ENS Lyon

Formalisation of the Delaunay Triangulation

Clément Sartori

09/06/2017

Advisor .... Yves Bertot

Abstract

This report is the result of my internship at INRIA Sophia-Antipolis in the Marelle team, for the validation of the second year of my master's degree in Computer Science. During this internship I formalised Delaunay Triangulations in Coq. My code, although incomplete, can be found here : https://github.com/Nemeras/StageDelaunay.

Contents

1 Introduction

I did my second year of master's degree internship under the supervision of Yves Bertot at the INRIA Sophia-Antipolis in the Marelle team. The goal of this internship was to formalize Delaunay triangulations in the proof assistant Coq.

The goal of this internship was to formalize the Delaunay triangulations in the Coq proof assistant, that is, to try and find a minimal number of geometrical properties that had to be satisfied in order to get Delaunay triangulations. One of the goal of the internship was to stay as close as possible to Knuth's paradigm in [Knu92] with the CC system, by describing the biggest number of geometrical properties using only combinatorial properties such as the orientation predicate "is left of".

1.1 Triangulations and Applications

In geometry, a triangulation T of an object X of $\mathbb{R}^d$ is the partition of this object into simplices such that :

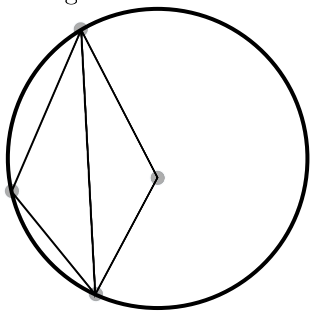
- the intersection of two simplices is either empty or a face of those simplices
- every bounded set of $\mathbb{R}^d$ cuts a finite number of simplices of T
- the union of the simplices of T is X itself.

In the plane (where we stayed during this internship), the triangulation of a set of points is the partition of this set of points into triangles.

More particularly, during this internship, we were interested in Delaunay triangulations : in the plane, a triangulation T of a set P of points is said to be a Delaunay triangulation if no point of P is in the circumcircle of a triangle of T.

This, for instance, is not a Delaunay triangulation :

Figure 1: Instance of a triangulation that is not a Delaunay triangulation



Delaunay triangulations are very useful in robotics. One of the interesting byproducts of Delaunay triangulations is the Voronoi diagram : the Voronoi diagram is the dual graph of the Delaunay triangulation and can be very useful, for instance, to modelise cellular coverage maps [PA08], or for path planning [GMB06].

1.2 Coq/The Mathematical Component library

To make things simpler, I used the Mathematical Component library and the extension of Coq's tactic language, SSReflect. The goal of the Mathematical Component library project is to create a library of mathematical results, so that those result are easily reusable in Coq. Examples of successful application of the Mathematical Component library and SSReflect are the proof of the four-color theorem [Gon08] and of the odd order theorem [GAA$^+$13].

2 The Algorithm

The context of the algorithm studied is, given a set of points in a plane, how to compute the Delaunay triangulation of this set of points. The algorithm studied is an incremental algorithm in two parts : given an existing triangulation and a point, the algorithm consists in

1. Adding the point to the existing triangulation
2. Flipping the illegal edges to obtain a Delaunay triangulation.

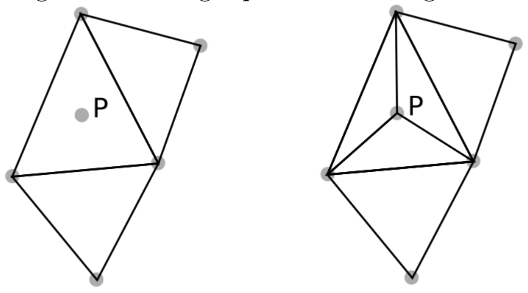
2.1 Adding points

In itself, adding the point to the existing triangulation is not a trivial problem, depending on whether the new point is in the convex hull of the existing points or not. In the litterature ([SD97]), for incremental algorithm, it is generally considered that the sites studied are inclosed within large triangles, which basically means that every new point will be in the convex hull of the previous ones. This was considered to be the case here. If this is not the case, then there is the need of an incremental algorithm to create convex hulls. The formalization of such an algorithm has been done by David Pichardie in [PB01].

