisector at V and the axis of the angle between one of the edges starting at V and the line supporting the tested line segment e_i (see Fig. 4). Simple check should be performed to properly handle the case when one of the line segments starting at V is parallel to e_i.

The resulting point B is selected from all the candidates B_i as the nearest point to the vertex V.

Managing of the LAV in the case of a split event

When the intersection B (called I on Fig. 6) of the split event type is processed, it is necessary to split the appropriate polygon into two parts. Splitting of the polygon also implies splitting of the appropriate LAV into two parts and implies insertion of two new nodes V_1 and V_2 with the coordinates taken from I into them – each copy into one LAV (see details in Fig. 6 and in the second algorithm).

Both copies V_1 and V_2 of I point to the same split edge e_i and share it.

A special case of multiple splitting of the edge

As mentioned before, the algorithm works with the original edges and does not construct the intermediate roof shapes in the cross-sections. This yields the situation when one original line segment is shared by more than one intermediate polygons after the previous polygon split. In such a case when one line segment is already split and the next edge event occurs we have to choose the opposite line segment end-points correctly (resp. the vertices/nodes which are active in the current roof construction level as the points X and Y in Fig. 5).

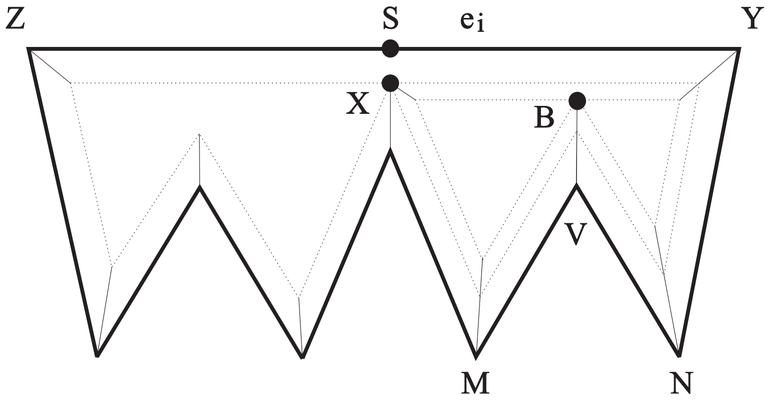

Figure 5: Correct selection of the active vertex/node X for already split edge

In the case of the sub-edge SY (a part of the edge $e_i = ZY$), the real end point of the original line segment e_i is the point Z, but we search for the point X, which is necessary for the correct interconnection of pointers when the new node with the coordinates of B is inserted into the SLAV. We mention two methods how to determine X.

One possible solution is to split the border line segment physically and to create a new auxiliary vertex S. It removes the cause of the problem, because no edge will be divided more than once and the split edge end-points determination is trivial. It also implies that the insertion of the new node B into the SLAV is also trivial, as we know exactly in which of the sub-polygons (LAV) it has to be interconnected. It solves the problem with the shared edge e_i but such a vertex S has to be handled in a special way, because it is not included in the skeleton.

Our solution is based on the idea to store only the nodes which are present in the straight skeleton and on the fact, that the references to the split line segments are stored in all of the sub-polygons which share these line segments. It reveals during the traversal of all sub-polygon LAVs in multiple hits of the split line segment, each time for each sub-polygon sharing it.

For instance the two subpolygons in Fig. 5 share a reference to the edge e_i. During the processing of the intersection B associated with the vertex V the polygon $XMVNY$ is split into two parts XMB and BNY and the vertex V is marked as processed.

All it is done for the reason to correctly interconnect B between Y and X and not between the original endpoints Y and Z. During the SLAV traversal the correct part of the split edge e_i is selected (determined by Y and X) by means of the same criterion as described before, i.e. the candidate point B lays rightwards of the traversed vertex bisector Y and leftwards of the bisector of the traversed vertex's successor X in the LAV.

