Optimal Area Polygonalization

Dana Chaillard, Ilan Coulon

***Abstract*—The problem of finding an optimal area simple polygonalization for a given set of points in the plane is NP-hard. Due to the complexity of the problem we aim at finding an algorithm to obtain approximate solutions.**

In this project, we tried a strategy in order to optimize the polygonalization area and tried to solve the associated problems. We ended up noticing our algorithm does not give correct solutions in every case but we were able to determine in which context it does not and gave some leads to how correct and improve it.

I. Introduction

While the classical geometric Travelling Salesman Problem is to find a (simple) polygon with a given set of vertices that has shortest perimeter, it is natural to look for a simple polygon with a given set of vertices that minimizes another basic geometric measure: the enclosed area.

The point of this project is to compute a simple polygonalization of a set of points that has maximum or minimum area among all possible polygonalizations. This problem is known for being NP-hard [1]. It may arise in the context of pattern recognition, image reconstruction, and clustering.

We are given a set of points S with integer even coordinates, indexed between 0 and $n-1$. The goal is then to return an array of n integers, representing the polygon.

II. Preliminary work

In order to perform the optimal polygonalizations, we need to write some preliminary algorithms to check our solutions. We will also be able to use them in our final algorithm.

A. Polygon validation

We need to be able to check whether a polygon (represented as an array of integer) is valid. First, we check that it is a valid candidate: the array must be of size n, and there cannot be any duplicates (i.e. it must be a permutation of $[\![0, n-1]\!]$).

Then, we need to check if the polygon intersects itself. Some efficient solutions exist, such as the Bentley-Ottman algorithm [2] which detects the intersections in time $O((n+k)logn)$ (where k is the number of intersections). Because this was not such a limiting factor in our case, we implemented the very naive algorithm that tests every pair of segments, which takes $O(n^2)$.

B. Area computing

For the total area, we use a very simple formula that computes it in linear time [4]. We assume that the points are counted in a specific order around the perimeter. For each segment, we add the area to the left of it all the way to the y axis using $(X_{i+1} + X_i) \times (Y_{i+1} - Y_i)\ \forall i$. This area is represented Fig. 1.

For a segment going upwards, with this formula, the grey area is counted negatively which in the end leaves us with only the interior area.

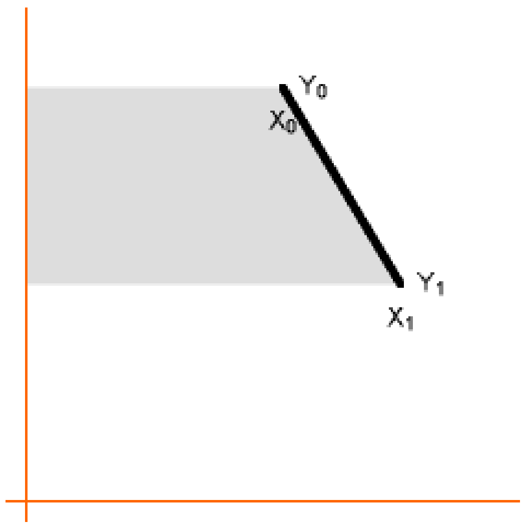

Fig. 1. Area computed by the algorithm for each segment

The main benefit of this method is that it does not involve any angle computation or triangulation. With only addition and multiplication, we will not have any rounding errors and it is therefore completely compatible with our integer coordinates and will not cause approximations.

C. Convex Hull computation

To compute the convex hull, we used the Andrew algorithm as seen during TD1 [3].

III. Optimal Polygonalization

As we have a pretty efficient algorithm to compute the convex hull and we need the convex hull to know the score anyway, we decided to start from the convex hull to find our polygonalization.

The basic idea is to iteratively ”dig” triangles in our polygon. At each step, we have to make sure our polygon does not intersect itself and every remaining point is still in the polygon. To try to have an optimal area in the end, we take the smallest/biggest available candidate triangle.