Thus, adding a new point P to a triangulation T, when the new point is in the convex hull of the triangulation, is a two step process :

1. The first step is to find the triangle t of the triangulation in which P is
2. The second step is to replace t by three new triangles.

Figure 2: Adding a point to a triangulation

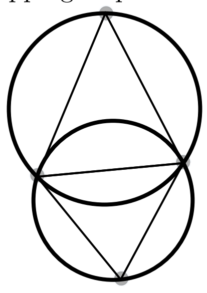


2.2 Flipping edges

It is possible , from any triangulation of a set of point, to obtain a Delaunay triangulation by repeating the same step : for each pair of triangles that do not meet the Delaunay condition, flip the common edge of the triangles.

For instance, while Figure 1 was not a Delaunay triangulation, the result of an edge-flipping step, Figure 3, however, is a Delaunay triangulation
This makes it possible to create a "flipping" algorithm : for any set of points, it is possible to obtain the Delaunay triangulation of this set of point by creating a random triangulation then applying this flipping step as many times as necessary.

Figure 3: Result of an edge-flipping step : example of Delaunay triangulation



3 Formalization of the Delaunay triangulation

The goal of the formalization is to stay as much as possible at a level where geometrical properties, predicates, etc, speak about mathematical concepts (for instance, points) as concepts (for instance, a point being a point versus a point being a pair of coordinates). Basically, the formalization describes what properties every implementation of the concept should satisfy.

Here we describe a formalization of the Delaunay triangulation. Any implementation should contain a type P for the points, a type T for the triangles, a type E for the edges, a type $\mathbb{R}$ for the coordinates of the points (a real integral domain where all elements are positive or negative). Then , any implementation should contain basic functions :

- A function `vertices_to_edge` to create an edge from two points
- A function `vertices_to_triangles` to create a triangle from three points
- A function `vertex` that gives the first, second or third vertex of a triangle
- A function `xCoord` that gives the X coordinate of a point
- A function `yCoord` that gives the Y coordinate of a point

Here, a choice has been made so that proofs are simpler : the order of the vertices of a triangle is important. This means that the triangle that has a, b, c as vertices won't be the same triangle than the triangle that has b, c, a as vertices.

3.1 Geometrical predicates

Then, any implementation should also contain some geometrical predicates :

1. `is_left_of` and `is_left_or_on_line` a b c which test if the point c is (strictly) left of the points a,b, e.g. if a,b,c form a (strictly) well-oriented triangle.
2. `in_circle` p a b c which tests if a point p is in the circumcircle of the triangle described by a,b,c.
3. `in_triangle` t p and `in_triangle_w_edges` t p which test if p is (strictly) in t.
4. `is_on_line` which tests is 3 points are aligned.

Those predicates easily come from another predicate : `oriented_surface` $a\ b\ c$ which computes the oriented surface described by the three points a,b,c :

- `is_left_of a b c` = (`oriented_surface a b c > 0`)
- `is_left_or_on_line a b c` = (`oriented_surface a b c` $\geq$ `0`)
- `is_on_line a b c` = `oriented_surface a b c = 0`

-
  ```
  in_triangle t p =  is_left_of (vertex1 t) (vertex2 t) p ∧
                     is_left_of p (vertex2 t) (vertex3 t) ∧
                     is_left_of (vertex1 t) p (vertex3 t)
  ```

-
  ```
  in_triangle_w_edges t p =  is_left_or_on_line (vertex1 t) (vertex2 t) p ∧
                             is_left_or_on_line p (vertex2 t) (vertex3 t) ∧
                             is_left_or_on_line (vertex1 t) p (vertex3 t)
  ```

.

The `is_left_of` predicate is inspired by Knuth's idea of CC systems [Knu92]. However, unlike him, we don't considere that our points are necessarily in general position. Thus, the opposite of `is_left_of a b c` is not `is_left_of b a c` but `is_left_or_on_line b a c`. A broader explanation of what happens to Knuth's axioms when degenerate cases (aligned points) can happen can be found in [PB01].

