Computing the Straight Skeleton of a Monotone Polygon in $O(n\log n)$ Time

Gautam K. Das* Asish Mukhopadhyay† Subhas C. Nandy‡ Sangameswar Patil§ S. V. Rao¶

Abstract

The straight skeleton of a simple polygon is defined as the trace of the vertices when the initial polygon is shrunken in self-parallel manner [2]. In this paper, we propose a simple algorithm for drawing the straight skeleton of a monotone polygon. The time and space complexities of our algorithm are $O(n\log n)$ and $O(n)$ respectively.

1 Introduction

Skeleton like structure is often used for the description of basic topological characteristics of a 2D object, and have numerous applications in image processing, computer vision, solid modeling, mesh generation, to name a few [3, 11]. The medial axis or the skeleton of a simple polygon P is the locus of the centers of all the circles inside P that touch the boundary of P in two or more points. The medial axis of an arbitrary simple polygon consists of line segments and parabolic arcs, where each parabolic arc corresponds to a reflex vertex. It can also be viewed as the Voronoi diagram whose sites are the edges and vertices of the polygon boundary. The medial axis is basically the one dimensional representation of a simple polygonal object. In spite of many uses of medial axis, its curved arcs have been considered to be a shortcoming in different applications.

Straight skeleton for a simple polygon has been introduced in [2]. It is solely made of straight line segments which are pieces of angular bisectors of polygonal edges. Unlike medial axis, the straight skeleton of an arbitrary simple polygon is not the Voronoi diagram of its edges.

The definition of straight skeleton can be generalized to (i) arbitrary planar straight line graphs [1], which has applications in planar motion planning, and (ii) in 3D polyhedra, which has applications to solid modeling. Some nice applications of the straight skeleton are also demonstrated in computational origami [6, 7]

Although the medial axis of a simple polygon can be constructed in linear time [4], the fastest known algorithm for the straight skeleton runs much slower [9]. Table 1, lists the time and space bounds of various algorithms. Here n is the total number of vertices and r is the number of reflex (non-convex) vertices of the polygon; ϵ is an arbitrarily small positive constant.

Cheng and Vigneron [5] established connection between straight skeleton and motorcycle graph, and presented an $O(n\sqrt{n}\log n)$ time algorithm for computing motorcycle graph. They also proposed a randomized algorithm for computing the straight skeleton of a simple polygon, which runs in $O(n\log^2 n)$ time after computing motorcycle graph induced by reflex vertices of the polygon. Combining these two algorithms they proposed a randomized algorithm for computing the straight skeleton of a simple polygon with r reflex vertices which runs in $O(n\log^2 n + r^{3/2}\log r)$ expected time.

2 The straight skeleton and its properties

The straight skeleton of a simple polygon P is denoted by $S(P)$. The $S(P)$ is defined using an appropriate shrinking process for P. The boundary of P is contracted towards its interior in a self-parallel manner, and at the same speed for all edges. Each vertex of P moves along the angular bisector of its incident edges. This situation continues as long as the boundary does not change topologically. Then a new node of $S(P)$ is created. The shrinking process continues in the resulting polygon(s). The process terminates when the area of all the resulting polygons become zero. For the demonstration of the shrinking process, the reader may refer to [2]. It gives a hierarchy of nested polygons. Each polygon in the hierarchy is obtained after any one of the following three events.

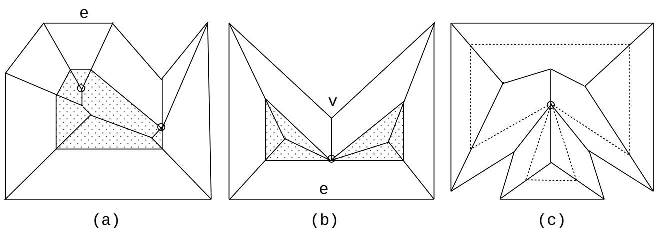

Figure 1: Illustration of (a) Edge-event, (b) split-event, and (c) vertex-event

edge-event: An edge e of P vanishes due to shrinking. This event is denoted by $EE(e)$. After the occurrence of $EE(e)$, the edges incident to the two end-points

*University of New Brunswick, Fredericton, Canada
†University of Windsor, Windsor, Canada
‡Indian Statistical Institute, Kolkata, India
§Tata Consultancy Services, Pune, India
¶Indian Institute of Technology, Guwahati, India

Table 1: Time and space complexity of earlier algorithms.

