Generating Random x-Monotone Polygons with Given Vertices

Chong Zhu[1] Gopalakrishnan Sundaram[2] Jack Snoeyink[1,4] Joseph S. B. Mitchell[3,5]

Abstract

There are many possible definitions for random geometric objects such as polygons. This paper considers a definition that is motivated by the desire to generate random test instances for problems in geographic information systems. Given a set S_n of n points in the plane, no two with the same x-coordinate, generate uniformly at random one of the possible x-monotone polygons with vertex set S_n. We give an algorithm that uses $O(n)$ space and $O(K)$ time, where $n < K < n^2$ is the number of edges of the visibility graph of the x-monotone chain with vertex set S_n. We also discuss generating nested polygons and the difficulty of removing the word "monotone."

1 Introduction

This paper details some results that we have obtained in our study of generating random polygons. In particular, we describe an algorithm for generating x-monotone polygons uniformly at random. As well as being of theoretical interest, the generation of random geometric objects has applications which include the testing and verification of time complexity for computational geometry algorithms. In order to have some control over the characteristics of the output, we would like to fix the vertex set and then generate uniformly at random a simple polygon with the chosen vertices.

Others have considered generating random simple polygons by various processes that move vertices (e.g. [6]). When the vertices are fixed, Epstein [1] gives an $O(n^4)$ algorithm to generate triangulation of a given simple polygon at random. Meijer and Rappaport [5] studied monotone traveling salesmen tours and show that the number of x-monotone polygons on n vertices is between $(2+\sqrt{5})^{(n-3)/2}$ and $(\sqrt{5})^{(n-2)}$.

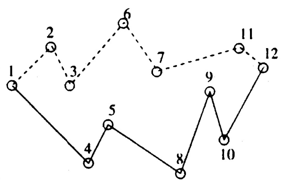

Figure 1: Monotone chains

Let $S_n = \{s_1, s_2, ..., s_n\}$ be a set of n points sorted according to their x coordinate. We assume, in this paper, that no two points have the same x-coordinates. We use a Real RAM model of computation. We generate a random monotone polygon by scanning S_n forward and counting all monotone polygons, then picking a random number and scanning backward to generate the polygon of that number. ("Monotone" will always mean x-monotone in this paper.) Thus, after some initial definitions, we count monotone polygons in section 1.1 and generate one in section 1.2. Our algorithms depend on the size of visibility graph of the monotone chain joining the vertices of S_n, so we show in section 2 how to generate the visibility edges forwards and backwards.

Let $S_i = \{s_1, s_2, \ldots, s_i\}$ for $1 \le i \le n$. Let $N(i)$ denote the number of monotone polygons with vertex set S_i. Any monotone polygon constructed from S_i can be divided into two monotone chains of which the leftmost vertex is s_1 and rightmost vertex is s_i. Points s_1 and s_i are on both chains; any other point in S_i is on either the top or bottom chain.

[1]Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada
[2]Center for Mathematics Research, University of Montreal, Montreal, Quebec, Canada
[3]Department of Applied Math & Statistics, SUNY Stony Brook, NY, USA
[4]Supported in part by an NSERC Research Grant and a B.C. Advanced Systems Institute Fellowship.
[5]Supported in part by grants from Boeing Computer Services, Huges Research Laboratories, Air Force Office of Scientific Research contract AFOSR-91-0328, and by NSF Grants ECSE-8857642 and CCR-9204585

For convenience, we frequently denote the line segment or polygon edge $\overline{s_i s_j}$ by (i, j) and the line $\overleftrightarrow{s_i s_j}$ by $\ell(i, j)$

1.1 Counting Monotone Polygons

We begin by establishing a recurrence that counts the monotone polygons on S_i in terms of those on S_j for $j < i$.

Note that a monotone polygon with vertex set S_i has edge $(i-1, i)$ as one of the two edges incident to s_i. Let $T(i)$ be the set of monotone polygons with vertex set S_i that have edge $(i-1, i)$ on their top chain and define the number of these polygons $TN(i) = |T(i)|$. Similarly, let $B(i)$ be the set of monotone polygons with vertex set S_i that have edge $(i-1, i)$ on their bottom chain and define $BN(i) = |B(i)|$. Our recurrence will actually count $TN(i)$ in terms of $BN(j)$ for $j < i$.

