Computational Geometry Project Proposal
INFO-F420

# Compatible Point Set Triangulations

| | |
|---|---|
| Muthi Dorel-Adrian | 000 427 829 |
| Perret Romain | 000 427 613 |

Université Libre de Bruxelles
MA-INFO, Academic Year 2020 - 2021

15 December 2020

## 1 Introduction

The *triangulation* in the case of a point set is the process of connecting as many points from this set as possible such that the resulting edges don't intersect each other. We thus end up with a set of triangles, contained in the convex-hull of the set. A *compatible triangulation* corresponds to a triangulation of a point set that has the same combinatorial structure as another one, where the points delimiting their triangles can be linked by a bijection, implying that their vertices represent two isomorphic sets. Given two sets of n points $S_1$ and $S_2$, a triangulation $T_1$ of $S_1$ and a triangulation $T_2$ of $S_2$, are considered as compatible triangulations if there is a bijection $\phi$ such that $ijk$ is a triangle of $T_1$ with no points of $S_1$ inside iff $\phi(i)\phi(j)\phi(k)$ is a triangle of $T_2$ with no points of $S_2$ inside. However, this relationship is not ensured for every two sets of two dimensional points (or planar points). In particular, determining compatible triangulations for two such sets in general position, where no three points are aligned, and for which the number of extreme points is the same, remains an *open problem* [8].

The problem is considered to be true for some [1], while also being considered false for points which are not in general position. Furthermore, compatible triangulations do not always exist if the bijection between the points is given and fixed [9]. When the bijection is not given, the conjecture is proven only for point sets with at most three points interior to the hull [1]. Nonetheless, compatible triangulations are always possible if considering the addition of a bounded linear number of interior points called *Steiner points*. A list of articles related to the topic: [3, 7, 6].

## 2 Implementation

In order to expose the topic, we will aim at letting the user place points in an arbitrary manner for two separate figures, which will then be used to build planar graphs contained in the respective convex hull of their point set. The user will then be able to connect points in these sets to create polygons (simple at first), before querying the application to detect if these have compatible triangulations. Because finding compatible triangulations requires the two point sets to exhibit the same number of points, and similarly for the number of extreme points defining the convex hull, we will only allow the user to place points for the second set once having finished the first one. This way, the user will be limited and will only be able to place the required number of points in the second figure. Once two such point sets will be drawn, we will then compute the triangulations of both and find if bijective mappings can be made between them. These mappings could then be highlighted graphically by the means of different colors.

## 3 Input of point sets

The webpage contains two P5 canvas, one for each point set. The user must first insert a point set in the left canvas and validate it by clicking on a validate button. If the points are in general position, the convex hull of this first point set is computed using the Graham Scan algorithm and the second canvas is unlocked. The user is now allowed to insert the exact same number of points as in the left canvas, also requiring an, identical number of convex hull points on validation. Again, the point set not being in general position or not containing the right number of points will generate an error message on validation and require the user to redraw a point set. Once the both point sets were successfully validated, the search of a compatible triangulation between the two point sets can be launched.

## 4 Enumerating triangulations of a point set

We can enumerate by brute force the triangulations of a point set. The first step is to search all the inner edges of the point set convex hull found at the input of the point sets. Second step is to search the different combinations of those inner edges. We compute the edges number of any point set triangulation using the formula of the theorem 9.1 from *Computational Geometry : Algorithms and Applications 3rd* [4]. Let P be a point set with n points and k by the points number of the convex hull boundary. Then for any triangulation of P the formula for the edges number is: 3n-3-k and the formula for the number of triangles (faces) is: 2n-2-k. We implemented also the last formula to make verifications, if needed. We subtract from the edges number of the triangulation, the edges number of the convex hull in order to find the triangulation inner edges number. This number is used to find all the combinations of the inner edges with the size of that number. The third step is to reject all the combinations where the edges are intersecting. As we used the edges number of the theorem, we assume that the combinations without intersections are valid triangulations as their edges number correspond to the one specified by the theorem. The fourth step consists in the extraction of the triangles from the combination of edges and the convex hull edges. We browse all the 3-tuples of the triangulation edges and we save the 3-tuple in a list if it is forming a triangle. Three edges are forming a triangle if each pair of edges has a common vertex and if each common vertex is different from the other. This last step is needed in order to search and display in colors a compatible triangulation between the two given points set.

We first tried to implement a DCEL (doubly connected edge list) data structure to store each inner edge combination, then allowing us to browse the triangles (faces) of the triangulation. However, this would only make the triangle lookup faster but would not improve the bottleneck part of our code which dominates the overall complexity. Indeed, since we use a brute force algorithm, the computations of the inner edge combinations is the most intensive section of our code. this is why we gave up on the idea of using the cumbersome DCEL in favor to simple edge lists.

We also considered taking advantage of the algorithm described in [2] in order to efficiently enumerate a point set's triangulations. However, the implementation appeared to be quite complicated, first requiring us to establish a lexicographicalorder for the edges. Second, we would have to build and manage multiple data-structures representing the variants of a lexicographically maximal triangulation to which flips would be applied. More specifically, a DCEL, 4 trees (3 balanced search trees and 1 static search tree) and arrays of lexicographically ordered edges for the triangulations. Furthermore, this would require us to implement a Delaunay triangulation algorithm or use an external library supporting the enumeration of different triangulations. The positive side would have been that our application could support much higher point set sizes. The compatible triangulation lookup however would have been very similar to what we do now: selecting one triangulation in the left canvas's triangulation tree and then iterating on the right canvas's tree until we find a compatible triangulation. In [5], a parallel algorithm is also presented for enumerating triangulations, but its implementation would have required us to make extensive use of message passing techniques which would make our code far too complex for such as project.