Time	Space	Reference
$O(n^2\text{log}n)$	$O(n)$	[1]
$O(nr\text{log}n)$	$O(nr)$	[2, 9]
$O(n\text{log}n + nr\text{log}(n/r))$	$O(nr)$	[1, 8, 9]
$O(n\text{log}n + nr + r^2\text{log}r)$	$O(n)$	[1, 9]
$O(n\text{log}n + nr)$	$O(n + r^2)$	[1, 8, 9]
$O(n\text{log}n + nr)$	$O(n)$	[10]
$O(n^{1+\epsilon} + n^{8/11+\epsilon}r^{9/11+\epsilon})$	$O(n^{1+\epsilon} + n^{8/11+\epsilon}r^{9/11+\epsilon})$	[9]
$O(nlog^2n + r^{3/2}logr)^*$	$O(n)$	[5]

* indicates randomize algorithm

of e become adjacent (see Fig. 1(a)); thus the number of edges in the resulting polygon is reduced by one.

split-event: An edge is split by a reflex vertex. In other words, a reflex vertex touches this edge during the shrinking process. This event is denoted by $SE(v, e)$, where v and e are the participating reflex vertex and edge of P respectively. After the occurrence of $SE(v, e)$, the polygon P is split into two sub-polygons of non-zero area (see Fig. 1(b)).

We will use the terms arcs and nodes for denoting the objects that form the boundary of the skeleton, in order to distinguish them from the objects forming the polygon, which will be called edges and vertices. Bisector pieces are called *arcs*, and their endpoints which are not vertices of P are called *nodes* of $S(P)$. Each edge/split-event introduces a node of degree three in $S(P)$. The angle(s) incident at the new vertex in the resulting polygon(s) is/are always convex.

In the degenerate cases, $S(P)$ may have nodes of degree greater than three, introduced by the occurrence of simultaneous events at the same location. In most cases, we can handle these events one at a time using standard perturbation techniques. This replaces a high-degree node with several nodes of degree three, connected by zero-length arcs.

vertex-event: It occurs when two or more vertices (but no edge) reach the same point simultaneously. Unlike edge/split-event, a vertex-event may introduce a new reflex vertex into the shrunken polygon, although the total number of reflex vertices always decreases (see Fig. 1(c)).

Proposition 1 *([1]) The straight skeleton $S(P)$ of P satisfies: (a) $S(P)$ is a tree with n leaf nodes corresponding to the vertices of P, less than or equal to $(n-1)$ non-leaf nodes and less than or equal to $(2n - 3)$ arcs, (b) it partitions P into n connected faces; each face is a monotone polygon, and (c) in the boundary of each face, exactly one edge of the P appears.*

Proposition 2 *Let e_1 and e_2 be two edges incident on a vertex v. Let v' be the position of v at an instant of time during the shrinking process (before v vanishes due to edge/split-event). If r is the Euclidean distance $dist(v, v')$ between v and v', ϕ is the angle incident at v inside the polygon, and χ is the distance traveled by e_1 and e_2 in self-parallel manner to meet v', then $\chi = rsin(\frac{\phi}{2})$, and is referred to as shrinking parameter at that instant of time.*

The node of $S(P)$ generated by an edge-event $EE(e)$ is the point (ξ) of intersection of the bisectors of the angles of P incident to the end points of e.

A reflex vertex v may need to be tested with several edges to identify the exact edge e which participates in the split-event (with v). Let v be tested with the edges $e_1, e_2, \ldots, e_k$ and the corresponding event points be respectively $\xi_1, \xi_2, \ldots, \xi_k$. The split-event $SE(v, e_i)$ may occurs if $dist(v, \xi_i) \leq dist(v, \xi_j)$ for all $j = 1, 2, \ldots, k$. Lemma 6 says that a split-event may disappear due to an edge-event during the shrinking process.

