Time-Space Trade-offs for Triangulating a Simple Polygon*

Boris Aronov† Matias Korman‡ Simon Pratt§ André van Renssen¶‖ Marcel Roeloffzen¶‖

Abstract

An s-workspace algorithm is an algorithm that has read-only access to the values of the input, write-only access to the output and only uses $O(s)$ additional words of space. We give a randomized s-workspace algorithm for triangulating a simple polygon P of n vertices, for any $s \in \Omega(\log n) \cap O(n)$. The algorithm runs in $O(n^2/s)$ expected time. We also extend the approach to compute other similar structures such as the shortest-path map (or tree) of any point $p \in P$, or to partition P using only diagonals of the polygon so that the resulting sub-polygons have $\Theta(s)$ vertices each.

1 Introduction

Triangulation of a simple polygon, often used as a preprocessing step in computer graphics, is performed in a wide range of settings including on embedded systems like the Raspberry Pi or mobile phones. Such systems frequently run read-only filesystems for security reasons and have very limited working memory. An ideal triangulation algorithm for such an environment would allow for a trade-off in performance in time versus working space.

These memory constraints can be modeled by the so-called *s-workspace* model of computation frequently used in the literature (see, for example, [2, 5, 6, 10]). In this model the input data is given in a read-only array or similar structure, and the output we produce must be written to write-only memory.

In our case, the input is a simple polygon P; let $v_1, v_2, \ldots, v_n$ be the vertices of P in clockwise order along the boundary of P. We assume that, given an index i, in constant time we can access the coordinates of the vertex v_i. We also assume that the usual word RAM operations can be performed in constant time (such as, given i, j, k, finding the intersection point of the line passing through vertices v_i and v_j and the horizontal line passing through v_k).

In addition to the read-only data, an s-workspace algorithm can use $O(s)$ variables during its execution, for some parameter s determined by the user. Implicit memory consumption (such as the stack space needed in recursive algorithms) must be taken into account when determining the size of a workspace. We assume that each variable or pointer is stored in a data word of $\Theta(\log n)$ bits. Thus, equivalently, we can say that an s-workspace algorithm uses $O(s \log n)$ bits of storage. In this model, the aim is to design an algorithm whose running time decreases as s grows. Such algorithms are called *time-space trade-off* algorithms [14].

Previous Work

Several variants of this model have been studied (we refer the interested reader to [11] for an overview). In the following we discuss the results related to triangulations. The concept of memory-constrained algorithms was introduced to the computational geometry community by the work of Asano *et al.* [4]. Among other results, they presented an algorithm for triangulating a set of n points in $O(n^2)$ time using $O(1)$ variables. More recently, Korman *et al.* [12] introduced two different time-space trade-off algorithms for the same problem: the first one computes an arbitrary triangulation in $O(n^2/s + n \log n \log s)$ time using $O(s)$ variables. The second is a randomized algorithm that computes the Delaunay triangulation of the given point set in expected $O((n^2/s) \log s + n \log s \log^* s)$ time within the same space bounds.

The first algorithm for triangulating simple polygons was due to Asano *et al.* [2], and runs in $O(n^2)$ time using $O(1)$ variables. Faster algorithms for some particular cases (such as monotone polygons [5, 3]) are also known. To the best of our knowledge, no general time-space trade-off algorithm for simple polygons was previously known. In this paper we introduce a randomized algorithm with expected running time $O(n^2/s)$ that uses $O(s)$ variables to triangulate a simple n-gon, for any $s \in \Omega(\log n) \cap O(n)$. Our approach uses a recent result by Har-Peled [10], which computes the shortest path between two vertices of a simple polygon in expected $O(n^2/s)$ time. Due to lack of space, proofs in this paper are omitted or sketched. Details can be found in the extended version [1].

*Work on this paper by B. A. has been partially supported by NSF Grants CCF-11-17336 and CCF-12-18791. M. K. was supported in part by the ELC project (MEXT KAKENHI No. 24106008). S. P. was supported in part by the Ontario Graduate Scholarship and The Natural Sciences and Engineering Research Council of Canada.

†Tandon School of Engineering, New York University, New York, USA.

‡Tohoku University, Sendai, Japan.

§Cheriton School of Computer Science, University of Waterloo, Canada.

¶National Institute of Informatics (NII), Tokyo, Japan.

‖JST, ERATO, Kawarabayashi Large Graph Project.

2 Preliminaries

We study the problem of computing a triangulation of a simple polygon P in the s-workspace model. A *triangulation* of P is a maximal crossing-free straight-line graph whose vertices are the vertices of P and whose edges lie inside P. Since our workspace is not large enough to store the triangulation explicitly, the goal is to report a triangulation of P in a write-only data structure. After a value is reported, it cannot be accessed or modified.