If the point is a potential candidate (fullfilling all the conditions), we then insert said point in the polygon so as to exclude the triangle it makes with the chosen edge from the interior of the polygon as shown Fig. 2

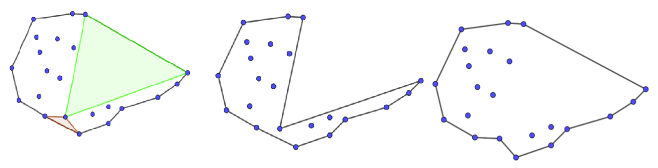

Fig. 2. First iteration of the algorithm: minimal area on the second image, maximal area on the third

A. *First naive correction*

In order to know if the new triangle would form a valid polygon, we, in a modular way of thinking, used the algorithm that states whether a polygon is valid at each step. The algorithm worked quite well for 100 points, but the time it took to compute the polygon with 1000 points was not reasonable (a few days).

A first improvement was made by assuming that at each step, the polygon is valid. Then, it is possible to know if the new polygon intersects itself by checking only with the 2 new segments.

This reduced the computation of 1000 points to approximately 2 minutes for minimal area and 15 seconds for the maximal one. Nonetheless, we noticed an issue after this computation. The returned polygon was invalid for the minimal area.

B. *A more fundamental error*

After some investigation, we found that it was due to a lack of potential candidate points at the end of the polygon building. Looking precisely in the data stored during the polygonalization, we found that it was caused by a situation equivalent to the situation depicted Fig. 3.

In this figure, we can see that no edges are entirely visible from the central point P. Therefore, at this step, our algorithm cannot keep ”digging” the polygon.

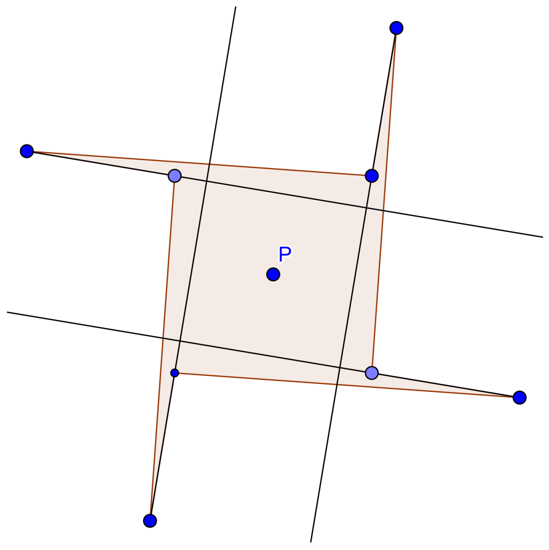

Fig. 3. A figure making our algorithm not able to find a solution

A solution would be to check at every step that every point can still see at least one edge. If it is not the case, then we would have to reverse our last action and choose another triangle (and maybe in some cases we would have to climb higher in the tree).

The algorithm becomes instantly way more complex, and we could not go further on this path since we noticed the issue at the last moment.

C. *Results*

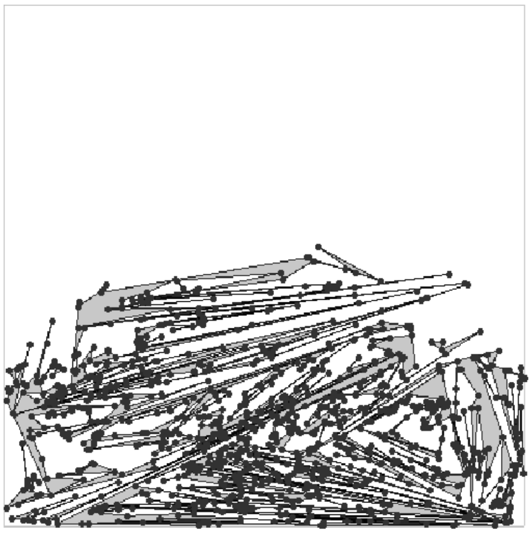

Fig. 4. An approximate solution for the minimal area polygonalization with the paris-0001000 point set

