The Visual Computer

Efficient triangulation of simple polygons

Godfried Toussaint

School of Computer Science, McGill University,
3480 University Street, Montreal, Quebec,
Canada H3A 2A7
e-mail: godfried @opus.cs.mcgill.ca.

This paper considers the topic of efficiently triangulating a simple polygon with emphasis on practical and easy-to-implement algorithms. It also describes a new *adaptive* algorithm for triangulating a simple n-sided polygon. The algorithm runs in time $O(n(1+t_0))$ with $t_0<n$. The quantity t_0 measures the *shape-complexity* of the *triangulation* delivered by the algorithm. More precisely t_0 is the number of obtained triangles contained in the triangulation that share zero edges with the input polygon and is, furthermore, related to the shape-complexity of the *input* polygon. Although the worst-case complexity of the algorithm is $O(n^2)$, for several classes of polygons it runs in linear time. The practical advantages of the algorithm are that it is simple and does not require sorting or the use of balanced tree structures. On the theoretical side, it is of interest because it is the first polygon triangulation algorithm where the *computational* complexity is a function of the *output* complexity. As a side benefit, we introduce a new measure of the complexity of a polygon triangulation that should find application in other contexts as well.

Key words: Polygon – Algorithm – Triangulation – Computational geometry – Geometric Complexity

1 Introduction

We are concerned with triangulating a special type of polygon in the Euclidean plane E^2 referred to as a *simple* (also *Jordan*) polygon. For any integer $n\geq 3$, we define a *polygon* or *n-gon* in the Euclidean plane E^2 as the figure $P=[x_1, x_2, \ldots, x_n]$ formed by n points $x_1, x_2, \ldots, x_n$ in E^2 and n line segments $[x_i, x_{i+1}]$, $i=1, 2, \ldots, n-1$, and $[x_n, x_1]$. The points x_i are called the *vertices* of the *polygon* and the line segments are termed its *edges*. A polygon P is called a *simple* polygon provided that no point of the plane belongs to more than two edges of P and the only points of the plane that belong to precisely two edges are the vertices of P. A simple polygon has a well defined interior and exterior. We will follow the convention of including the interior of a polygon when referring to P. The boundary of P in this case will be referred to as $bd(P)$.
Our problem is that of constructing a *triangulation* of P, i.e., decomposing P into a set of non-overlapping triangles (where their interiors do not intersect) without adding new vertices. Mathematicians have been interested in constructive proofs (algorithms) of the existence of triangulations for simple polygons as early as 1911 (Lennes). The "algorithm" of Lennes works by recursively inserting diagonals between pairs of vertices of P and runs in $O(n^2)$ time. Since then this type of "algorithm" has very often reappeared in a score of papers and text books during the past 70 years surprisingly containing fundamental errors. See the paper by Ho (1975) for a series of counter-examples to published triangulation "proofs." A rather different inductive proof was offered more recently by Meisters (1975). He proposed a method based on searching for "ears" and "cutting" them off. We call a vertex x_i of polygon P a *principal* vertex provided that no vertex of P lies in the interior of the triangle $[x_{i-1}, x_i, x_{i+1}]$ or in the interior of the diagonal $[x_{i-1}, x_{i+1}]$. A *principal* vertex x_i of a simple polygon P is called an *ear* if the diagonal $[x_{i-1}, x_{i+1}]$ that bridges x_i lies entirely in P. We say that two ears x_i and x_j are *non-overlapping* if $int[x_{i-1}, x_i, x_{i+1}] \cap int[x_{j-1}, x_j, x_{j+1}]=\varnothing$. The following *Two-Ears* theorem was proved by Meisters (1975).

Theorem *Two-Ears: Except for the triangles, every simple polygon P has at least two non-overlapping ears.*

A straightforward implementation of this idea leads to a complexity of $O(n^3)$. However, it was recently discovered that a prune-and-search technique will actually find an ear in linear time, thus

The Visual Computer (1991) 7:280–295

yielding an $O(n^2)$ implementation of Meisters' algorithm (ElGindy et al. in press). A good subpolygon of a simple polygon P, denoted by GSP, is a subpolygon whose boundary differs from that of P in at most one edge. A *proper ear* of a good subpolygon GSP is an ear of GSP which is also an ear of P. One of the key observations in ElGindy et al. is that a good subpolygon has at least one proper ear. The strategy of their algorithm is as follows. Given a polygon P on n vertices, split it in $O(n)$ time into two subpolygons such that one of these subpolygons is a good subpolygon with at most $\lfloor n/2 \rfloor + 1$ vertices. This splitting step is the crucial step in the algorithm. Subsequently, apply the algorithm recursively to this good subpolygon, which is guaranteed to have a proper ear. The worst-case running time of the algorithm is given by the recurrence $T(n) = cn + T(\lfloor n/2 \rfloor + 1)$, where c is a constant, which has solution $T(n) \in O(n)$.