The algorithm for simple non-convex polygons

Algorithm for non-convex polygons is in the principle similar to the algorithm described in the section 2.1. As an extension, it handles the intersections that may generate polygon splits (split events):

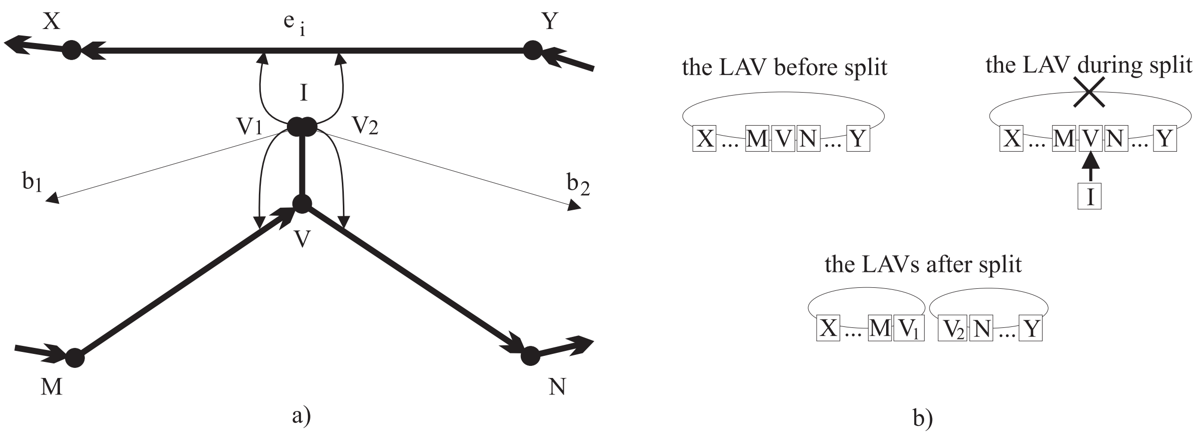

Figure 6: Split event at the reflex vertex: a) managing the edge pointers, b) LAV split caused by the new nodes V_1 and V_2

1. Initialization
   (a) Generate one LAV as in the convex case and store it in the SLAV,
   (b) compute the vertex bisectors as in the convex case,
   (c) compute intersections with the bisectors from the previous and the following vertices as in the convex case and for reflex vertices compute also the intersections appropriate to the opposite edges (point B in figs. 3 and 4) as described before. Store the nearest intersection point I of these three ones into the priority queue. In addition store also the type of the intersection point (edge event or split event).
2. While the priority queue is not empty do:
   (a) Pop the lowest intersection point I from the queue as in the convex case. If the type of I is the *edge event*,
   then process steps 2b to 2f of the algorithm for convex polygons,
   else (*split event*) continue within this algorithm,
   (b) if the intersection point points to already processed vertices continue on step 2 as in the convex case,
   (c) do the same as in the convex case, only the meaning is a bit different, because more local peaks of the roof exist,
   (d) output one arc VI of the straight skeleton, where vertex/node V is the one pointed to by the intersection point I. Intersections of the *split event* type always point exactly to one vertex/node V in the SLAV (compare that to the convex version of the algorithm, where the *edge event* type intersections point to two vertices/nodes V_a and V_b),
   (e) modify the set of lists of active vertices/nodes (SLAV):
      - Mark the vertex/node V (pointed to by I) as processed,
      - create two new nodes V_1 and V_2 with the same coordinates as the intersection point I,
      - search for the opposite edge in the SLAV (sequentially),
      - insert both new nodes into the SLAV (break one LAV into two parts — see Fig. 6b for an example). Vertex V_1 will be interconnected between the predecessor of V (M in Fig. 6) and the vertex/node which is an end point of the opposite line segment (X on Fig. 6). V_2 will be interconnected between the successor of V (N on Fig. 6) and the vertex/node

which is a starting point of the opposite line segment (Y on Fig. 6). This step actually splits the polygon into two parts (as discussed before in this section).
- link the new nodes V_1 and V_2 with the appropriate edges (see Fig. 6a).

(f) for both nodes V_1 and V_2:
- Compute new angle bisectors between the line segments linked to them in step 2e,
- compute the intersections of these bisectors with bisectors starting at their neighboring vertices according to the LAVs (e.g. at points N and Y for node V_2 and M and X for node V_1 in Fig. 6a), the same way as in step 1c. New intersection points of both types may occur,
- store the nearest intersection into the priority queue.

2.3 Straight Skeleton Computation for Polygon with Holes

Our algorithm can also handle the polygons with holes if they have the appropriate orientation. That means the polygon interior lays leftwards from all of the line segments, the vertices of outer boundary are in counter-clockwise order and the vertices of holes are in clockwise order. It results in more cycles of vertices (LAVs) in the SLAV in the step 1a of the algorithm.

An example of the straight skeleton of a polygon with one hole is in Fig. 7.