Lemma 1.1 *For any point set S_k with $k > 2$, the number of monotone polygons with vertices S_k is*

$$N(k) = TN(k) + BN(k) \tag{1}$$

We say that a point s_i is *above-visible* from s_k if $i < k$ and s_i is above the line $\ell(j, k)$, for all points s_j with $i < j < k$. Similarly, s_i is *below-visible* from s_k if $i < k$ and s_i is below $\ell(j, k)$, for $i < j < k$. Let $V_T(i)$ be the set of points that are *above-visible* from point s_i, and let $V_B(i)$ be the set of points that are *below-visible* from point s_i. We can now count monotone polygons on S_k where both edges into s_k are specified.

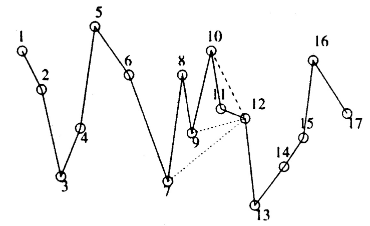

Figure 2: $V_T(12) = \{10\}$ and $V_B(12) = \{7, 9\}$

Lemma 1.2 *The number of polygons in $T(k)$ that contain edge (j, k), for $j \in V_B(k)$, is $BN(j+1)$. The number $B(k)$ that contain edge (j, k), for $j \in V_T(k)$, is $TN(j+1)$.*

Proof: Let $P(j, k)$ be the set of polygons in $T(k)$ with (j, k) as a bottom edge, for $j \in V_B(k)$. For the polygons in $P(j, k)$, we know that points s_j and s_k are on the bottom chains, and $s_{j+1}, \ldots, s_k$ are on the top chains. So the path of $s_j, s_k, s_{k-1}, \rightsquigarrow s_{j+1}$ is fixed. We can treat this path as an edge $(j, j+1)$ that is on the bottom chain. Figure 3 shows an example. Thus, $|P(j, k)|$ equals the number of monotone polygons generated from S_{j+1} with the edge $(j, j+1)$ on the bottom chains, which is BN(j+1). ∎

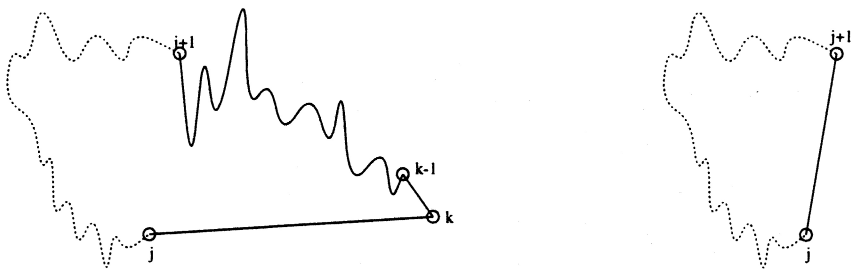

Figure 3: The original polygon and its equivalent set in $B(j+1)$.

Theorem 1.3 *For any point set S_k with $k > 2$, we have*

$$TN(k) = \sum_{j \in V_B(k)} BN(j+1) \tag{2}$$

$$BN(k) = \sum_{j \in V_T(k)} TN(j+1) \tag{3}$$

Proof: We prove formula 2. According to the definition of below-visible, the bottom edge (j,k) of any $P \in T(k)$ uses a point $s_j \in V_B(k)$. By lemma 1.2 there are $BN(j+1)$ monotone polygons having edges $(k-1,k)$ and (j,k). Therefore, we have $\sum_{j \in V_B(k)} BN(j+1)$ polygons in total. ∎

This theorem gives us a procedure to calculate TN and BN, assuming that we have $V_B(k)$ and $V_T(k)$. We can start with $TN(2) = BN(2) = 1$, since in the degenerate case of two vertices the line segment can be considered as the top and the bottom edge of a degenerate polygon. Then we use the recurrence to determine $TN(i)$ and $BN(i)$ for $i := 3$ to n.

