Generating Triangulations at Random

Extended Abstract

Peter Epstein, Jörg-Rüdiger Sack[§]

School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6

Abstract

We describe an $O(n^3)$ algorithm to count triangulations of a simple polygon and an $O(n^4)$ algorithm to generate triangulations of a simple polygon at random with a uniform probability distribution. As well, we relate the problem of counting triangulations to existing graph theory problems.

1 Introduction

This paper details some recent results we have obtained in our study of geometrical probability. In particular, we describe an algorithm for generating triangulations of a polygon uniformly at random. The remainder of this section provides motivation for this research and develops notation for describing probability distributions of geometric object generators. In Section 2, we go on to describe our algorithm. A summary of our results and related open problems are presented in Section 3.

1.1 Motivation

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.

Algorithm Testing

The most direct use for a stream of geometric objects generated at random is for testing computational geometry algorithms. We can test such algorithms in two ways. The first involves the construction of geometric objects that the implementer considers difficult cases for the algorithm. For example, a polygon triangulation algorithm based on a plane sweep may find polygons with horizontal or vertical edges require special case code, making such polygons likely candidates for exposing errors. The second approach to testing involves executing the algorithm on a large set of geometric objects generated at random. We expect errors to be exposed if enough different valid inputs are applied to the algorithm. Although a uniform distribution is not essential, it is important that any one of the valid inputs may be generated. If there exists a valid input that cannot be generated, then this input may be the only one for which the algorithm fails.

Examples of algorithms that would benefit from a generator of geometric objects at random include:

- Algorithms to triangulate simple polygons
- Algorithms to sort Jordan sequences.
- Algorithms to find shortest paths or shortest path trees in a triangulated polygon.
- Algorithms to compute visibility polygon in a triangulated polygon.

Many researchers in computational geometry construct streams of geometric objects generated at random to test implementations of their algorithms. Fairly crude algorithms are often used to generate these random objects since their distributions are not important. However, if these algorithms are not carefully designed, it is possible for the resulting geometric objects to share some properties that the tested algorithm can take advantage of, leading to inadequate testing.

Verification of Time Complexity

In implementation-oriented computational geometry research, we are often given the problem of verifying that an implementation of an algorithm achieves the stated algorithm time complexity. This is done by timing the

§ Research partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

execution of the algorithm for various inputs of different sizes. There are a variety of possible inputs of any given size, and the choice is important, since an algorithm may take more time on some inputs than others of the same size. If an average execution time is computed over a set of randomly generated objects of a given size, the relationship between time and problem size will typically follow a curve corresponding to its complexity. We can then check this complexity against the stated algorithm's complexity.

For example, consider a linear time visibility algorithm that accepts a simple polygon as input. While some polygons have many spirals, making the algorithm perform complex steps, other polygons of the same size are convex, making the algorithm trivial. If we are to get a meaningful graph of time versus polygon size, it is important that the algorithm is tested on many different random polygons of any given size.

1.2 Notation

We refer to a *probability space* as (Ω,E,Pr), where Ω is the *sample space*, E is the *event space*, and Pr is the *probability function*. The sample space is the set of all *elementary events* which are the possible outcomes of the experiment being described. The event space is the set of all subsets of Ω that are assigned a probability. The function $Pr:E \rightarrow \Re_0^+$ defines the probability of events.

We define a simple polygon as $P = (v_0,v_2,\ldots,v_{n-1})$, where $v_0,v_2,\ldots,v_{n-1}$ are the vertices of P in clockwise order. We define $P - v_i = (v_0,\ldots,v_{i-1},v_{i+1},\ldots,v_{n-1})$ to be the polygon P with the vertex v_i removed. We define the *sub-polygons* of P as $P(i,j) = (v_i,v_{i+1},\ldots,v_j)$ if $i < j$, and $P(i,j) = (v_i,\ldots,v_{n-1},v_0,\ldots,v_j)$ if $i > j$, where $i,j \in [0,n-1]$. The sub-polygons of P are the polygons produced by adding an edge between a pair of mutually visible vertices of P. We define N(P) to be the number of triangulations of the simple polygon P. We refer to edges of a polygon triangulation of P as (v_i,v_j). We say a polygon is in *standard form* if all its vertices are distinct and no three consecutive vertices are collinear.

A *geometric object generator* is an algorithm that produces a stream of geometric objects of a given type. We say that a generator is *complete* if it can produce every object in a given sample space Ω.