3 The straight skeleton of a monotone polygon

A simple polygon P is said to be monotone with respect to a line L if its intersection (if any) with any line perpendicular to L is connected. Without loss of generality, we may assume that the line L is the Y-axis. The left (resp. right) monotone chain of P is the boundary of P from v_t to v_b in anti-clockwise (resp. clockwise) order where v_t and v_b are the vertices of P having maximum and minimum y-coordinate respectively. In the following lemmata we investigate the effect of monotonicity of a polygon in connection with the computation of its straight skeleton. The proof of the lemmata are omitted due to space constraint.

Lemma 3 *For any split-event in a monotone polygon, the reflex vertex and the edge being split always belong to different monotone chains.*

Lemma 4 *Let v_1 and v_2 be two reflex vertices belonging to the same monotone chain such that v_1 and v_2 participates in the split-events $SE(v_1, e_1)$ and $SE(v_2, e_2)$ respectively. If v_1 is above v_2 (i.e. having larger y-coordinate value), then either e_1 and e_2 are the same edge, or e_1 is above e_2.*

Lemma 5 *During the shrinking process of a monotone polygon, a vertex-event with the bisectors of two angles belonging to the same chain can not happen.*

Lemma 6 *The shrinking process never introduces a new reflex vertex.*

Lemma 7 *If ξ is the event point of $SE(v, e)$ then $d(\xi, \ell(e)) = \min\{d(\xi, \ell(f)); f \in P\}$, where $\ell(x)$ denote the line containing edge x and $d(\xi, \ell(f))$ is the perpendicular distance from ξ to $\ell(f)$.*

4 Algorithm

We create two arrays A_L and A_R with the vertices (and hence the edges) in the left and right monotone chains in a top to bottom order. In addition, we store the reflex vertices of the left and right chains in two arrays, namely REF_L and REF_R, in top to bottom order. The output of this algorithm is the skeleton tree $S(P)$ of the polygon P. We use a priority queue Q that stores the edge- and split-events; Q is maintained in the form of a min-heap in increasing order of the shrinking parameters of these events. Since a vertex-event can be considered as a split-event also, we need not have to identify the vertex-events separately. During the execution of the algorithm, we maintain a variable Δ which indicates the amount of shrinking of the polygon P up to the current instant of time.

An edge-event $EE(e)$ is stored in Q in the form of a tuple $('EE', e, \xi_1, \xi_2, \chi)$, where ξ_1 and ξ_2 are the two nodes of $S(P)$. When a node ξ is created in $S(P)$ at the time of processing $EE(e)$, ξ_1 and ξ_2 are connected with ξ. Initially, ξ_1 and ξ_2 contain the two end-points of the edge e in its original position [1]. Similarly, a split-event $SE(v, e)$ is stored in Q in the form of a tuple $('SE', v, e, \xi, \chi)$, where both v and e point to their own presence in A_L and A_R, and ξ is the node in $S(P)$ corresponding to the original position of the reflex vertex v. Each edge (resp. reflex vertex) in A_L and A_R also points to its corresponding edge-event (resp. split-event) in Q. Note that, after each edge-/split-event during the shrinking process, the edges of P get modified. But, we do not make these changes in A_L and A_R.

We also attach a height balanced binary tree $T(e)$ with each edge e in A_L and A_R. The reason is that the same edge may participate in different split-events with different vertices. Initially $T(e)$ contains e only. During the execution of a $SE(v, e)$, e is split into two parts. Each of these parts may further be split while executing another split-event $SE(v', e)$. These pieces of e are stored in $T(e)$.

[1]If e is a polygonal edge, then ξ_1 and ξ_2 are the two end-points of e. If e is generated by a edge- or a split-event at a point ξ^* (say), then coordinate of ξ^* is stored as the corresponding end-point of e.

The execution starts by computing $EE(e)$ for all edges e of P, and storing them in Q. We now explain the method of computing the split-events corresponding to the reflex vertices of P.