The first algorithm to break the $O(n^2)$ upper bound was that of Garey et al. (1978). Their algorithm runs in time $O(n \log n)$, which is the time required by the first step to decompose the polygon into monotone sub-polygons. Then they apply an algorithm for triangulating monotone polygons in linear time. Note that a simpler linear-time algorithm for the latter problem is now available (Toussaint 1984). An alternate decomposition method with the same complexity appears in Fournier and Montuno (1984). An entirely different *divide-and-conquer* approach by Chazelle (1982) also achieves an $O(n \log n)$ upper bound. Finally, this upper bound was reduced even further by Tarjan and Van Wyk (1988). With very complicated and sophisticated data structures, they are able to triangulate a simple polygon in $O(n \log \log n)$ time. However, recently the same complexity was demonstrated using simple data structures (Kirkpatrick 1990).

Until May 1990 (Chazelle), one of the most outstanding open problems in computational geometry has been to determine if a simple polygon can be triangulated in $O(n)$ time. As an alternative, some researchers searched for large classes of polygons that can be triangulated in linear time. Such classes include *monotone* polygons (Garey et al. 1978; Toussaint 1984), *star-shaped* polygons (Schoone and van Leeuwen 1980; Woo and Shin 1985), *edge-visible* polygons (Toussaint and Avis 1982), *spiral* polygons (Feng and Pavlidis 1975; Toussaint 1986), *L-convex* polygons (ElGindy et al. 1983), *intersection-free* polygons (Lee and Chwa 1987), *weakly-externally-visible* polygons (ElGindy 1985), *palm-shaped* polygons (ElGindy and Toussaint 1984, 1985), and *anthropomorphic* polygons (Toussaint 1988). In yet another approach to the problem, researchers designed adaptive algorithms that run fast in many situations. Hertel and Mehlhorn (1975) have described a sweep-line-based algorithm that performs better the fewer reflex vertices it has. The running time of their method is $O(n + r \log r)$, where r denotes the number of reflex vertices of P. Hertel and Mehlhorn's algorithm takes the first step towards obtaining an adaptive algorithm sensitive to the *shape* of the polygon. Unfortunately, r is not a truly relevant measure of the shape complexity. To see this it is sufficient to realize that given any polygon of no matter what shape it is a trivial matter to insert n vertices (one betwen every original pair) and pull them an infinitesimal amount towards the interior of the polygon. Such a transformation will make r proportional to n without changing the basic shape of the polygon.

Chazelle and Incerpi (1984) took a further step to achieve a time complexity that more faithfully reflects the *shape complexity* of the polygon. They describe a triangulation algorithm that runs in time $O(n \log s)$ with $s < n$. The quantity s measures the *sinuosity* of the polygon, i.e., the number of times the polygon's boundary alternates between complete spirals of opposite orientation. Unlike r, s has the advantage that in many practical situations it is very small or a constant even for very winding polygons. Consider the motion of a straight line $L[x_i, x_{i+1}]$ passing through edge $[x_i, x_{i+1}]$ as i goes from 1 to $n-1$. Every time $L[x_i, x_{i+1}]$ reaches the vertical position in a clockwise (respectively counter-clockwise) manner, we increment (respectively decrement) a *winding-counter* by one. $L[x_i, x_{i+1}]$ is said to be *spiraling* (respectively *anti-spiraling*) if the winding counter is never decremented (respectively incremented) twice in succession. In this way, the polygon may be decomposed easily in $O(n)$ time into spiraling and anti-spiraling polygonal chains. An example of a polygon with a sinuosity of five is shown in Fig. 1a. Note that a new polygonal chain is restarted only when the previous chain ceases to be spiraling or anti-spiraling. The *sinuosity* s of P is defined as the number of polygonal chains thus obtained.