1.3 The Uniform Probability Distributions

Probability theory defines both *discrete* and *continuous* uniform probability distributions. We are interested only in the discrete case:

Definition The *discrete uniform probability space* for a finite sample space Ω_U is defined as (Ω_U,E_U,Pr_U), where E_U is the set of all subsets of Ω_U, and $Pr_U(A) = 1/|\Omega_U|$ for all $A \in \Omega_U$.

In other words, in a finite sample space, a uniform distribution is one in which each elementary event is equally likely.

2 Generating Triangulations of a Simple Polygon at Random

Generating triangulations of a convex polygon uniformly at random can be done in optimal linear time (see [Atkinson & Sack 92]). We now describe our $O(n^4)$ algorithm for arbitrary simple polygons.

2.1 Definition of Uniformity

Since the sample space is finite, we use the discrete uniform probability distribution:

Definition All sets of triangulations are *events*. A polygon triangulation generator is *uniform* if each of the triangulations of the polygon has the same probability of being generated.

2.2 Simple Techniques

Two approaches for generating triangulations of a simple polygon are described in this section that do not provide uniformity. Both techniques use the set E of possible triangulation edges.

In the first method, we simply generate a uniform random ordering E' (i.e., a random permutation) of the set E, and add edges from E' to construct the triangulation. If the edge being added crosses edges already added, then the edge is skipped.

In the second method, we let $E_1 = E$. In the i^{th} step, we select an edge e uniformly at random from E_i, add the edge to the triangulation, and compute E_{i+1} from E_i by removing edges that cross the edge e.

Examples show that neither of these techniques provides a uniform probability distribution.

2.3 Computing the Number of Possible Triangulations

The number of triangulations of a convex polygon is fairly straightforward to compute using the Catalan numbers. For arbitrary simple polygons, counting the number of triangulations can be done using Algorithm 1 to follow. The algorithm has the additional property that it computes the number of triangulations of all sub-polygons, which we will find to be crucial for our algorithm.

Algorithm 1

Input	A simple polygon $P = (v_0, v_2, \ldots, v_{n-1})$ in standard form.
Output	$N(P(i,j))$ for all $i,j \in [0,n-1]$.
Complexity	$O(n^3)$ time $O(n^2)$ space, requires $O(n)$ bit word size.

We use dynamic programming, computing values of $N(P(i,i+k))$ for increasing values of k, storing the results in an $n \times n$ array T. All index calculations are modulo n. We consider all possible triangles that can be formed from the edge (v_i, v_{i+k}), and count the number of triangulations including each such triangle (v_i, v_j, v_{i+k}). For a given triangle, the number of triangulations is $N(P(i,j))N(P(j,i+k))$ (see Figure 1).

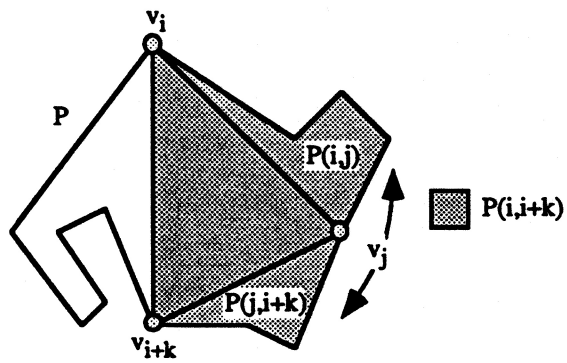


Figure 1 Counting triangulations of P(i,i+k) that include the triangle (v_i, v_j, v_{i+k}).

```
1   FOR i := 0 TO n-1 DO
2       IF v_i and v_{i+2} are mutually visible in P THEN
3           LET T[i,i+2] := 1.
4       ELSE
5           LET T[i,i+2] := 0.
6       END IF.
7   END FOR.
8   FOR k := 3 TO n-1 DO
9       FOR i := 0 TO n-1 DO
10          IF v_i and v_{i+k} are mutually visible in P THEN
11              LET T[i,i+k] := T[i,i+k-1] + T[i+1,i+k].
12              FOR j := i+2 TO i +k-2 DO
13                  LET T[i,i+k] := T[i,i+k] + (T[i,j])(T[j,i+k]).
14              END FOR.
15          ELSE
16              LET T[i,i+k] := 0.
17          END IF.
18      END FOR.
19  END IF.
20  RETURN T.
```

Lines 1 to 7 find all ears of P. Lines 8 to 19 compute $N(P(i,i+k))$ for increasing values of k. Line 11 counts triangulations of P(i,i+k) that include either the triangle (v_i, v_{i+1}, v_{i+k}) or the triangle $(v_i, v_{i+k-1}, v_{i+k})$. All other possible triangles for the vertices v_i and v_{i+k} are counted in lines 12 to 14.

