A Simple Polygon Triangulation Algorithm

Piyush Kumar
Department of Mathematics
Indian Institute of Technology
Kharagpur 721302, India
piyush@acm.org

Subir Kumar Ghosh
School of Computer Science
Tata Institute of Fundamental Research
Bombay 400005, India
ghosh@tifr.res.in

Abstract

In this Paper we propose an easily implementable polygon triangulation algorithm using *Shortest Paths*. The algorithm runs in $O(nlogk)$ deterministic time with $k < \frac{n}{3}$. Our algorithm runs in linear time for a large class of polygons. The biggest advantage of the algorithm is that it uses very simple data structures and it computes the triangulation just by scanning the boundary of the polygon. We also implemented part of the algorithm that actually calculates k for a given input polygon. Experimetal evidence shows that in most cases k is very small, making the algorithm run in near linear time for most practical purposes.

1 Introduction

Polygon Triangulation is a well studied problem in Computational Geometry[21, 13, 12, 23, 11, 25, 19, 14, 20, 17, 24, 22, 2, 3]. This Paper talks about Polygon Triangulation without Steiner points[7, 6, 5] or any kind of optimization criterion[27]. In May 1990 Bernard Chazelle [2, 3] showed that a simple polygon of n vertices could be triangulated in linear time. Till 1999 there are no implementations that work in linear time due to the high constants involved in the implementation of his algorithm. The algorithm is very intricate and uses involved tools and notions such as a *planar separator theorem*, *polygon cutting theorem[4]*, *conformality[8]*, etc. Though the algorithm is asymptotically optimal, it is conceptually difficult and too complex to be considered practical. Chazelle [2] mentions that the existence of a truly simple linear time algorithm remains open question and conjectures that such an algorithm is unlikely. As an alternative researchers started looking for classes of polygons that can be triangulated in linear time. Such classes include monotone,star-shaped,edge visible, spiral,L-convex,intersection free, weakly externally visible, palm shaped, anthromorphic,LR etc[13, 26, 24, 1]. Some very notable attempts to triangulate a simple polygon based on the shape complexity appeared in the 80's and early 90's[15, 9, 24].

This study is a step towards finding out a shape dependent algorithm that eventually could

lead us to a simple practical linear time algorithm for this problem . In its current form it can triangulate a large class of polygons in linear time although the worst case still remains at an abysmal $O(nlogn)$.

We have omitted the proofs of the existence of a triangulation for an arbitrary simple polygon in this Paper. Interested readers could refer to standard textbooks[29, 28] for some very interesting discussion on the history and proofs related to this problem.

The Paper is organised as follows. We give an algorithm to decompose an arbitrary simple Polygon into Polygonal Chains in Section 2. In Section 3 we give a linear time algorithm to merge two such neighbouring chains and the overall algorithm. We also discuss what are the other different approaches which could be tried. Section 4 concludes the Paper. Work is still in progress on the Sections 3 of this Paper.

2 Decomposing a Simple Polygon

Let $SP_c(u, v)$ denote the correct Euclidean shortest path inside P from a point u to another point v. For any vertex u of P the *shortest path tree* of P rooted at u, denoted as $SPT_c(u)$, is the union of the shortest paths from u to all vertices of P. By abuse of notation, we use SP(u,v) to denote the partial incorrect Shortest paths which is only correct with restect to the clockwise polygonal chain from u to v. The same is true for the SPT(u) notation too. Let $C_1 = \{p_1, p_2, ..., p_m\}$ denote a polygonal chain in the clockwise sense such that the vertex p_{m+1} in not a leaf of $SPT(p_1, p_{m+1})$. We call the vertex p_m a *reverse turn*. Let $|C_i|$ denote the number of vertices in the chain C_i. We run the following algorithm to decompose P into k polygonal chains.