The Chazelle-Incerpi algorithm is much more interesting theoretically than the algorithm of Hertel and Mehlhorn, because of the implications it has on the complexity of triangulating different known

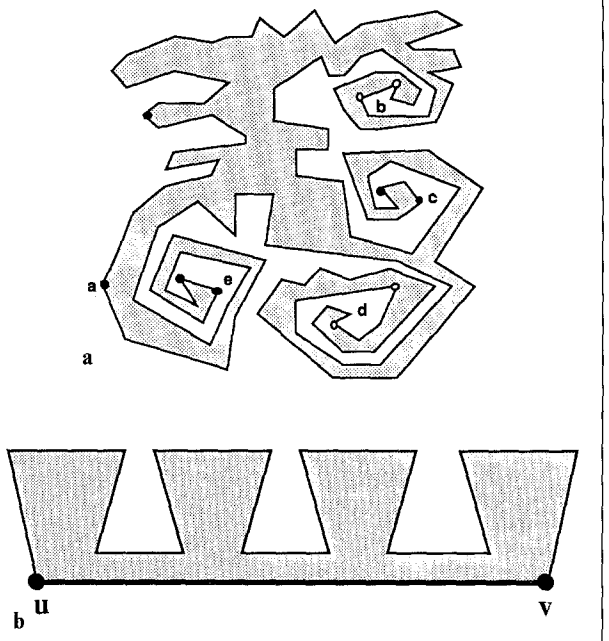

Fig. 1. (a) This fairly complicated-looking polygon has a sinuosity of only five. Scanning starts at *a* in a clockwise fashion. **(b)** A polygon *edge-visible* from *uv* with a sinuosity of $O(n)$

classes of polygons. Because r, the number of reflex vertices, is independent of whether a polygon is *monotonic*, *star-shaped*, *edge-visible* or whatever, Hertel and Mehlhorn's algorithm can run in $O(n \log n)$ time for these classes of polygons, for which linear time algorithms are known. On the other hand, *star-shaped* polygons have a sinuosity of one and thus the Chazelle-Incerpi algorithm runs in linear time for these polygons. Furthermore the algorithm makes no use of the *kernel* of P. In Schoone and van Leeuwen (1988) and Woo and Shin (1985), a point in the *kernel* is required and this implies a non-trivial (although linear time) effort. For a completely different and extremely simple algorithm for triangulating a *star-shaped* polygon without making use of the *kernel* of P, see ElGindy and Toussaint (1988, 1989). However, the sinuosity is not completely satisfactory as a measure of the shape complexity. It has the disconcerting property that it can vary by an order of magnitude depending on the orientation of the input polygon. Consider the *edge-visible* polygen illustrated in Fig. 1b. Recall that a polygon P is edge visible if there exists an edge $[u, v]$ of P such that for each point x in P there exists a point y in $[u, v]$ such that the line segment $[x, y]$ lies in P. The sinuosity for the polygon in Fig. 1b is $O(n)$ and thus the Chazelle-Incerpi algorithm runs in $O(n \log n)$ time on this polygon, whereas a linear-time algorithm exists (Toussaint and Avis 1982). Furthermore, by rotating the polygon through an angle of 90° the sinuosity reduces to $O(1)$. This represents an order of magnitude change in the *sinuosity* of P for no change in the *shape* of P (naturally we assume shape is invariant under translation and rotation).

Finally, we mention a new adaptive algorithm discovered recently (Kong et al. in press) that is based on the Graham scan. The Graham scan is a fundamental backtracking technique in computational geometry which was originally designed to compute the convex hull of a set of points in the plane (Graham 1972) and has since found application in several different contexts. In Kong et al., it is shown how to use the Graham scan to triangulate a simple polygon in $O(kn)$ time where $k-1$ is the number of concave vertices in P. Although the worst-case running time of the algorithm is $O(n^2)$ and hence not as good asymptotically as the algorithm of Hertel and Mehlhorn, it is much easier to implement and is therefore of practical interest. In fact, together with the algorithm presented in this paper, it is probably the best route to take in practice. A simple test to determine for a given polygon what the value of k is will determine which algorithm to use. If k is small, use the Kong et al. algorithm, if it is large use the algorithm proposed in this paper. For completeness and availability we include a full description of this algorithm. For a proof of correctness and a complexity analysis, the reader is referred to Kong et al.

The algorithm adapts the Graham scan in the following manner. The vertices of the polygon are scanned in order starting with x_2. At each step, the current vertex is tested to see if it is an ear. If it is not an ear, then the current vertex is advanced. If it is an ear, then the ear is cut off; that is, a diagonal is added to the triangulation and a vertex is deleted from the polygon. The current vertex is not advanced in this case, except in the special case that the ear is the vertex following x_0. This prevents x_0 from being cut as an ear.