Lemma 1 *Algorithm 1 computes* T[i,j] = N(P(i,j)) for all integers i,j ∈ [0,n-1] *for any simple* n-*gon* P.

Proof

Let P be any simple n-gon. If v_i and v_{i+k} are mutually visible in P then the algorithm computes:

$$T[i,i+k] = T[i,i+k-1] + T[i+1,i+k] + \sum_{j=i+2}^{i+k-2} (T[i,j])(T[j,i+k])$$

We use induction on k. If k = 2 then we have triangles. Such polygons clearly have only one triangulation, so N(P(i,i+2)) = 1, as given by lines 6 to 10 of the algorithm. If k > 2, then we assume that T[i,i+k'] = N(P(i,i+k')) for all k' < k. We partition the triangulations of P(i,i+k) into k-1 sets according to which vertex is the third vertex of the triangle with vertices v_i and v_{i+k}. This is clearly a partition since any triangulation of P(i,i+k) lies in exactly one of these sets. Consider a set in this partition which includes those triangulations containing the triangle (v_i,v_j,v_{i+k}) for some j. Every triangulation in this set can be constructed by triangulating the two polygons $(v_i,v_{i+1},\ldots,v_j)$ and $(v_j,v_{j+1},\ldots,v_{i+k})$, so the number of triangulations in the set is

$$N(P(i,i+k-1)) + N(P(i+1,i+k)) + \sum_{j=i+2}^{i+k-2} (N(P(i,j)))(N(P(j,i+k)))$$

By induction, the algorithm computes this using the previous results in T.

If v_i and v_{i+k} are not mutually visible in P then there can be no triangulation of the sub-polygon P(i,i+k), so the algorithm correctly computes T[i,i+k] = N(P(i,i+k)) = 0 since lines 1 to 5 initialize all elements of the T array to zero, and lines 7 and 13 ensures that no change is made to this initial value. □

Theorem 1 *Algorithm 1 computes* N(P(i,j)) *for all integers* i,j ∈ [0,n-1] *for any simple* n-*gon* P *in* $O(n^3)$ *time.*

Proof omitted for brevity.

Corollary 1 *Algorithm 1 computes the number of triangulations of any simple* n-*gon in* $O(n^3)$ *time.*

Proof omitted for brevity.

2.4 Generating Triangulations at Random

Generator G_1

Input	A simple polygon P = $(v_0,v_2,\ldots,v_{n-1})$ in standard form.
Output	A triangulation of P.
Properties	Complete, uniform probability distribution.
Complexity	$O(n^4)$ time, $O(n^2)$ space, requires O(n) bit word size.

We will use Algorithm 1, to compute the number of triangulations of P and of all polygons produced by adding a triangulation edge to P in $O(n^3)$ time.

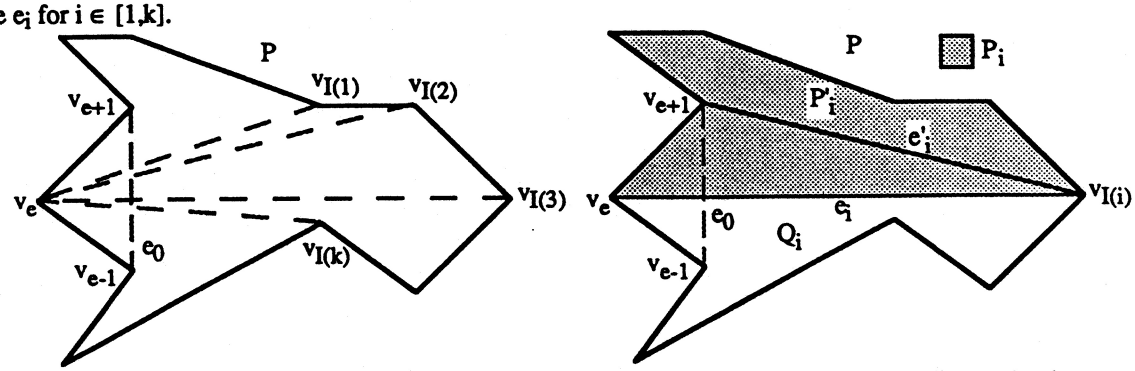
Given the ability to count triangulations of a polygon in polynomial time, we can easily count the number of triangulations that include a given edge E by multiplying the number of triangulations of each of the smaller polygons produced by adding the edge E. One might expect this to lead directly to an algorithm to generate uniformly random triangulations of P. However, difficulties arise if we choose not to include E, since this implies that at least one edge that crosses E must be included in the triangulation.