In preceding similar research [2, 3], the triangulation is reported as a list of edges in no particular order (with no information on neighboring edges or faces). Moreover, it is not clear how to modify these algorithms to obtain such information. Our approach has the advantage that, in addition to the list of edges, we can report adjacency information as well. More details can be found in [1].

Given two points $p, q \in P$, the *geodesic* between them is defined as the shortest path that connects p and q and that stays within P (viewing P as a closed set). The length of that path is called the *geodesic distance*. It is well known that, for any two points of P, their geodesic π always exists and is unique (hence, the geodesic is also often simply referred as the *shortest path* between p and q). Moreover, such a path is a polygonal chain whose vertices (other than p and q) are reflex vertices of P. Thus, we often identify π with the ordered sequence of reflex vertices traversed by the path from p to q. When that sequence is empty (i.e., the shortest path consists of the straight segment pq) we say that p *sees* q (and vice versa).

3 Algorithm

Let π be the geodesic connecting v_1 and $v_{\lfloor n/2 \rfloor}$. From a high-level perspective, the algorithm uses the approach of Har-Peled [10] to compute π, and reports the edges of the shortest path one by one, in order. Our aim is to use this path to subdivide P into smaller subproblems that can be solved recursively.

We start by introducing some definitions that will help in recording which portion of the polygon has already been triangulated. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ split the boundary of P into two chains. We say that a vertex (other than v_1 and $v_{\lfloor n/2 \rfloor}$) is a *top* vertex if it is in the chain that is traversed when walking along the boundary of P from v_1 to $v_{\lfloor n/2 \rfloor}$ in clockwise fashion or a *bottom* vertex if it lies in the other chain. Note that all vertices, other than v_1 and $v_{\lfloor n/2 \rfloor}$ are either top or bottom vertices. We say that a diagonal c is an *alternating* diagonal if one of its endpoints is a top vertex and the other a bottom vertex (or one of its vertices is either v_1 or $v_{\lfloor n/2 \rfloor}$). Otherwise we say that the diagonal is a *non-alternating* diagonal.

We will use these diagonals to partition P into two parts. Since any two vertices consecutive along the boundary of P can see each other, the partition induced by the "diagonal" connecting them is trivial (i.e., one subpolygon is P and the other is a segment).

Observation 1 *Let c be a diagonal of P such that neither endpoint is v_1 or $v_{\lfloor n/2 \rfloor}$. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ belong to different components of $P \setminus c$ if and only if c is an alternating diagonal.*

Corollary 1 *Let c be a non-alternating diagonal of P. The component of $P \setminus c$ that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$ has at most $\lceil n/2 \rceil$ vertices.*

While triangulating the polygon, we maintain an alternating diagonal a_c. Intuitively, the connected component of $P \setminus a_c$ that does not contain $v_{\lfloor n/2 \rfloor}$ has already been triangulated. Since it will prove useful that a_c is not necessarily part of π, we also maintain the property that at least one of the endpoints of a_c will be a vertex of π that has already been computed in the execution of the shortest-path algorithm. Let v_c denote the endpoint of a_c that is on π and that is closest to $v_{\lfloor n/2 \rfloor}$.

With these definitions in place, we can give an intuitive description of our algorithm: we start by setting a_c as the degenerate diagonal from v_1 to v_1. We then use the shortest-path computation approach of Har-Peled. Our aim is to walk along π until we find a new alternating diagonal a_{new}. At this moment we pause the execution of the shortest-path algorithm, triangulate the subpolygons of P that have been created (and contain neither v_1 nor $v_{\lfloor n/2 \rfloor}$) recursively, update a_c to the newly found alternating diagonal, and then resume the execution of the shortest-path algorithm.

Although our approach is intuitively simple, there are several technical difficulties that must be carefully considered. Ideally, the number of diagonals we walked along π is small and can be stored explicitly. But if we do not find an alternating diagonal in just a few steps (indeed, it could even be that there is no alternating diagonal in π), we need to use other diagonals. We also need to make sure that the complexity of each recursive subproblem is reduced by a constant fraction, that we never exceed space bounds, and that no part of the triangulation is reported more than once.

Recall that, at any instant of time, v_c denotes the endpoint of a_c that is in π, and that the subpolygon defined by a_c containing v_1 has already been triangulated. Let $w_0, \ldots, w_k$ be the portion of π up to the next alternating diagonal. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_k, \ldots, v_{\lfloor n/2 \rfloor})$ where $w_0w_1, \ldots, w_{k-2}w_{k-1}$ are non-alternating diagonals, and $w_{k-1}w_k$ is an alternating diagonal (or $w_k = v_{\lfloor n/2 \rfloor}$ if no pair of vertices consecutive on π between v_c and $v_{\lfloor n/2 \rfloor}$ forms an alternating diagonal).