1.2 Generating Monotone Polygons Uniformly

Once we have $TN(i)$ and $BN(i)$, for all $i \leq n$, we can generate a monotone polygon on vertex set S_n uniformly at random. Again, we assume that we have $V_B(k)$ and $V_T(k)$, the *below-visible* and *above-visible* vertices. The `Generate()` algorithm scans the point set S_n from the right to the left to generate monotone polygons.

```
Generate($S_n$)
    PICK $x \in [1, N(n)]$ uniformly at random;
    ADD $s_n$ TO top_chain;
    ADD $s_n$ TO bottom_chain;
    IF $x \leq TN(n)$
        ADD $s_{n-1}$ TO top_chain;
        Generate_Top($n, x$);
        ADD $s_1$ TO bottom_chain;
    ELSE
        $x = x - TN(n)$;
        ADD $s_{n-1}$ TO bottom_chain;
        Generate_Bottom($n, x$);
        ADD $s_1$ TO top_chain;
    END IF
```

```
Generate_Top($k, x$)
1.  IF $k \leq 2$ RETURN;
2.  FIND SMALLEST $i$ THAT SATISFIES:
            $x \leq \sum_{(j \in V_B(k)) \wedge (j \leq i)} BN(j+1)$;
3.      ADD POINT $s_i$ TO bottom_chain;
4.      ADD $s_{k-2}, s_{k-3}, \ldots, s_{i+1}$ TO top_chain;
5.      $k = i + 1$;
6.      $x = x - \sum_{(j \in V_B(k)) \wedge (j < i)} BN(j+1)$;
7.  Generate_Bottom($k, x$)
```

`Generate_Top()` and `Generate_Bottom()` are two mutually recursive procedures. `Generate_Top()` deals with the case in which s_{k-1} and s_k are on the top chain. `Generate_Bottom()` can be obtained from `Generate_Top()` by swapping all "top"s and "bottom"s.

`Generate()` determines a one-to-one correspondence between the integers in $[1, N(n)]$ and the monotone polygons $\{P_1, P_2, \ldots, P_{N(n)}\}$ that can be generated from S_n. Thus, it generates monotone polygons uniformly at random.

Theorem 1.4 *For $n \geq 2$ and $\forall x \in [1, TN(n)]$,* `Generate_Top()` *generates a unique monotone polygon $P_x \in T(n)$; For $n \geq 2$ and $\forall x' \in [1, BN(n)]$,* `Generate_Bottom()` *generates a unique monotone polygon $P_{x'} \in B(n)$.*

2 Computing Visibility

The algorithms of the previous section assumed that the *above-visible* and *below-visible* sets, $V_T(i)$ and $V_B(i)$ for $i = 1, \ldots, n$, were available. A closer look, however, shows that these sets are only needed for one index i at a time: the algorithm to obtain TN and BN needs the sets in increasing order and algorithms `Generate_Top()` and `Generate_Bot()` need them in decreasing order.

We can obtain $V_T(i)$ incrementally using the following idea. Let S_k denote the monotone chain with vertices $s_1, s_2, \ldots, s_k$. If we think of S_k as a fence and compute the shortest paths in the plane above S_k

from s_k to each s_i with $i \leq k$, then we obtain a tree that is known as the *shortest path tree rooted at* s_k [3, 4]. The *above-visible* set $V_T(i)$ is exactly the set of children of s_k in the shortest path tree rooted at s_k. Thus, we will incrementally compute shortest path trees rooted at $s_1, s_2, \ldots, s_k$ to get the *above-visible* sets.

We represent shortest path trees (in which a node may have many children) by binary trees in which each node has pointers to its uppermost child and next sibling. Section 2.1 gives the details for computing these trees in the forward direction: computing $V_T(i)$ from $V_T(i-1)$. Section 2.2 gives the details for the reverse direction: computing $V_T(i)$ from $V_T(i+1)$.

2.1 Computing Visibility Forward

We store *top_tree*(i) and *bot_tree*(i) using child and sibling pointers. For each vertex $j \in [1, n]$, we have a record for *top_tree*

j:	*ptr*	*ptr* stores the coordinates of vertex j
	upc	*upc* is a pointer pointing the upper child of j in *top_tree*(k)
	sib	*sib* is a pointer pointing the sibling of j in *top_tree*(k)

The initial value of *top_tree* is $1.ptr = s_1$, $1.upc = nil$ and $1.sib = nil$. We assume that *top_tree*$(i-1)$ has been computed and call the procedure Make_top$(i-1, i, tmp)$ to calculate the *top_tree*(i)—$i.upc$ will be set to $tmp.sib$.

```
Make_top($j, k$, Var:$lastsib$)
    WHILE $j.upc \neq nil$ and $k$ is above $\ell(j.upc, j)$
        Make_top($j.upc, k$, Var:$lastsib$);
        /* make subtree for this child of $j$, which can be seen by $k$. */
        $j.upc = j.upc.sib$; /* consider next child of $j$ */
    END WHILE
    $lastsib.sib = j$; /* make the connection to $j$, one of the children of $k$ */
    $lastsib = j$;
```

To compute the *bot_tree* is similar to computing the *top_tree*. Knowing *top_tree*(k) and *bot_tree*(k), we know the *above-visible* and *below-visible* point sets, $V_T(k)$ and $V_B(k)$ of vertex k. Now we give the theorem to show us how to get $V_T(k)$ from *top_tree*(k).