Algorithm 1 Decompose P into k-Chains

Require: P to be simple.
Ensure: $P = \{v_0, v_1, ..., v_n\}$ where P lies right of $\{v_0, v_1\}$

```
r ⇐ v_0; Mark(r)
k ⇐ 1
for u = v_1 to v_n do
  if u is not a leaf of SPT(r) then
    r ⇐Pred(u)
    k ⇐ k + 1
  end if
end for
```

Note that in the above algorithm $SPT(r)$ denotes the Shortest Path Tree constrained only by the boundary r,u and does not mean the complete shortest path tree rooted at vertex r whose notation we fixed as $SPT_c(r)$. It could be easily shown that in the worst case $k \in O(n)$. But even if $k \in O(n)$, it is not guaranteed that our algorithm is pushed to it's worst case of $O(nlogk)$. A glaring example is the worst case polygon of the shape complexity based triangulation paper by Chazelle[9]. Once we state the algorithm in its full entirety it would be trivial to prove that this case(See Figure 1) is solvable in linear time by our

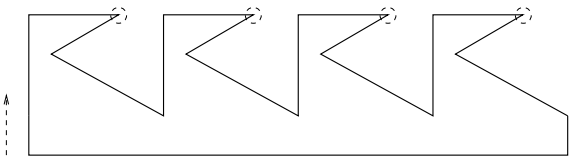

Figure 1: When this Polygon is traversed in clockwise sequence $k \in O(n)$.

algorithm. Algorithm 1 Marks k vertices of P and hence divides the polygonal chain $\{v_0, v_n\}$ into k sub-chains. Let C_i denote the sub-chain starting at the k-th Marked vertex. It also computes as a biproduct, the Shortest Path Trees of all the sub-chains. These k shortest path trees that we computed using Algorithm 1 could all be wrong since other chains might be protruding inside the shortest path trees of a chain. If $k = 1$, the entire $SPT_c(v_0)$ has been computed. In this case P can be triangulated easily once $SPT_c(v_0)$ is known.

3 $SPT(u) \bigoplus SPT(v)$ in linear time

Let u,v be two consecutive reverse turns of P in clockwise order. While decomposing the polygon we already computed $SPT(u)$ and $SPT(v)$. Let w be the next reverse turn after v. Let us denote the polygonal chain (u,v) by C_i and so the polygonal chain (v,w) becomes C_{i+1}. In this section we give a linear time algorithm to correct the SPT(u) for the polygonal chain $C_i \cup C_{i+1}$.

LEMMA 1 *Two Convex Polygonal Chains emerging from a point can at most intersect once.*

Lemma 1 is a direct consequence of the fact that two convex polygonal chains can at most intersect twice on the plane provided no three points are collinear. Thus, $SP(u, v)$ and $SP(v, w)$ can intersect at most once due to their convexity.

DEFINITION 1 (CUT) *A* Cut *(See Figure 2) is an edge that connects two vertices z, z' on $SP(u, v)$ and $SP(v, w)$ respectively such that the $SP(z, u)$ and $SP(z', w)$ do not intersect and the edge (z, z') is not intersected by any edge $\in C_i \cup C_{i+1}$.*

It is clear from the above definition that the *Cut* may not be unique. We define a *Max-Cut* as a cut where either of z, z' cannot be moved towards u,w respectively without making them invisible(See Figure 2). It is easy to show that, Given a *Cut* it is possible to find a *Max-Cut* in linear time of the number of vertices on the Shortest Path between (u,z) and (z',w) by simple walking and finding parents alternatively. If we started from v and somehow got a *Cut* in linear time then we had broken the merging problem into two parts. Let us for the time being assume that we have such an algorithm which can find out a *Cut* (z, z') in Linear Time given $SPT(u)$ and $SPT(v)$. Let us denote the *Max-Cut* by z_c, z'_c(See Figure 2). The structure we have now is what is called an *Hour-Glass*(See Figure 3). We now need to shift the root of the vertices on the chain (z', w) to z_c. Let us walk from the vertex z'_c towards z'

