INFO-F420: Preliminary project report
Compatible triangulations of polygons

Laurie Van Bogaert

2 November 2020

1 Introduction

For the course of computational geometry, students are asked to do a project. The present document is the preliminary version of the project report. The subject that I choose is: **Compatible triangulations of polygons** with fixed correspondence between the points.

2 Why this problem ?

The thematic for the projects this year is about the open problems in computational geometry. One of them is about compatible triangulations. On the website of the open project the problem is stated as determining if two sets of points of the same size and in the general position possess compatible triangulations[1]. The project here is about a variant of this problem where the triangulation is done on simple polygons in place of sets of points.

Finding the compatible triangulations has practical application in 2D animations and shape morphing [2]. Therefore, this subject can be used to give entertaining visual results.

3 Implementation

The project is divided in 3 parts.

3.1 User Input

First, the user have to create a polygon by selecting new points with his mouse. Then he have to press the next button to close the polygon. At this points, the program check if there is any edge crossing. Also, the rest of the code suppose that the polygon follow the left-turn convention in computational geometry. Therefore , if the program detect it to be right turn it will make it left turn by inverting the points and the arrays. Next, to create the second polygons, the points of the first polygons are moved. Once again the program check that there is no edge-crossing. Since the second polygon is build by moving the point of the first polygon, it gives the correspondence between the points of the two polygons.

3.2 Algorithm for compatible triangulations

First, the number of possible triangulations for a convex polygon of size n is equal to the (n-2)th Catalan number: $C_{n-2} = \frac{(2n-4)!}{(n-1)!(n-2)!}, \qquad n \geq 3.$ [3]. For non-convex simple polygons, the number of combinations is smaller, as there are vertices that are not visible by others.

The polygons may be triangulated without using Steiner point by using the following brute force algorithm (see pseudo-code algorithm 1 in the next page). This recursive algorithm will research compatible triangulations between the first polygon and the second polygon.

The idea here is to explore the tree of possible enumerations of triangles to form a triangulation. It is know that a polygon of size n is composed of exactly $n-2$ triangles [4]. So there is $(n-2)!$ different enumerations of triangles. The number $(n-2)!$ is actually far above the number of possible triangulations for a convex polygon of size n. Is this due to the fact that a triangulation is a set of

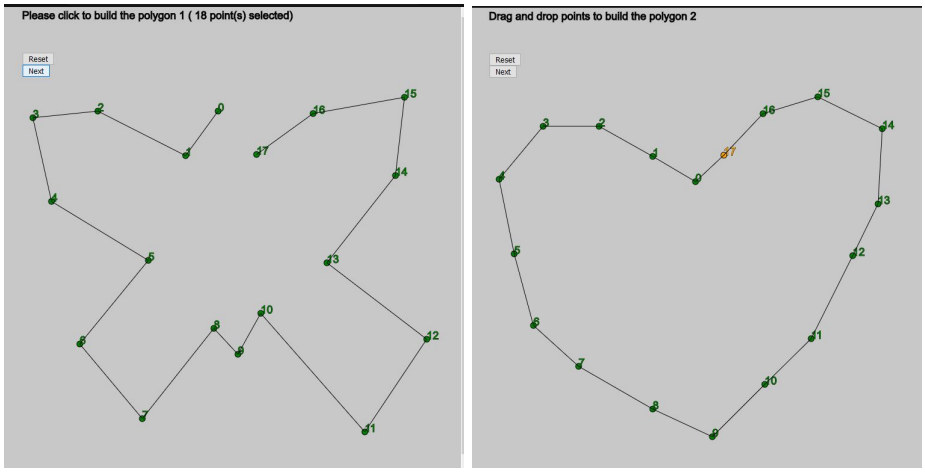

Figure 1: An example of input given by the user. With the first polygon at the left and the second one at the right

triangles, so we didn't care about order unlike a enumeration. Therefore, in the worst case scenario where we have to explore the entire tree the complexity is quite bad and follow a factorial factor.

In the algorithm 1, a triangle *abc* and is considered as valid if it:

- is left-turn in the first polygon. The polygons are supposed to be left turn so a triangle that contains an edge of the polygon is left turn if the triangle is in the polygon.
- is left-turn in the second polygon
- does not contains a vertex of the first polygon.
- does not contains a vertex of the second polygon.
- does not cross an edge of the first polygon or the valid triangles already computed.
- does not cross an edge of the second polygon or the valid triangles already computed.