Let r be a record in the *top_tree*. We define that $r.sib^i = r.sib^{i-1}.sib$, for any integer $i \geq 0$, and $r.sib^0 = r$. Then we know that the upper child of k and its siblings are this kind of format. Now we claim that the upper child of k and its siblings are the vertices visible from k, and any vertex that is visible from k is either the upper child of k or its sibling. This is proved in the next theorem.

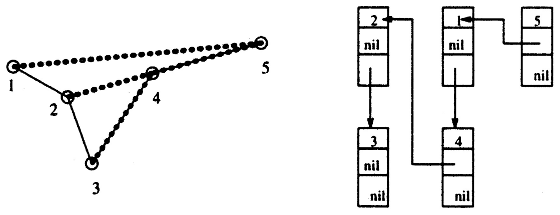

Figure 4: Point set S_5 and *top_tree*(5)

Theorem 2.1 *Let $CT(k)$ be the set of points $\{j \mid j = k.upc.(sib)^i$ for some $i\}$. (The children of k in the shortest path tree.) We have $V_T(k) = CT(k) - \{k-1\}$.*

2.2 Computing Visibility backward

In procedure Generate_Top() and Generate_Bottom(), we need to find the smallest i in line 2. Here we assume that *top_tree*$(k+1)$ and *bot_tree*$(k+1)$ have been completed, we use procedures **Back_top** and **Back_bot** to generate *top_tree*(k) and *bot_tree*(k).

Let $t_{i-j} = (k+1).upc.sib^j$, for $j = 0, 1, \ldots, i$. Then $t_i = (k+1).upc$ and $t_0 = k$. Let $Q = \{t_j, j = 0, \ldots, i\}$. From theorem 2.1, we know $Q = V_T(k+1) - \{k\}$. If we take t_0 as the origin of coordinates, according to the *above-visible* definition, the points in Q are sorted lexicographically by polar angle and distance from t_0. Then from a Graham-Scan [2] convex hull computation we can get the correct *top_tree*(k). One example to calculate *top_tree*(k) from *top_tree*$(k+1)$ is shown in Figure 5. The details are not hard, but are omitted due to space constraints [7].

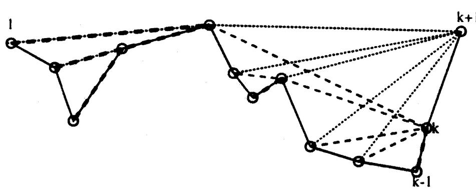

Figure 5: *top_tree*(k) is generated from *top_tree*$(k+1)$.

3 Time and Space Complexity Analysis

Now we have all the procedures to build up our algorithm. Next we give its time and space complexity, beginning with `Make_top()` and `Back_top()`.

Lemma 3.1 *The runtime of* `Make_top`$(k-1, k,$ `Var`$:t)$ *or of* `Back_top`$(k-1, k)$ *is* $O(|V_T(k)|)$.

Proof: Each call to `Make_top()`, except the first, implies that the calling procedure found a child of k. All other work in the procedure takes constant time per call.

In `Back_top`$(k-1, k)$, which computes *top_tree*$(k-1)$ from *top_tree*(k), ■

Theorem 3.2 *Our algorithm has time complexity of* $O(K)$ *and space in* $O(n)$. *where* K *is the total number of above-visible and below-visible points.*

4 Generating nested monotone polygons

We can modify our algorithm to generate, on a given vertex set S_n, a random x-monotone polygon that is nested inside another x-monotone polygon P. All we need to change is the definition and computation of visibility.

We say that s_i is *below-visible* from s_k if $i < k$, the line segment (i, k) does not intersect the exterior of P, and s_i is below $\ell(j, k)$, for $i < j < k$. Similarly, s_i is *above-visible* from s_k if $i < k$, the line segment (i, k) does not intersect the exterior of P, and s_i is above $\ell(j, k)$, for $i < j < k$.