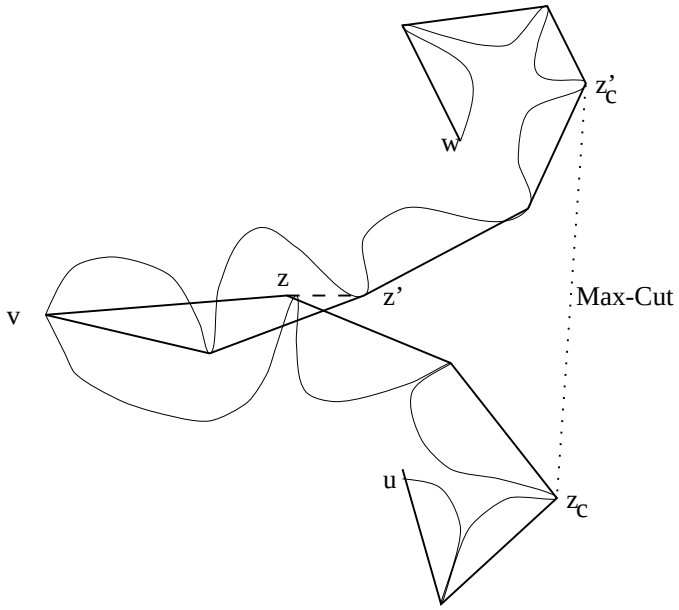

Figure 2: zz' is a *Cut*.

along the polygon. Each new edge considered has one vertex whose parent is z_c and for the other we want to compute the shortest path to z_c. Let us denote this edge by ab.(See Figure 3). Initially $b = z'_c$ and a is the neighbouring vertex of b in the anti-clockwise sense on the polygon. Note that $SP(z', a)$ already exists. Then there need to be handled three simple cases to compute the $SP(a, z_c)$ when $SP(b, z_c)$ is known.

Case I a lies on the right of the ray formed by the parent of b in $SP(b, z_c)$.

Case II Case I does not happen and $SP(a, z')$ does not intersect the parent of a on $SP(b, z_c)$.

Case III Case I and Case II do not happen.

Case I can be handled trivially by making b the parent of a. If Case I does not happen, we first walk along $SP(b, z_c)$ and find the parent of a. If this parent of a intersects with $SP(z', a)$ we are in Case III. In this case we just scan the boundary till we come to the current parent of b and do not disturb the Shortest Path Tree rooted at the parent of b.

3.1 Case Analysis: The Two Cases

In this section we give the actual meathod to compute the *Cut* in linear time. We would see how the *Hour-Glass* structure comes naturally and since we now have a meathod to resolve it, the whole merging step is going to be pretty simple. By Lemma 1 there can only be two cases while walking on the $SP(v, w)$, either $SP(v, w)$ does not intersect the polygonal boundary(See Figure 4) or it does(See Figure 5). Triangulation by boundary

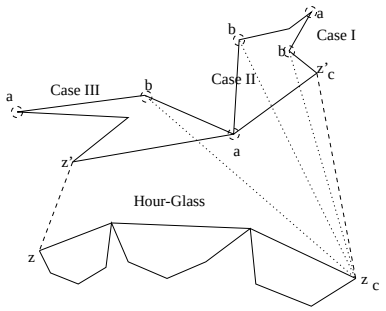

Figure 3: An Example *Hour-Glass*

scanning was first used by Bhattacharya and Ghosh[1] for recognising and triangulating LR-Visibility polygons in linear time. We are still working on the merging step in both theory and implementation. The Merging step is yet to be written totally.

3.1.1 Solving Merging For Case I

(See Figure 4) We know that the $SP(v, w)$ does not intersect any polygonal edge from u to v. We just walk on the $SP(v, w)$ starting from v and keep deleting edges from $SPT(u)$ which intersect it. The termination criterion for this walk is when the extension of this edge does not hit the polygonal boundary. This is where we join the starting vertex of the edge on which we are walking to the parent of the last edge we deleted in $SPT(u)$. Note that we have now a cut which we can solve using our previous discussion and the orphans which were created due to the deletion of edges in $SPT(u)$ can easily be taken care of by searching for parents on the $SP(v, w)$ starting from v. We will write more about this case and the subsequent case as the work progresses.