Algorithm 1 A brute force algorithm

```
1:  procedure FINDCOMPATIBLETRIANGULATIONS(triangles , Polygon)
2:      Edge (a,b) ← Polygon.getEdge(0)
3:      for Vertex c in Polygon with c ≠ a and c ≠ b do
4:          if Triangle abc is valid for Polygon1 and Polygon2 then
5:              NewTriangles ← copy(Triangles)
6:              add Triangle abc to NewTriangles
7:              NewPolygon ← copy(Polygon)
8:              for Edge ij in triangle abc do
9:                  if Edge ij is an edge of Polygon then
10:                     remove ij from NewPolygon
11:                 else
12:                     add ij to NewPolygon
13:             if NewPolygon still have edges then
14:                 NewTriangles ← findCompatibleTriangulations(NewTriangles,NewPolygon)
15:                 if NewTriangles ≠ null then
16:                     return NewTriangles
17:             else
18:                 return NewTriangles
19:     return null
```

However, it would not work for every pair of polygon as it is known that not every pair of polygons possess compatible triangulations [5]. Nonetheless, it should deal with the simplest cases.

3.3 Showing result

Finally, the compatible traingulation is showed on the screen with red lines and a morphing animation is created between the two polygons. An example of it can been seen at figure 2 where the first and last correspond to the first and the second polygon and the other pictures are intermediary image taken from the animation.

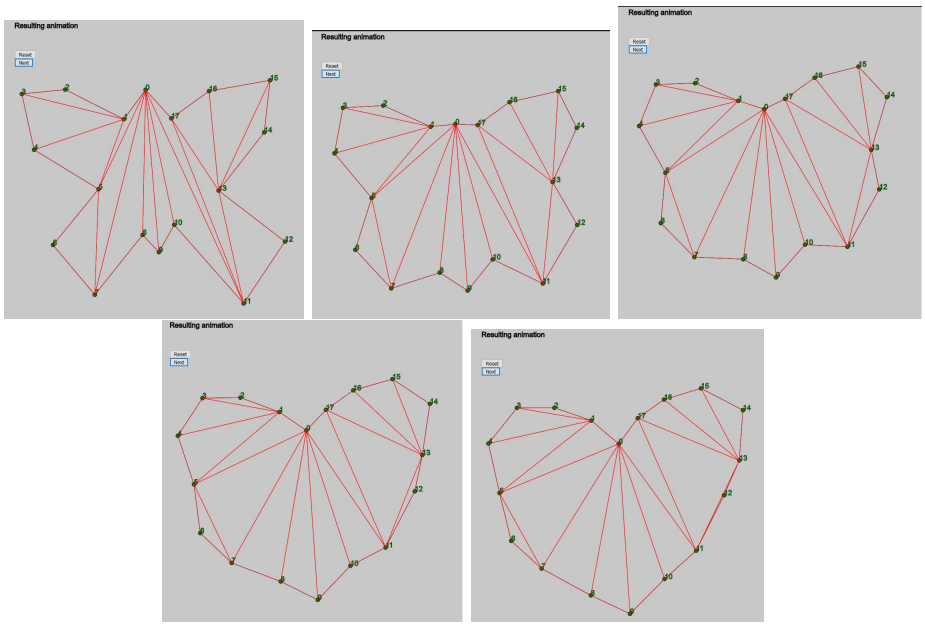

Figure 2: Animation generated with the program that show that the triangulation is compatible

4 TODO for the final project submission

Some improvements can be done before the final submission of the projects:

- Some minor bugs can be removed. In some special cases the triangulation is not compatible and the condition for a triangles to be valid need to be a little bit modified in the code.
- Compatible triangulations could be search with other points associations than the one build by the user.
- To show clearer correspondence between the two triangulations, the triangles can be coloured like in the paper [2].

5 Discussion

The program seems to work well on simple cased but some bugs can still be found in it. Also the algorithm is far from optimised. For example, it will be more effective to do use for example a plane-sweep algorithm to check for intersection. However, the goal here is to experiment with the problematic so an optimised algorithm is not needed.

References

[1] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke, "Problem 38: Compatible triangulations," 2017.

[2] Z. Liu, L. Zhou, H. Leung, and H. P. Shum, "High-quality compatible triangulations and their application in interactive animation," *Computers & Graphics*, vol. 76, pp. 60 – 72, 2018.

[3] P. S. Stanimirović, P. V. Krtolica, M. H. Saračević, and S. H. Mašović, "Decomposition of catalan numbers and convex polygon triangulations.," *International Journal of Computer Mathematics*, vol. 91, no. 6, pp. 1315 – 1328, 2014.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Polygon triangulation," in *Computational Geometry: Algorithms and Applications*, pp. 45–61, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[5] B. Aronov, R. Seidel, and D. Souvaine, "On compatible triangulations of simple polygons," *Computational Geometry*, vol. 3, no. 1, pp. 27 – 35, 1993.