A Simple Algorithm for Computing a Spiral Polygonization of a Finite Planar Set

M. Ahsan, S. Cheruvatur, M. Gamboni, A. Garg, O. Grishin, S. Hashimoto, J. Jermsurawong, G. T. Toussaint, and L. Zhang
Faculty of Science, New York University Abu Dhabi, United Arab Emirates

I. Introduction

A polygonization of a set of points S in the plane is a closed polygonal chain, the vertices of which are those points in S. A spiral polygon consists of the concatenation of exactly one chain of consecutive convex vertices and one (possibly empty) chain of consecutive concave vertices. Iwerks & Mitchell showed that a spiral polygonization can be computed in $O(n \log n)$ time and $O(n)$ space using the convex hull layers algorithm of Chazelle [1], [2], or the data structure of Hershberger & Suri [3], [4]. Here, a simpler algorithm is proposed for computing spiral polygonizations that generates pseudo-nested convex hulls that pair-wise share a unique edge. Using simple data structures, the algorithm runs in $O(n \log n + nh)$ time, where h is the number of nested convex hulls ($O(n^2)$ in the worst case) and $O(n)$ space. With the dynamic data structures of Hershberger & Suri the algorithm runs in $O(n \log n)$ time and $O(n)$ space. Our algorithm also admits a simpler proof of correctness.

II. The Algorithm

At a high level, our algorithm successively calculates pseudo-nested layers of convex hulls, with each convex hull sharing one unique edge with the previous convex hull, and at the end joins them using the shared edges to create a spiral polygon. First consider doubly linked lists: X (unsorted points), A (convex points), and B (concave points), with A and B starting as empty lists and all input points being in X in any order. The use of doubly linked lists allows us to move points between the lists in $O(1)$ time without disturbing the overall order of the points. We then sort the points in X, with any $O(n \log n)$ sorting algorithm (such as mergesort or heapsort; we used the latter in our implementation). We create and set a flag Q to true. While X is not empty, we do successive Graham scans ($O(n)$ time complexity). Each Graham scan gives us the convex hull of the points remaining in X. Note that the Graham scan should return the linked list representation of the convex hull, without the end nodes joined. Thus, there is a concept of the first and last nodes in the convex hull. If the size of the list returned by the Graham scan is less than that of X, meaning some points are still not in the convex hull, then we mark the last two points of the convex hull as having to stay in X, call them x_1 and x_2. These points link the current convex hull layer to the next layer. Then for all other points detected by the Graham scan, we remove these from X in-place and place them in a new linked list L in $O(k)$ time complexity where k is the size of convex hull. Unless we are in the first Graham step, the x_1 and x_2 of the last scan must necessarily be part of this convex hull. Rotate L such that x_1 is the tail of the list, and x_2 is the head, in $O(1)$ time. If we were in the first Graham scan, ignore this step. Then, if Q is true, we concatenate them with list A, otherwise we concatenate them with list B (w.l.o.g., and similarly for the rest of the analysis). Afterwards, we invert the value of Q, and repeat until no points remain in X. Finally, we have a fully convex list, and concave list, that are both spirals, so we simply reverse the convex list, in $O(n)$ time complexity, and concatenate it with the concave list. Now we have a spiral polygon represented as a double linked list.

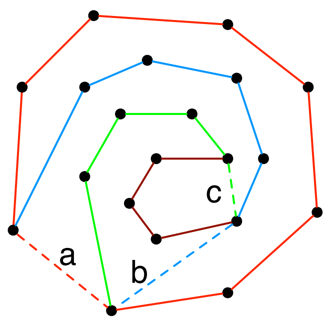

Fig. 1. An example illustrating the pseudo-nested convex hulls at different stages of the algorithm.

A. *Proof of Correctness*

In a line through the center of a spiral polygon, the polygon edges it intersects alternate between being on the concave and convex chains. Because the algorithm places two points back into the unpolygonized set X if there is at least one point in there, and three points are sufficient to form a convex hull, our algorithm incorporates all points and successfully finds successive convex hulls. To prove that the convex hulls are correctly linked consider the two points returned into X at the end of each Graham step, call them x_1 and x_2. In the next convex hull, x_2 is the head, and x_1 is the tail, by specification in the algorithm. Consider the new x_1 and x_2, call them x'_1 and x'_2. We can see that $x_1 == x'_2$, and thus x_2 is in the chain of type C with the first convex hull, whereas, $x_1 == x'_2$ is of chain type C', containing the second convex hull. The proof follows by induction.