We put assumptions (that have to be proven true for any implementation) on the implementations of the predicates/functions. Some are easy :

- `Hypothesis oriented_surface_xx : ∀ x y, oriented_surface x x y` = 0.
- `Hypothesis oriented_surface_circular ∀ a b c, oriented_surface a b c = oriented_surface c a b`.
- `Hypothesis oriented_surface_change :`
  `∀ a b c, oriented_surface a b c = - oriented_surface b a c`.
- `Hypothesis vertices_to_edge_sym : ∀ a b, vertices_to_edge a b = vertices_to_edge b a`.
- `Hypothesis vertices_to_triangle_oriented :`
  `∀ a b c, oriented_triangle (vertices_to_triangle a b c)`.
  `oriented_triangle t` meaning that the triangle t is either well-oriented or flat.
- `Hypothesis is_on_line_trans :`
  `∀ a b c d, a != b → is_on_line a b c → is_on_line a b d → is_on_line a c d`.
- `Hypothesis vertices_to_triangle_correct2 : ∀ p1 p2 p3, ∀ t, (t = vertices_to_triangle p1 p2 p3) -> ((p1 ∈ vertex_set t) ∧ (p2 ∈ vertex_set t) ∧ (p3 ∈ vertex_set t))`.

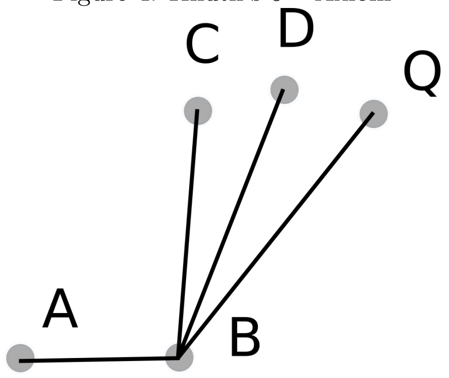
There are also assumptions on the functions that are made to link functions between each other : for instance, to explain what is the edge set of a strictly well-oriented triangle created thanks to vertices-to-triangle :

```
Hypothesis edges_set_vertices_to_triangle:
∀ a b c, is_left_of a b c → edges_set (vertices_to_triangle a b c) =
[ fset (vertices_to_edge a b); (vertices_to_edge a c); (vertices_to_edge b c) ].
```

Some are harder, because they are generalizations of Knuth's axioms or just come from harder geometrical facts:

-
  ```
  Hypothesis is_left_of_trans :  ∀ a b c d q,  is_left_of a b c →
                                               is_left_of a b d →
                                               is_left_of a b q →
                                               is_left_of q b d →  .
                                               is_left_of d b c →
                                               is_left_of q b c
  ```

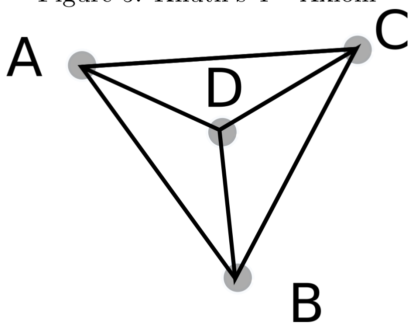
Figure 4: Knuth's 5th Axiom



  The idea to understand this hypothesis is that we're sorting `c d` and `q` in a semi-plane defined by `a` and `b`.

-
  ```
  Hypothesis Axiom4 :  ∀ a b c d,  is_left_of a b d →
                                   is_left_of b c d →
                                   is_left_of c a d →  .
                                   is_left_of a b c →
  ```

Figure 5: Knuth's 4th Axiom



- `Hypothesis on_line_on_edge` (Figure 6) :
  `∀ a b c, is_left_of a b c → ∀ q, is_on_line a c q → is_left_of a b q → is_left_of b c q → on_edge (vertices_to_edge a c) q.`