The visible sets $V_T(k)$ and $V_B(k)$ under this new definition of visibility can be computed both forward and backwards in time proportional to their size with a time and space overhead of $O(n + |P|)$. The additional computation is essentially to compute the *relative convex hull* of P [3, 8] and S_k up to the vertical line through the point s_k.

Theorem 4.1 *One can count the monotone polygons having vertex set* S_n *that are nested inside a monotone polygon* P *in* $O(n + |P|)$ *space and* $O(n + |P| + K)$ *time, where* K *is the total number of above-visible and below-visible points.*

5 Conclusion

In this paper we have used a definition of random polygons that separates the choice of vertex set from the choice of edges. We have shown how to efficiently generate, uniformly at random, x-monotone polygons that have a given n-point set S_n as their vertices. Our algorithm runs in $O(n)$ space and $O(K)$ time, where $n < K < n^2$ is the number of edges of the visiblity graph of the monotone chain on S_n. This algorithm allows us to generate nested x-monotone polygons for testing GIS algorithms.

We would like to be able to generate simple polygons with vertex set S_n uniformly at random as well. This appears difficult to do efficiently. One can, of course, generate permutations at random and check for simplicity. The worst-case for this approach occurs when the points are in convex position—only $2n$ of the $n!$

permutations correspond the the convex hull, which is the only simple polygon. There is, to our knowledge, no efficient enumeration procedure for simple polygons.

One approach that leads to a polynomial time algorithm is to generate a random permutation and then apply 2-opt moves to pairs of intersecting edges—removing two intersecting edges and replacing them with two non-intersecting edges so as to keep the polygon connected. It is a Putnam problem to observe that this replacement decreases total length and therefore converges to a simple polygon. van Leeuwen and Schoone [9] showed that at most $O(n^3)$ of these "untangling 2-opt" moves can be applied in any order; their proof is particulary nice if one considers the geometric dual. Unfortunately, as figure 6 illustrates, this approach does not give simple polygons uniformly at random.

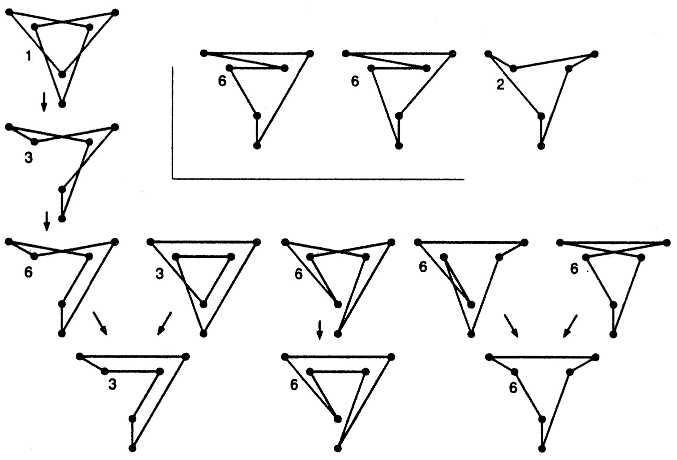

Figure 6: Polygons obtained by "untangling 2-opt" for the given six vertices. Numbers represent multiplicity due to symmetries of the point set.

Acknowledgement

Figure 6 was developed with Sandor Fekete—we thank him for discussions on generating random simple polygons.

References

[1] P. Epstein and J. Sack. Generating triangulation at random. In *Proceedings of the Fourth Canadian Conference on Computational Geometry*, pages 305–310, 1992.

[2] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

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

[4] Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, October 1989.

[5] Henk Meijer and David Rappaport. Upper and lower bounds for the number of monotone crossing free hamiltonian cycles from a set of points. *ARS Combinatoria*, 30:203–208, 1990.

[6] Joseph O'Rourke and Mandira Virmani. Generating random polygons. Technical Report 011, Smith College, *1991*.

[7] Jack Snoeyink and Chong Zhu. Generating random monotone polygons. Technical Report 93-28, Department of Computer Science, University of British Columbia, September 1993.

[8] Godfried T. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3(2):9–42, 1989. Also available as technical report SOCS 88.20, School of Computer Science, McGill University.

[9] J. van Leeuwen and Anneke A. Schoone. Untangling a travelling salesman tour in the plane. In J. R. Mühlbacher, editor, *Proc. 7th Conf. Graphtheoretic Concepts in Comput. Sci. (WG 81) (Linz 1981)*, pages 87–98, München, 1982. Hanser.