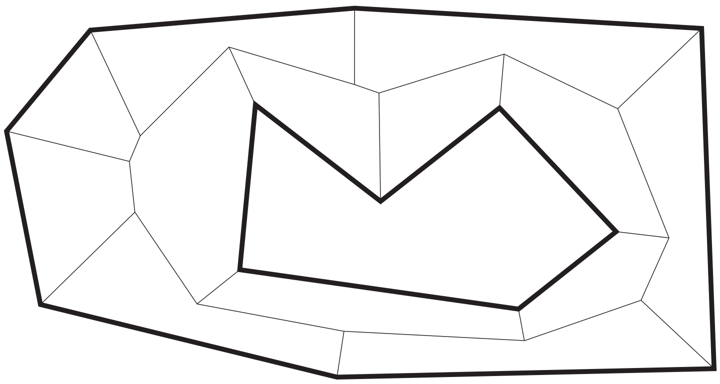

Figure 7: An example of the straight skeleton of a polygon with a hole

3 The Results and Conclusion

We have implemented an algorithm for construction of a straight skeleton of non-convex polygons with holes, which runs in $O(nm + n \log n)$ time, where n denotes the total number of polygon vertices and m the number of reflex ones. The algorithm handles convex, non-convex polygons and polygons with holes correctly.

Handling of the split events takes time $O(n \log n)$ with $O(n)$ storage in the priority queue, but it is not the significant part for the whole complexity. The most time consuming task is the computation of the split events — that means handling m ($m < n$) reflex vertices. That yields the global time complexity of the presented algorithm $O(n^2)$ ($O(nm)$) with $O(n)$ storage. That is a similar result as in [5].

We have performed the measurements of the time complexity of the algorithm for polygons with increasing number of vertices. The execution times for different non-convex polygons (as shown in Fig. 8) confirm the theoretical presumptions of its quadratic time complexity.

The C++ code for straight skeleton is available upon request.

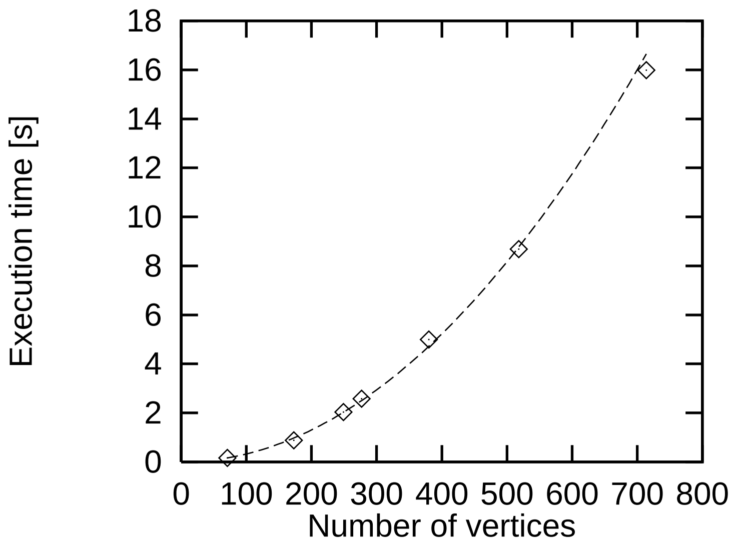

Figure 8: Results of the tests for polygons with different number of vertices

References

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science, http://www.iicm.edu/jucs_1_12, Institute for Image Processing and Computer Supported New Media*, 1(12):752–761, 1995.

[2] M. de Berg, M. vam Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications.* Springer Verlag Berlin Heidelberg New York, 1997.

[3] P. Felkel. 3D Reconstruction from Cross Sections by means of Contour Tiling. In A. Strejc, editor, *Workshop '98*, volume I, pages 241–242. Czech Technical University Publishing House, Prague, Czech Republic, Feb. 3.–5. 1998.

[4] P. Felkel and J. Žára. The Roof Construction Problem. In A. Strejc, editor, *Proceedings of CTU Workshop '93*, volume I - Informatics & Cybernetics, pages 85–86. CTU-Publishing House, Prague, Jan. 18–21, 1993.

[5] J.-M. Oliva, M. Perrin, and S. Coquillart. 3D Reconstruction of Complex Polyhedral Shapes from Contours using a Simplified Generalized Voronoï Diagram. *Computer Graphics Forum*, 15(3):C-397–C-408, 1996.

[6] F. P. Preparata and M. I. Shamos. *Computational Geometry – An Introduction.* Springer-Verlag, New york, 1985.

[7] M. Šonka, V. Hlaváč, and R. Boyle. *Image Processing, Analysis and Machine Vision.* Chapman and Hall, London, New York, 1993.

Acknowledgments

This research has been conducted at the Department of Computer Science & Engineering and has been supported by CTU grant No. 3097476.