- `Hypothesis on_edge_on_line` (Figure 6):
  `∀ a b c, is_left_of a b c → ∀ q, on_edge (vertices_to_edge a c) q → is_on_line a c q ∧ is_left_of a b q ∧ is_left_of b c q.`

- `Hypothesis intersection_of_lines` (Figure 7) : `∀ a b c d, is_left_of a b c -> is_on_line a b d -> is_on_line a c d -> d = a.`

Figure 6: Link between the geometrical predicates and being on an edge

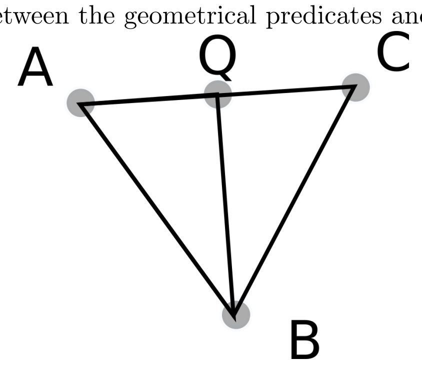
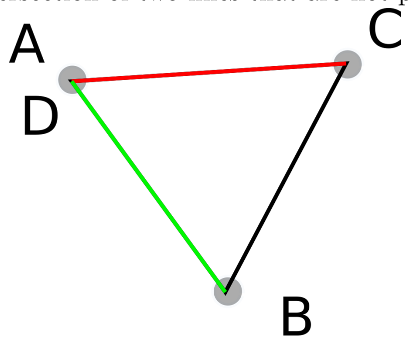


Figure 7: the intersection of two lines that are not parallel is a point !



3.2 Geometrical Properties

Then, we define a triangulation Tr on a data set D as a finite set of triangles with points of D as vertices which satisfies some properties :

1. `covers_hull Tr D` : the whole convex hull of D is covered by the triangles of Tr
2. `covers_vertices Tr D` : each point of D is the vertex of at least one triangle of Tr.
3. `no_cover_intersection` : there is no point which is (strictly) in two different triangles of the triangulation
4. `no_point_on_segment` : no point can be a vertex of a triangle, and on the edge (but not a vertex) of another triangle.
5. `triangle_3_vertices Tr` : every triangle of the triangulation has 3 vertices.
6. `triangle_nempty Tr` : no triangle of Tr is empty (no flat triangle).

The two first properties are here to express that the finite set of triangles Tr is indeed a triangulation of the whole data set D, and covers its convex hull with triangles. Then, the two next properties are here to avoid problems with how the triangles are positioned :

1. `no_cover_intersection` is here to avoid problems like in Figure 8 where two triangles `ABC` and `DEF` aren't well positioned; there exists a point `P` that is in both `ABC` and `DEF`.

2. `no_point_on_segment` is here to avoid problems like in Figure 9, where the vertice `B` of the triangle `ABC` is on the edge `DE` of the triangle `DEF`.

In both case, the properties are here to assure the fact that the first part of the definition of a triangulation is respected : the intersection of two simplices should either be empty or a face of those simplices.

Figure 8: `no_cover_intersection`

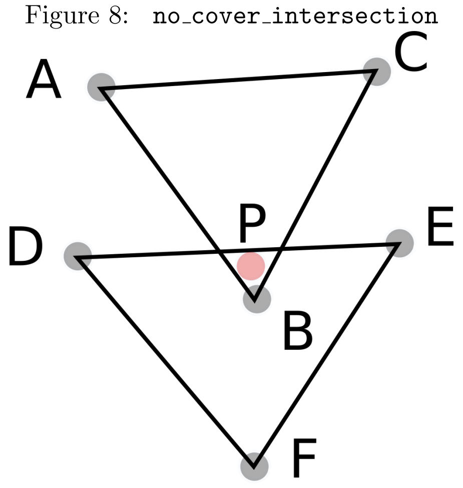
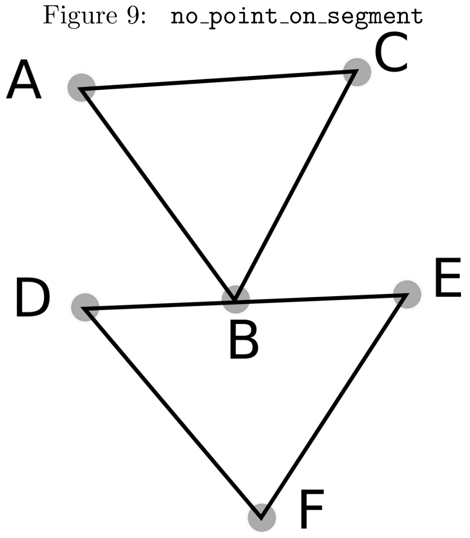