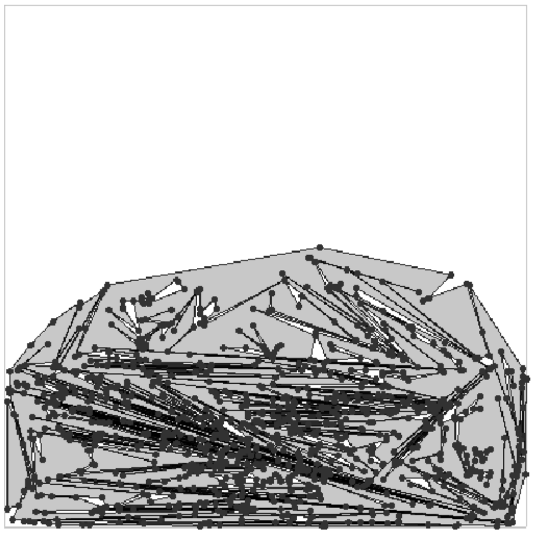

Fig. 5. An approximate solution for the maximal area polygonalization with the paris-0001000 point set

First of all, we can clearly see that the algorithm yields a very different result for max and min area which is reassuring cf Fig. 4, Fig. 5 To our knowledge there do not exist public benchmarks to compare our results to. If we compare the score of our first very naive implementation of the algorithm to the current one, we see they are quite similar.

TABLE I
SCORES AND BENCHMARKS WITH OUR BROKEN ALGORITHM.

Dataset	ScoreMax	ScoreMin	TimeMax (s)	TimeMin (s)
uniform-0000010-1	0.84	0.58	0.33e-3	1.47e-3
uniform-0000100-1	0.84	0.29	0.05	0.33
euro-night-0000100	0.89	0.30	0.06	0.13
uniform-0001000-1	0.43	invalid	14	542
paris-0001000	0.83	0.25	11	697

With the scores of different datasets we can see that they are mostly always the same even when the distribution or the number of points is different. There is, of course, the exception of *uniform-0001000-1* which seems to be in a configuration that doesn't work with our algorithm.

We also notice that for every dataset, computing the max area polygonalization takes less time than the min area.

IV. Conclusion and Future Work

Our algorithm was based on a really intuitive way of thinking. It did not work for the general case, but it made us see how and why this was not just a bug caused by an indexing error but a more fundamental and geometric error.

Even if this algorithm did not produce really good results, this intuitive way of thinking may be a good way to select the triangles.

If we had more time, in some kind of *combinatorial optimization* pattern, we could have built a progression tree in order to climb it up when encountering the situation depicted Fig. 3.

In order to know if the situation is like Fig. 3 more easily, and in a matter of performance optimization, we could also have stored the visible edges from each point in a *dynamic programming* pattern. Knowing how to store this information and how to update it would also have been a challenge but it seems to be a good path in order to make this algorithm correct and relatively efficient.

We could also have chosen to start by a triangle and expand it iteratively instead of reducing the convex hull. This could have avoided the situation causing our algorithm to stop.

References

[1] S. P. Fekete, On Simple Polygonalizations with Optimal Area, Discrete and Computational Geometry 23:73-110 (2000)

[2] J. Bentley et T. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers*, 1979 (DOI 10.1109/TC.1979.1675432)

[3] As seen during TD1 of *INF562 Computational Geometry: from Theory to Applications*, 2020 http://www.enseignement.polytechnique.fr/informatique/INF562/TD/TD1/INF562-TD1-1.html

[4] Darel Rex Finley, Ultra-Easy Polygon Area Algorithm With C Code Sample http://alienryderflex.com/polygon_area/