3.1.2 Solving Merging For Case II

(See Figure 5) The basic idea of how to compute the $SPT(u)$ from the given information is to again walk down from v on the $SP(v, w)$ discovering $Cuts$ as they come, and then resolving the orphans. Ultimately we would get a Cut that gives us the $Hour - Glass$ that we discussed. This Cut is nothing but the indication that we have walked past the second intersection of the Shortest Paths or we are done. This Case needs some further Case Analysis that remains to be done.

The Example Figures were created by our $C++$ implementation of the algorithm that calculates k for an input polygon. The circles at the top show the reverse turns in the clockwise sense. The program also outputs the Partial Shortest Path Trees starting from the lowest vertex of the polygon and moving clockwise.(See Figure 4 ,5). We are still working on the implementation of the merging step.

THEOREM 1 *The shortest path tree from a vertex of a simple polygon can be computed in* $O(nlogk)$ *time.*

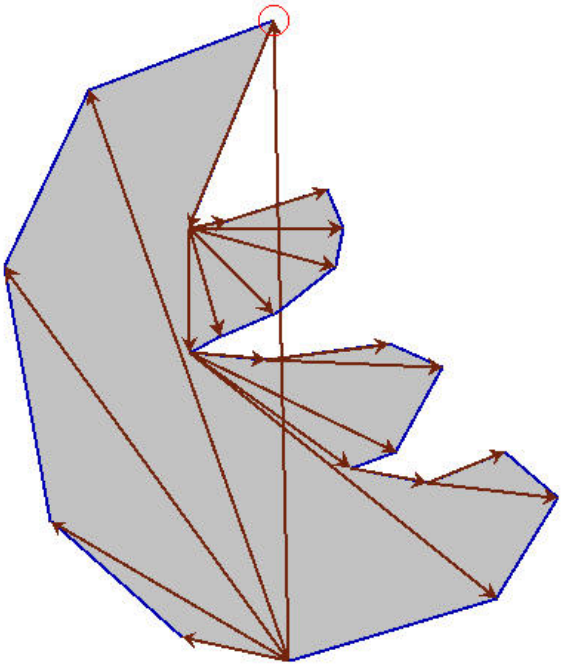

Figure 4: Case I

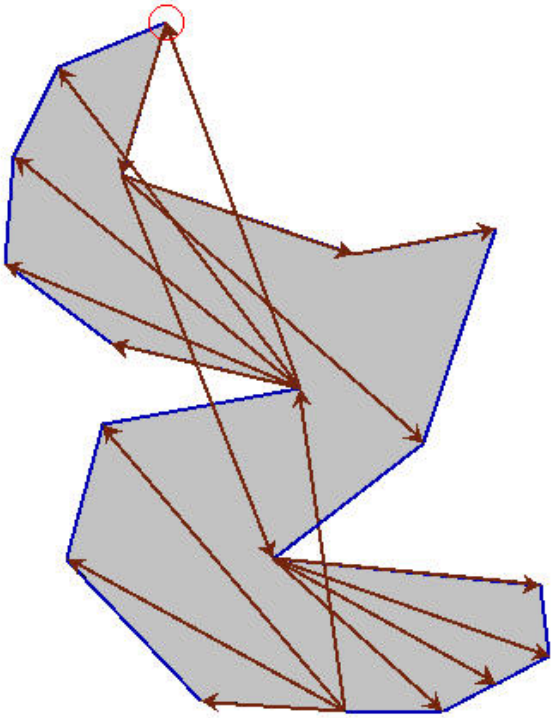

Figure 5: Case II

4 Concluding Remarks

Since the merging step can be done in linear time of the number of vertices in the chains, we can merge these chains in a binary tree structured way and can thus compute the Shortest Path Tree from a vertex of the polygon in $O(nlogk)$ time. Now we could compute the triangulation in linear time from the Shortest Path Tree of the Polygon. Hence we can compute the triangulation of the given polygon in $O(nlogk)$ time.