4.1 Identification of split-events

In order to identify the edge e associated to a split-event $SE(v, e)$ corresponding to a reflex vertex v, we may need to inspect several edges of the opposite chain (see Lemma 3). Let $\{v_1, v_2, \ldots, v_k\}$ be the set of reflex vertices in REF_L from top to bottom. First we compute the split-event corresponding to the vertex v_i, $i = \frac{1+k}{2}$, using Lemma 7 in $O(n)$ time. Let the split-event be $SE(v_i, e_j)$, and it partitions A_R in two parts A'_R and A''_R. We include appropriate portion of e_j in both A'_R and A''_R. After computation of $SE(v_i, e_j)$, the edge participating to the split-event of a reflex vertex in $\{v_1, v_2, \ldots, v_{i-1}\}$ will be a member in A'_R, and the edge participating to the split-event of a reflex vertex in $\{v_{i+1}, v_{i+2}, \ldots, v_k\}$ will be a member in A''_R (Lemma 4). Again, we find the split-event for the vertices $v_{i'}$ and $v_{i''}$ where $i' = \frac{i}{2}$ and $i'' = \frac{i+1+k}{2}$ in $O(n)$ time, and so on. Thus, the time complexity for computing all the split-events is $O(n \log n)$ (see Lemmata 4 and 7). The same method applies to compute the split-events for the reflex vertices in REF_R. All these split-events are stored in Q. Since, a reflex vertex may be destroyed after processing an edge-event (see Lemma 6), a split-event generated at the beginning or later stage of the shrinking process, may not occur during the further shrinking process. If such a situation takes place, the split-event corresponding to that vertex is deleted from Q.

4.2 Processing of edge/split-events

All the vertices of P are the leaf nodes in $S(P)$. Initially, Q contains the edge-events corresponding to all the edges in P and split-events corresponding to all the reflex vertices in P, Δ is set to 0. We choose an event having minimum shrinking parameter χ from Q, and perform one of the following procedures as stated below. The process continues until Q becomes empty. Observations 1 and 2, say that while processing an event some existing events will no longer be valid. These are to be deleted from Q. Also some events in Q may need to be modified during the shrinking process. The pointers attached to the vertices and edges of P in A_L and A_R help in accessing those events in Q.

Observation 1 *An edge-event $EE(e)$ for an edge e may not take place during the shrinking process. This happens if the shrinking parameter of a split-event $SE(v, e)$ for some reflex vertex v and the edge e is smaller than that of $EE(e)$. In this case, after $SE(v, e)$ the shrunken edge e is split into two pieces e_1 and e_2, and they lie in different sub-polygons. Here, after the occurrence of $SE(v, e)$, we delete $EE(e)$ from Q, and store the new events $EE(e_1)$ and $EE(e_2)$ in Q.*

Observation 2 *A split-event $SE(v,e)$ for a reflex vertex v may not take place during the shrinking process. This situation arises if one of the edges e' attached to v may disappear due to $EE(e')$ prior to $SE(v,e)$.*

Processing of an edge-event $EE(e)$**:** The tuple attached to $EE(e)$ is $('EE', e, \xi_1, \xi_2, \chi)$, where $e = [\alpha, \beta] \in A_L$; $e_1 = [\alpha, \gamma]$ and $e_2 = [\beta, \delta]$ are two edges adjacent to e. We compute the event point ξ, and create a node of $S(P)$ at ξ; join ξ with ξ_1 and ξ_2 to create two arcs of $S(P)$. The edge e is deleted from A_L, and the event $EE(e)$ is deleted from Q. If e_1 and e_2 changes to $e'_1 = [\xi, \gamma']$ and $e'_2 = [\xi, \delta']$ respectively after $EE(e)$, and χ_1 and χ_2 are the shrinking parameter of $EE(e'_1)$ and $EE(e'_2)$ respectively, then the shrinking parameter of $EE(e_1)$ and $EE(e_2)$ in Q are updated to $\Delta + \chi_1$ and $\Delta + \chi_2$ respectively. The ξ_1 parameter of $EE(e_1)$ and ξ_2 parameter of $EE(e_2)$ are also updated with ξ. Finally, Δ is updated with $\Delta + \chi$.

If the angle incident at one of the end-points, say α of $e = [\alpha, \beta]$ is reflex, then the corresponding split-event may be deleted from Q (see Lemma 6). Note that, if an edge-event $EE(e)$ occurs then the edge e is not involved in any split-event (see Lemma 7).