The algorithm is recursive, so we consider adding various edges which break the polygon into parts, each part being either a triangle or a smaller polygon. If n = 3, then P is a triangle, so the only possible triangulation has no internal edges. This is the way our recursion will terminate.

If n > 3, we start by finding a convex vertex v_e whose neighbors v_{e-1} and v_{e+1} are mutually visible (see Figure 2a). At least two such vertices must exist by the well known Two Ear Theorem. We could triangulate P by cutting off v_e, and using $e_0 = (v_{e-1},v_{e+1})$ as a new polygon edge. Let $\{v_{I(1)},v_{I(2)},\ldots,v_{I(k)}\}$ be the vertices visible from v_e, excluding v_{e-1} and v_{e+1}, in clockwise order around the polygon, starting from v_e. Let $e_i = (v_e,v_{I(i)})$ for i ∈ [1,k]. Let $e'_i = (v_{e+1},v_{I(i)})$ for i ∈ [1,k], if that edge is a possible triangulation edge. Let $P_i = (v_e,v_{e+1},\ldots,v_{I(i)})$ and $Q_i = (v_e,v_{I(i)},\ldots,v_{e-1})$ be the polygons produced by adding edge e_i to P (see Figure 2b). Further, let P'_i be $P_i - v_e = (v_{e+1},\ldots,v_{I(i)})$.

We now consider the addition of the ear edge e_0 to a triangulation of P. If the triangulation is to include e_0 then it cannot include any edge incident on v_e. Conversely, if the triangulation does not include e_0 then it must include

some edge incident on v_e. If we choose not to include e_0 we must now carefully consider the selection of such an edge e_i for $i \in [1,k]$.



a) Vertex labelling of P from the ear v_e. b) The corresponding edge labelling and polygons.

Figure 2 Vertex and edge labelling from an ear.

Let T be the set of all triangulations of P. Let T_0 be the subset of T in which the edge e_0 is present. Let T_i be the subset of T in which e_i is present and no edge e_j is present where $j \in [1,i)$. A given T_i may be empty, since it may be that some edge e_i cannot be the extreme edge incident on v_e. This will be the case if the edge e'_i is not a possible triangulation edge. We will prove that $\{T_0,T_1,\ldots,T_k\}$ forms a partition of T (see Lemma 2). So,

$$|T| = \sum_{i=0}^{k} |T_i|, \quad |T_0| = N(P - v_e), \quad \text{and } |T_i| = \big(N(P'_i)\big)\big(N(Q_i)\big) \text{ for } i \geq 2.$$

To generate a uniformly random triangulation, we select a partition T_i, and then recursively select a random triangulation uniformly within that partition. To select a partition, we compute the sizes of the partitions using Algorithm 1, and then select one such that the probability of selecting any given T_i is $|T_i|/|T|$. If any given set T_i is empty, then we never select such an i. Once T_i has been selected, proceed as follows:

Case 1 If $i = 0$ then add edge e_0 to result, and recursively triangulate the convex polygon $P - v_e$.

Case 2 If $i = 1$ and $I(1) = e+2$ then add edge e_1 to result, and recursively triangulate the convex polygon $P - v_{e+1}$.

Case 3 Otherwise add edges e_i and e'_i to result. Recursively triangulate the polygons P'_i and Q_i.

Lemma 2 *$\{T_0,T_1,\ldots,T_k\}$ forms a partition of T.*

Theorem 2 *G_1 is complete.*
Proofs omitted for brevity.

Theorem 3 *G_1 provides a uniform probability distribution.*
Proof

We must prove that each triangulation of P is generated with the same probability. Specifically, this probability is $1/N(P)$.

We use induction on n (the size of P). Our base case is $n = 3$, in which there is only one triangulation, and our algorithm finds it. In this case, the probability should be $1/N(P) = 1/1 = 1$, as expected. We now assume $n > 3$. We assume that the algorithm provides a uniform probability distribution for any polygon with fewer than n vertices. Let t be an arbitrary triangulation of P. We will show that G_1 will construct t with probability $1/N(P)$.