Consider the partition of P that these diagonals create, see Figure 1. Let P_1 be the subpolygon induced

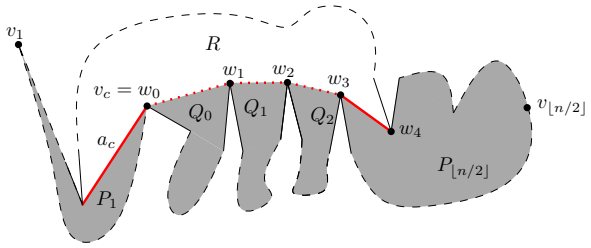

Figure 1: Partitioning P into subpolygons P_1, $P_{\lfloor n/2 \rfloor}$, R, Q_1, …, Q_{k-2}. The two alternating diagonals are marked by thick red lines.

by a_c that does not contain $v_{\lfloor n/2 \rfloor}$. Similarly, let $P_{\lfloor n/2 \rfloor}$ be the subpolygon that is induced by the alternating diagonal $w_{k-1}w_k$ and does not contain v_1. For any $i < k-1$ we define Q_i as the subpolygon induced by the non-alternating diagonal w_iw_{i+1} that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$. Finally, let R be the remaining component of P. Note that some of these subpolygons may be degenerate and consist only of a line segment (for example, when w_iw_{i+1} is an edge of P).

Lemma 2 *Each of the subpolygons R, Q_1, Q_2, …, Q_{k-2} has at most $\lceil n/2 \rceil + k$ vertices. Moreover, if $w_k = v_{\lfloor n/2 \rfloor}$, then the subpolygon $P_{\lfloor n/2 \rfloor}$ has at most $\lceil n/2 \rceil$ vertices.*

This result allows us to treat the easy case of our algorithm. When k is small (say, a constant number of vertices), we can pause the shortest-path computation algorithm, explicitly store all vertices w_i, recursively triangulate R as well as the subpolygons Q_i (for all $i \leq k-2$), update a_c to the edge $w_{k-1}w_k$, and resume the shortest-path algorithm.

Handling the case where k is large is more involved. Note that we do not know the value of k until we find the next alternating diagonal, but we need not compute it directly. We will be given a parameter τ related to the workspace allowed for our algorithms, and when $k > \tau$, we say that the path is *long*. Initially we set $\tau = s$ but the value of this parameter will change as we descend the recursion tree. We say that the distance between two alternating diagonals is *long* whenever we have computed τ vertices of π besides v_c and no pair of consecutive vertices forms an alternating diagonal. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_\tau, \ldots v_{\lfloor n/2 \rfloor})$ and $w_0w_1, \ldots, w_{\tau-1}w_\tau$ are all non-alternating diagonals. In particular, the vertices $w_0, \ldots, w_\tau$ must form a convex chain (see Figure 1). Rather than continue walking along π, we look for a vertex u of P that together with w_τ forms an alternating diagonal. Once we have found this diagonal, we will partition P into $\tau - 2$ subpolygons using the diagonals $a_c, w_0w_1, w_1w_2, \ldots, w_{\tau-1}w_\tau$, and uw_τ similarly to the easy case: P_1 is the part induced by a_c which does not contain $v_{\lfloor n/2 \rfloor}$, $P_{\lfloor n/2 \rfloor}$ is the part induced by uw_τ which does not contain v_1, Q_i is the part induced by the edge w_iw_{i+1} on its boundary, which contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$, and R is the remaining component.

Lemma 3 *We can find a vertex u that together with w_τ forms an alternating diagonal in $O(n)$ time using $O(1)$ space. Moreover, each of the subpolygons R, Q_1, Q_2, …, $Q_{\tau-2}$ has at most $\lceil n/2 \rceil + \tau$ vertices.*

Proof sketch. We use ray shooting to find an edge e outside P_1 which is partially visible to w_τ. Let p_N be one of the endpoints of e. Note that p_N need not be visible to w_τ. However, the triangle formed by w_τ, p_N, and the visible point of e contains one or more reflex vertices. Among those vertices, we know that the vertex r that maximizes the angle $\angle p_N w_\tau r$ must be visible (see Lemma 1 of [6]). As described in Lemma 1 of [6], in order to find such a reflex vertex we need to scan the input polygon at most three times, each time storing a constant amount of information. □

At high level, our algorithm walks from v_1 to $v_{\lfloor n/2 \rfloor}$. We stop after walking τ steps or when we find an alternating diagonal (whichever comes first). This generates several subproblems of smaller complexity that are solved recursively. Once the recursion is done we update a_c (to keep track of the portion of P that has been triangulated), and continue walking along π. The walking process ends when the walk reaches $v_{\lfloor n/2 \rfloor}$. In this case, in addition to triangulating R and the Q_i subpolygons, we must also triangulate $P_{\lfloor n/2 \rfloor}$.