Processing of a split-event $SE(v,e)$**:** The tuple attached to $SE(v,e)$ is $('SE', v, e, \xi_1, \chi)$. We create a node of $S(P)$ at the event point ξ, join ξ with ξ_1, and then delete the event $SE(v,e)$ from Q. Next, we compute the perpendicular projection η of ξ on the edge e, and search in the tree $T(e)$ attached to the edge e to find the segment e^* of e that contains η. The root of $T(e)$ is reached using the pointer stored in the e field of the tuple. Let e^*_1 and e^*_2 be the two pieces of e^* generated after the split-event $SE(v,e)$, and e'_1 and e'_2 be the shrunken form of e^*_1 and e^*_2 after $SE(v,e)$. If χ_1 and χ_2 are the shrinking parameter of $EE(e'_1)$ and $EE(e'_2)$ respectively, then the shrinking parameter of $EE(e^*_1)$ and $EE(e^*_2)$ are $\Delta + \chi_1$ and $\Delta + \chi_2$ respectively. We now delete $EE(e^*)$ from Q, and insert two new events $EE(e^*_1)$ and $EE(e^*_2)$. If the tuple attached to $EE(e^*)$ is $('EE', e^*, p_1, p_2, \chi)$, then the tuple of $EE(e^*_1)$ and $EE(e^*_2)$ will be $('EE', e^*_1, p_1, \xi, \Delta + \chi_1)$ and $('EE', e^*_2, \xi, p_2, \Delta + \chi_2)$ respectively. Finally, we delete e^* and insert e^*_1, e^*_2 in $T(e)$. Finally, Δ is updated with $\Delta + \chi$.

Note that, at the time of processing a split-event $SE(v,e)$, we do not need to update the existing split-event involving e in the queue, because the other split-events automatically detect its proper position on e (see identification procedure of split-event in Section 4).

4.3 Complexity analysis

Computation of all the edge-events of P takes $O(n)$ time. From the discussions in identification of split-events in Section 4, the split-events for all reflex vertices in P, and all the vertex-events (if any) can be computed in $O(n\log n)$ time. As initially we have generated $O(n)$ events, the creation of the initial heap Q needs $O(n)$ time. While processing an $EE(e)$ or a $SE(v,e)$, at most two edge-events can be generated. In addition a $O(\log n)$ time search in $T(e)$ is required while processing a $SE(v,e)$. Moreover, the processing of an event also includes deleting at most two old events from Q and inserting at most two new events in Q, which requires $O(\log n)$ time. Since, the total number of internal nodes in the skeleton tree $S(P)$ is $O(n)$, the total number of events processed is also $O(n)$. Thus, the time complexity of our algorithm is $O(n\log n)$ in the worst case. Apart from storing the polygon during the shrinking process and the output skeleton tree $S(P)$, we also use (i) a priority queue Q for storing the event points, and (ii) the height balanced binary tree attached to each edge of P. The space required for both these data structures depends on the number of edge-events and split-events, which is $O(n)$ in the worst case.

References

[1] O. Aichholzer and F. Aurenhammer Straight skeletons for general polygonal figures in the plane. *International Computing and Combinatorics Conference*, 117-126, 1996.

[2] O. Aichholzer, F. Aurenhammer, D. Alberts and B. Gartner A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1:752-761, 1995.

[3] F. Aurenhammer Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys 23*, 3:345-405, 1991.

[4] F. Chin, J. Snoeyink and C. A. Wang Finding the medial axis of a simple polygon in linear time. *Discrete Computational Geometry*, 21:405-420, 1999.

[5] S. -W. Cheng and A. Vigneron Motorcycle graphs and straight skeletons. *Algorithmica*, 47:159-182, 2007.

[6] E. D. Demaine, M. L. Demaine and A. Lubiw Folding and one straight cut suffice. *Symposium on Discrete Algorithms*, 891-892. 1999.

[7] E. D. Demaine, M. L. Demaine and J. S. B. Mitchell Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Symposium on Computational Geometry*, 105-114, 1999.

[8] D. Eppstein Fast hierarchical clustering and other applications of dynamic closest pairs. *Symposium on Discrete Algorithms*, 619-628. 1998.

[9] D. Eppstein and J. Erickson Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete Computational Geometry*, 22:569-592, 1999.

[10] P. Felkel and S. Obdrzalek Straight skeleton implementation. *Spring Conference on Computer Graphics*, 210-218, 1998.

[11] D. G. Kirkpatrick Efficient computation of continuous skeletons. *IEEE Symposium on Foundation of Computer Science*, 18-27, 1979.