B. *Time and Space Complexity*

The algorithm presented above has a worst case time complexity of $O(n^2)$. This complexity can be reduced to $O(n \log n)$ using the modification described below, but experiments with various random distributions of points show that the above algorithm is much faster. It is also significantly less complex to implement.

1) Analysis: The Graham scan takes $O(n \log n)$ time. We use a doubly linked list to sort of all the points once at the beginning, and we simply move the pointers of the convex hull points detected by the Graham scan in order to maintain the sorted property of all non-spiralized points for the duration of the algorithm. Thus, the Graham scan takes only $O(n)$ time at each convex hull level. In the worst case, there are n convex hulls that need to be computed, since a convex hull can consist of as few as three points, and at each step, two points from the convex hull are put back into the set of spiralized points. Thus, in the worst case, there are n $O(k)$ Graham scans (where k is the number of remaining points), resulting in a worst case complexity of $O(n^2)$. Note that in random distributions where the average number of points in a convex hull is much greater than 2, then the time complexity is $O(n \log n + nh)$ where h is the number of pseudo-nested convex hulls.

2) Achieving $O(n \log n)$ Time Complexity: We can substitute the Graham scan for the Hershberger-Suri data structure. This data structure can store the upper (w.l.o.g. lower) convex hull of a subset P of points S as points are deleted online from P. Building the convex hull of S takes $O(n \log n)$ time and $O(n)$ space, and each deletion is $O(\log n)$ worst-case time.

III. Experimental Results

As seen in Fig. 2, the Graham Scan performs better in practice than the Hershberger-Suri data structure. Apparently, no implementations of the Hershberger-Suri data structure have been previously empirically tested.

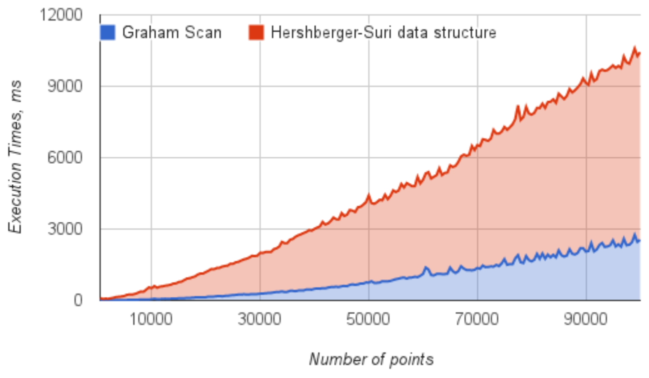

Fig. 2. Execution times for the two algorithms.

IV. Conclusion

The new algorithm presented is simpler than the one proposed by Iwerks & Mitchell, and admits a simpler proof of correctness. Two methods for computing the convex hulls needed in the algorithm were tested, the Graham scan and the Hershberger-Suri deletion-only data structure. While the complexity using the Graham Scan is $O(n \log n + nh)$ compared to $O(n \log n)$ for the Hershberger-Suri data structure, experimental results show that the Graham scan is faster for random independent uniformly distributed points in the unit square.

Acknowledgment

We thank Joe Mitchell and Justin Iwerks for making the Java code of their polygonization algorithm available. This research was supported by the Provost's Office of New York University Abu Dhabi, U.A.E.

Code Available at: https://github.com/lingz/spiral-polygonization

References

[1] Iwerks, J., Mitchell, J. S. B. (2011). Spiral Serpentine Polygonization of a Planar Point Set. In: Proceedings of XIV Spanish Meeting on Computational Geometry, pp. 181184.

[2] Chazelle, B. (1985). On the Convex Layers of a Planar Set. IEEE Transactions on Information Theory, 31, 509517.

[3] Iwerks, J., Mitchell, J. S. B. (2012). Spiral Serpentine Polygonization of a Planar Point Set. In Encuentros de Geometria Computacional, (Hurtado Festschrift), A. Mrquez et al. (Eds.): LNCS 7579, pp. 146-154.

[4] Hershberger, J. & Suri, S. (1991). Finding Tailored Partitions. Journal of Algorithms, 12(3), 431-463.