## 5 Find a compatible triangulation

In article [1], some properties related to point set are described in order to ensure the existence of compatible triangulations for a given point set. However, these do not provide ways to generate such triangulations and could only allow us to detect the absence of such triangulations. Because the other methods mentioned previously were too complicated for such a development period, we decided to find a simpler working brute force method to enumerate triangulations and detect if two of them were compatible. Nonetheless, this method was too impractical for point sets with more than 8 points. For this reason, we tried to amortise this complexity, as explained higher in this document, by creating generator functions to get one triangulation at a time on demand instead of generating them all at once.

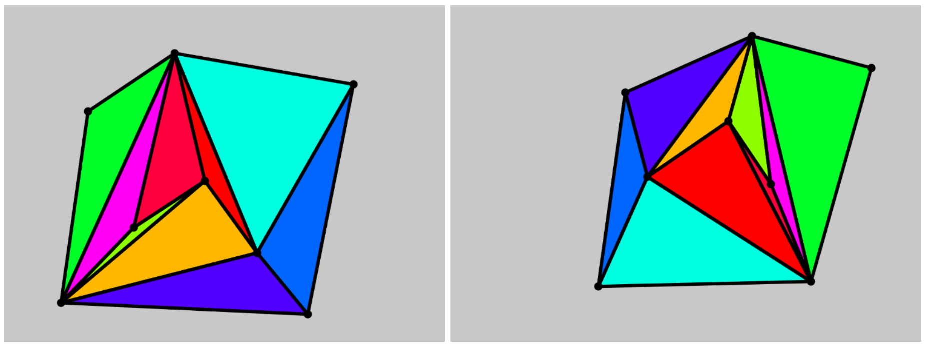

Figure 1: Example of compatible triangulation

To make that possible, we decided to assign unique identifiers to the points of each point set. A point from the first point set is in a valid bijection with a point from the second point set if their ids are the same. We keep the ids of the first point set fixed and we permute the ids of the second point set until finding a valid bijection between the points sets. A valid bijection between the two point sets is when the points of each triangle from the first point set are in bijection with the points of a triangle from the second point set, in addition the adjacent triangles of each triangle from the first point set are in a valid bijection with the adjacent triangles of its valid bijected triangle from the second point set. Therefore, two triangulations are compatible if they have a valid bijection between their points, triangles and bijected triangles neighborhood. This implies that the edges of the triangulation are also in a valid bijection. Once a bijection has been established, we can then display the two point sets by showing the mapped triangles in color, as displayed in figure 1.

## 6 Work division

| **Name** | **Task** | **details** |
|---|---|---|
| Romain | Input of the point set | · Creation of two canvas<br>· Input of a point set<br>· Convex Hull computation of the point sets<br>· Validation and verification of the input sets<br>· Verification of general position |
| Romain | Searching other methods to enumerate the triangulations | - |

| | | |
|---|---|---|
| Romain | Code Refactoring | - |
| Dorel | Brute force triangulation enumeration | · Get inner edges of a set's CH<br><br>· Compute K = #inner-edges from a set's triangulation<br>· Get combinations of inner edges with size K<br>· Reject combinations with intersecting edges<br>· Extract the triangles(faces) of the valid combinations |
| Dorel | Generate colors for a set of triangles | - |
| Dorel | Brute force compatible triangulation lookup | · With fixed point ids, get permutations of points ids for the second point set<br>· Get adjacent triangles for each triangle of a triangulation<br>· For each permutation, use the ids to map triangles of the two sets by looking at their neighbours |
| Dorel | Display compatible triangulations in colors | |
| Dorel & Romain | Generator functions for triangulations | Iterate the triangulations one by one on demand instead of computing them all at the start |

## References

[1] Oswin Aichholzera et al. “Towards compatible triangulations”. In: (2003). URL: https://core.ac.uk/download/pdf/82242222.pdf.

[2] Sergei Bespamyatnikh. “An efficient algorithm for enumeration of triangulations”. In: (2002). URL: https://www.sciencedirect.com/science/article/pii/S0925772102001116.

[3] Jeff Danciger, Satyan L. Devadoss, and Don Sheeny. “Compatible triangulations and point partitions by series-triangular graphs”. In: (2005). URL: https://arxiv.org/pdf/cs/0502043.pdf.

[4] Mark de Berg et al. *Computational Geometry : Algorithms and Applications*. 3rd. Springer-Verlag Berlin and Heidelberg GmbH & Co. K (7 mars 2008), 2008. ISBN: 978-3540779735.

[5] Charles Jordan, Michael Joswig, and Lars Kastner. “Parallel Enumeration of Triangulations”. In: (2018). URL: https://arxiv.org/abs/1709.04746.

[6] Zhiguang Liu et al. “High-quality Compatible Triangulations and their Application in Interactive Animation”. In: (2018). URL: http://nrl.northumbria.ac.uk/id/eprint/35739/1/Accepted%20version.pdf.

[7] Anna Lubiw and Debajyoti Mondal. “On Compatible Triangulations with a Minimum Number of Steiner Points”. In: (2018). URL: https://arxiv.org/pdf/1706.09086.pdf.

[8] J. O’Rourke. *Problem 38: Compatible Triangulations*. URL: http://www.science.smith.edu/~jorourke/TOPP/P38.html#Problem.38. (accessed: 09.10.2020).

[9] Alan Saalfeld. “Joint triangulations and triangulation maps”. In: (1987). URL: https://www.census.gov/srd/papers/pdf/rr87-23.pdf.