There are some very interesting problems that remain unsolved in this Paper. Can this algorithm be modified to run in linear time? Can randomization be used to lower the time complexity and at the same time make the algorithm simpler? We could use the initial steps of the algorithm in [17] to make k smaller, (Actually $k < \frac{\sqrt{n}}{3}$) but this would make the algorithm more complex and difficult to implement without reducing the time complexity. Is there a way of using horizontal visibility maps to make the algorithm faster?

References

[1] Binay K. Bhattacharya and S.K.Ghosh Characterizing LR-Visibility Polygons and Related Problems. In *Proc. of 10th Canadian Conference on Computational Geometry* 1998.

[2] B. Chazelle. Triangulating a simple polygon in linear time. Technical Report No. CS-TR-264-90, Department of Computer Science, Princeton University, 1990.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485-529, 1991.

[4] B.Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339-349, 1982.

[5] M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 222-231, 1992.

[6] M. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 342-350, 1991.

[7] B. S. Baker, E. Grosse, and C. S. Rafferty. Nonobtuse triangulation of polygons. *Discrete Comput. Geom.*, 3:147-168, 1988.

[8] B. Chazelle. Efficient polygon triangulation. Preprint. Probably unpublished - precursor to [Cha90b], 1990.

[9] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Trans. on Graph.*, 3(2):135-152, 1984.

[10] M. T. Dickerson, R. L. S. Drysdale, S. A. McElfresh, and E. Welzl. Fast greedy triangulation algorithms. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 211-220, 1994.

[11] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. on Graph.*, 3(2):153-174, 1984.

[12] S. K. Ghosh. A linear time algorithm for decomposing a monotone polygon into star-shaped polygons. In *Proc. 3rd Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, pages 505-519, India, 1983.

[13] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7:175-179, 1978.

[14] S. Goldman. A space efficient greedy triangulation algorithm. *Inform. Process. Lett.*, 31(4):191-196, 1989.

[15] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory*, volume 158 of LNCS, pages 207-218. Springer-Verlag, 1983.

[16] K. Hoffman, K. Mehlhorn, P. Rosenthiel, and R. Tarjan. Sorting jordan sequence in linear time usuing level-link search trees. *Inform. and Control*, 68:170-184, 1986.

[17] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in O(n log log n) time with simple data structures. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 34-43, 1990.

[18] C. Levcopoulos. An $\Omega(pn)$ lower bound for the nonoptimality of the greedy triangulation. *Inform. Process. Lett.*, 25:247-251, 1987.

[19] Andrzej Lingas. Greedy triangulation can be efficiently implemented in the average case. In *Graph-Theoretic Concepts in Computer Science. Proceedings.*, 1989.

[20] C. Levcopoulos and A. Lingas. Fast algorithms for greedy triangulation. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of LNCS, pages 238-250. Springer-Verlag, 1990.

[21] B. A. Lewis and J. S. Robinson. Triangulation of planar regions with applications. *Comput. J.*, 21:324-332, 1978.

[22] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51-64, 1991.

[23] G. T. Toussaint. A new linear algorithm for triangulating monotone polygons. Technical Report SOCS 83.9, McGill University, 1983.

[24] G. T. Toussaint. An output-sensitive polygon triangulation algorithm. In *Proc. 8th Internat. Conf. on Comput. Graphics*, pages 443-446, 1990.

[25] R. E. Tarjan and C. J. Van Wyk. An O(n log log n)-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143-178, 1988.

[26] Schoone,A.A. and van Leeuwen,J. Triangulating a star-shaped polygon. Tech.Report,RUV-CS-80-3, University of Utrecht, April 1980.

[27] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest Introduction to Algorithms. MIT Press, 1990.

[28] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf Computational Geometry, Algorithms and Applications SpringerVerlag, 1997.

[29] Joseph O'Rourke Computational Geometry in C. Cambridge University Press, second edition, 1998