The algorithm on the deeper levels of recursion is almost identical. There are only three minor changes that need to be introduced. First, we compare the size of the polygon to τ rather than s. Recall that τ denotes the amount of space available to the current instance of the algorithm. Thus, if τ is comparable to n (say, $10\tau \geq n$), then the whole polygon fits into memory and can be triangulated in linear time [7]. If τ is significantly smaller, then we continue with the recursive algorithm as usual.

For ease in handling subproblems, at each step we also indicate the vertex that fulfills the role of v_1 (i.e., one of the vertices from which the shortest path must be computed). Recall that we have random access to the vertices of the input. Thus, once we know which vertex takes the role of v_1, we can find the vertex that will satisfy the role of $v_{\lfloor n/2 \rfloor}$ in constant time as well.

In order to avoid exceeding the space bounds, at each level of the recursion we will decrease the value of τ by a factor of $\kappa < 1$.

Theorem 4 *Let P be a simple polygon of n vertices. We can compute a triangulation of P in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

4 Other Applications

The above algorithm introduces a general approach for partitioning P into subpolygons, each of which has at most $O(s)$ vertices. Since our final objective is computing a triangulation, at the bottom level of recursion we use Chazelle's algorithm [7]. However, the same approach can be used for other structures: it suffices to replace the base case of the recursion with an appropriate algorithm. In this section, we mention two examples: computing the shortest-path map and splitting the polygon into pieces of size $\Theta(n)$.

Given a simple polygon P and a point $p \in P$ (which need not be a vertex of P), the *shortest-path tree* of p (denoted by $\mathrm{SPT}(p)$) is the tree formed as the union of all shortest paths from p to vertices of P. ElGindy [8] and later Guibas *et al.* [9] showed how to compute the shortest-path tree in linear time.

The *shortest-path map* of p (denoted by $\mathrm{SPM}(p)$) is the subdivision of P into maximal cells so that points in the same cell have topologically equivalent paths to p. It is well known that $\mathrm{SPM}(p)$ is a finer subdivision than the one induced by $\mathrm{SPT}(p)$. Guibas *et al.* [9, Section 2] showed how to further refine the shortest-path tree so as to obtain the shortest-path map. We refer the interested reader to Lee and Preparata [13] or Guibas *et al.* [9] for more information on shortest-path trees, maps, and their applications.

Theorem 5 *Let P be a simple polygon of n vertices and let p be any point of P (vertex, boundary or interior). We can compute the shortest-path map or shortest-path tree of p in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

Theorem 6 *Let P be a simple polygon of n vertices. For any $s \leq n$, we can partition P with $\Theta(n/s)$ diagonals, so that each subpolygon consists of $\Theta(s)$ vertices, in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

We note that both Asano *et al.* [2] and Har-Peled [10] already gave methods of partitioning P into subpolygons of roughly the same size. The first one is deterministic, runs in $O(n^2)$ and uses $O(1)$ variables. The one of Har-Peled is a proper trade-off and also runs in $O(n^2/s)$ expected time using $O(s)$ variables. This method introduces additional Steiner points. Our algorithm removes the need for these additional points (since it partitions only by diagonals between visible vertices), while preserving the same running time.

5 Acknowledgments

The authors would like to thank Jean-François Baffier, Man-Kwun Chiu, and Takeshi Tokuyama for valuable discussion in the early stages of the paper.

References

[1] B. Aronov, M. Korman, S. Pratt, A. van Renssen, and M. Roeloffzen. Time-space trade-offs for triangulating a simple polygon. *CoRR*, abs/1509.07669, 2015.

[2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *CGTA*, 46(8):959–969, 2013.

[3] T. Asano and D. Kirkpatrick. Time-space trade-offs for all-nearest-larger-neighbors problems. In *WADS*, pages 61–72, 2013.

[4] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *JoCG*, 2(1):46–68, 2011.

[5] L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

[6] L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing the visibility polygon using few variables. *CGTA*, 47(9):918–926, 2013.

[7] B. Chazelle. Triangulating a simple polygon in linear time. *DCG*, 6:485–524, 1991.

[8] H. A. ElGindy. *Hierarchical Decomposition of Polygons with Applications*. PhD thesis, McGill University, Montreal, Que., Canada, 1985.

[9] L. 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(1–4):209–233, 1987.

[10] S. Har-Peled. Shortest path in a polygon using sublinear space. In *SoCG*, pages 111–125, 2015.

[11] M. Korman. Memory-constrained algorithms. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–7. Springer Berlin Heidelberg, 2015.

[12] M. Korman, W. Mulzer, M. Roeloffzen, A. v. Renssen, P. Seiferth, and Y. Stein. Time-space trade-offs for triangulations and voronoi diagrams. In *WADS*, pages 482–494, 2015.

[13] D. T. Lee and F. P Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[14] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.