Case 1 If t includes edge e_0: The algorithm will construct t if and only if it selects $i = 0$ *and* it selects the triangulation of $P - v_e$ that, when augmented with the edge e_0 gives t. The algorithm selects $i = 0$ with probability $|T_0|/|T| = N(P - v_e)/N(P)$ by Lemma 2. By induction, all possible triangulations of $P - v_e$ will be constructed with equal probability. Specifically, any one triangulation of $P - v_e$ will be constructed with probability $1/N(P - v_e)$. These are independent events, so we multiply their probabilities to get the probability that t is generated, $\big(N(P - v_e)/N(P)\big)\big(1/N(P - v_e)\big) = 1/N(P)$, as required.

Case 2 If t excludes edge e_0 but includes edge e_1, and $I(1) = e+2$: The algorithm will construct t if and only if it selects $i = 1$ *and* it selects the triangulation of $P - v_{e+1}$ that, when augmented with the edge e_1 gives t. As in

Case 1, this implies that t is generated with probability $(N(P-v_{e+1})/N(P))(1/N(P-v_{e+1}))=1/N(P)$, as required.

Case 3 Otherwise, t must include some edge e_a such that no edge e_b exists for any $b \in [1,a)$. The algorithm will construct t if and only if it selects $i = a$ *and* it selects the appropriate triangulations of P'_i and Q_i, which, when combined and augmented with edges e_a and e'_a form t. The algorithm selects $i = a$ with probability $|T_a|/|T| = (N(P'_i))(N(Q_i))/N(P)$ by Lemma 2. By induction, all possible triangulations of P'_i will be constructed with equal probability. Specifically, any one triangulation will be constructed with probability $1/N(P'_i)$. Similarly, all possible triangulations of Q_i will be constructed with probability $1/N(Q_i)$. These are independent events, so we multiply their probabilities to get the probability that t is generated,

$$\left(\frac{(N(P'_i))(N(Q_i))}{N(P)}\right)\left(\frac{1}{N(P'_i)}\right)\left(\frac{1}{N(Q_i)}\right)=\frac{1}{N(P)}$$

as required. □

Theorem 4 G_1 *has time complexity* $O(n^4)$.

Proof omitted for brevity.

Generating triangulations of a *convex* polygon takes less time because counting the number of triangulations of convex polygons is easier than for arbitrary simple polygons.

3 Conclusions

While brute-force algorithms to generate triangulations of a polygon uniformly at random require exponential time, we have developed an $O(n^4)$ polynomial time algorithm that works for arbitrary simple polygons. While efficient algorithms exist for convex polygons [Atkinson & Sack 92], we know of no other polynomial time algorithm for arbitrary simple polygons. See [Epstein 92] for a complete algorithm that generates triangulations in $O(n^2)$ time, but does not provide a uniform probability distribution. A random triangulation generator is useful for testing the many algorithms that accept a triangulated simple polygon as input. Examples of such algorithms include shortest path and visibility query algorithms.

3.1 Open Problems

We are interested in finding a more efficient algorithm to generate triangulations of a simple polygon with a uniform probability distribution. The intersection graph of the possible triangulation edges of a simple polygon is known to be a *circle graph* (as defined, for example, in [Urrutia 80]), and the maximum independent sets of this intersection graph is known to correspond to the triangulations of the polygon (see [Epstein 92]). We are therefore interested in efficiently counting the number of maximum independent sets of a circle graph.

We are also interested in finding a polynomial time algorithm to generate triangulations of a point set with a uniform probability distribution. We have not found a property of point set intersection graphs, such as the circle graph property for polygon intersection graphs, which would allow the number of maximum independent sets to be computed in polynomial time. As well, we do not have an algorithm for generating triangulations of a point set uniformly in polynomial time given the ability to count point set triangulations in polynomial time.

4 Acknowledgments

The authors would like to thank Bruce Richter of Carleton University for his insight into the relationship between triangulations and circle graphs.

5 References

[Atkinson & Sack 92] M. D. Atkinson and J.-R. Sack, *Generating binary trees at random*, Information Processing Letters, Volume 41, pp. 21-23, North-Holland, 1992.

[Epstein 92] P. Epstein, *Generating Geometric Objects at Random*, Master's Thesis, School of Computer Science, Carleton University, April 1992.

[Urrutia 80] J. Urrutia, *Intersection Graphs of Some Families of Plane Curves*, Ph.D. Thesis, University of Waterloo, 1980.