To illustrate the execution of the algorithm, consider the polygon in Fig. 2. Initially, the algorithm tests x_1 and determines that it is not an ear (note that this is equivalent to testing whether x_2 is the top of an ear). The scan is advanced through x_2,

x_3, x_4 and x_5, at which time x_5 is determined to be an ear. Next x_5 is cut and then x_4 is tested and found not to be an ear. The next vertex tested is x_6. It is found to be an ear and cut. Again x_4 is tested and this time it is an ear so it is cut. The remaining vertices will be cut in the order x_7, x_3, x_8, x_2, x_9, x_1.

Algorithm. Triangulate (P): The algorithm takes as input a simple polygon $P=[x_1, x_2, \ldots, x_n]$, stored as a doubly linked circular list. SUCC(x_i) and PRED(x_i) indicate the successor and predecessor of x_i respectively. The algorithm produces a set D of diagonals comprising a triangulation of P. R is a set containing all the concave vertices of P. IsAnEar(P, R, x_i) is a function which returns true if x_i is an ear in polygon P and false otherwise.

```
1. x_i ← x_2;
2. while (x_i is not equal to x_0) do
3.    if (IsAnEar(P, R, PRED(x_i)) and P is not
         a triangle then  {PRED(x_i) is an ear.}
4.    D ← D ∪ (PRED(PRED(x_i)), x_i)
         {Store a diagonal.}
5.    P ← P − PRED(x_i)  {Cut the ear.}
6.    if x_i ∈ R and x_i is a convex vertex then
         {x_i has become convex.}
7.       R ← R − x_i
8.    if PRED(x_i) ∈ R and PRED(x_i) is a convex
         vertex then
         {PRED(x_i) has become convex.}
9.       R ← R − PRED(x_i)
10.   if (PRED(x_i) = x_0) then
         {SUCC(x_0) was cut.}
11.      x_i ← SUCC(x_i) {Advance the scan.}
12.   else x_i ← SUCC(x_i)  {PRED(x_i) is not an
         ear or P is a triangle. Advance the scan.}
13. end while
END Triangulate
FUNCTION IsAnEar(P, R, x_i):
1. if R = ∅ then return true
      {P is a convex polygon}
2. else if x_j is a convex vertex then
3.       if triangle (PRED(x_j), x_j, SUCC(x_j))
            contains no vertex of R then
4.          return true
5.       else return false
6.       else return false
End IsAnEar
```

In May 1990, Chazelle (1990) finally showed that a simple polygon of n vertices could be triangulated in $O(n)$ time. This discovery is a significant theoretical breakthrough. As a result there is not much merit from the *theoretical time-complexity* point of view in proposing algorithms that are adaptive and only sometimes run in $O(n)$ time unless they contribute also to a *new* theoretical perspective. However, Chazelle's linear time algorithm appears to be difficult to program and thus it is not clear presently if it will have practical consequences. In this light, the adaptive algorithm just described and the one proposed in this paper constitute contributions to the practical efficiency of triangulating polygons. These algorithms are easy to describe and program and they run fast in practice.

In this paper, we describe a new algorithm for triangulating a simple n-sided polygon. The algorithm runs in time $O(n(1+t_0))$ with $t_0<n$. The quantity t_0 measures the complexity of the triangulation delivered by the algorithm. More precisely, t_0 is the number of triangles obtained in the output triangulation that share zero edges with the input polygon and is related to the *shape-complexity* of the polygon. Although the worst-case complexity of the algorithm is $O(n^2)$, for several classes of polygons it runs in linear time. The practical advantages of the algorithm are that it is extremely simple and does not require sorting or the use of bal-

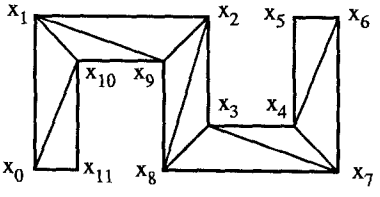

Fig. 2. Algorithm triangulate (P)

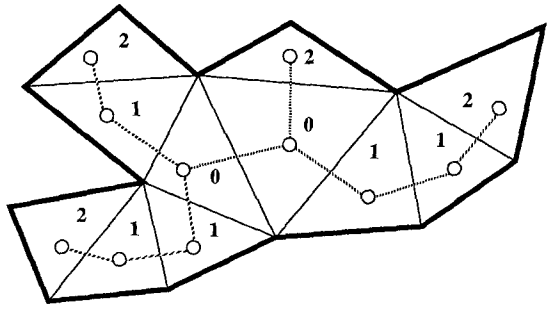

Fig. 3. A triangulation (*solid lines*) of a simple polygon P, its *dual tree* (*dotted lines*), and the integer *value* t_i associated with each triangle