Figure 9: `no_point_on_segment`



Finally, the two last properties are here to avoid degenerate cases with the triangles of the triangulation (triangulation shouldn't contain degenerate triangles). It is to be noted that the second property implies the first.

3.3 Convex Hulls

The definition of what is a triangulation is in dimension 2 requires first to define what is a convex hull of a point set D. Two different approach exist :

1. The first possibility, used by Knuth, is to define the convex hull as a cyclic sequence S : `(S1; S2; ...; Sn; S1)` of elements of D such that every point of D is "encompassed" within S, with `P` being "encompassed" within S iff `is_left_or_on_line Si Si+1 P` is true. It's a more geometrical definition.

2. The second possibility if via the notion of barycenter : it is the smallest subset `H` of `D` such that every point of `D` is a barycenter of the points of `H`.

It's the second approach that was used here. It has two main benefits :

1. First, it is independant of the dimension, whereas the first possibility only works in dimension 2.
2. It is independant to the definition of the geometrical predicates, and those can change (for instance between Knuth's case where points are in general positions and where they are not).

Also, it is very nicely linked with `oriented_surface` in dimension 2. Carathéodory's theorem for Convex hulls says :

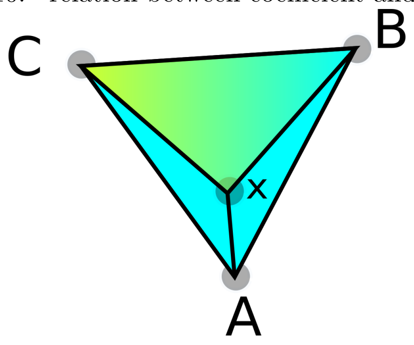
Theorem 1. *If a point of $\mathbb{R}^d$ lies in the convex hull of* `D`*, then it is a barycenter of $d+1$ points of* `D`*.*

In dimension 2, this means that every point in the convex hull of `D` is a barycenter of three points of `D` (ie, in a triangle with vertices some points of `D`). Moreover, there is a nice thing about the coefficients : if x is in ABC then we have :

$$x = x_A A + x_B B + x_C C$$

with : $x_A = \frac{\texttt{oriented\_surface B C x}}{\texttt{oriented\_surface A B C}}$, (...), which gives a nice relation between the coefficients and the surface of the triangles (Figure 10).

Figure 10: relation between coefficient and surface

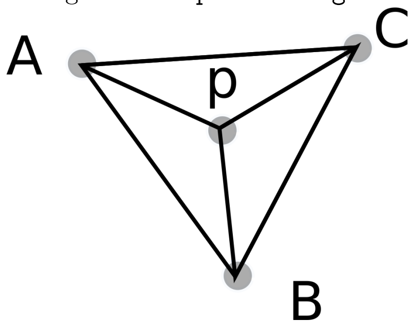


4 Theorems

4.1 Adding points

Following the algorithm described in section 2 the first step was writing a way to add points to a triangulation : given a triangle $t = ABC$, a point p, a triangulation tr, `split_triangle` tr t p returns $tr \setminus t \cup \{ABp; pBC; ApC\}$.

Figure 11: `split_triangle`



The theorem proven was :

```
Theorem triangulation_split_triangle:
∀ tr, t, p, d, d != fset0 -> p ∉ d → triangulation tr d → t ∈ tr ->
in_triangle t p → triangulation (split_triangle tr t p) (p |` d).
```

In english, this would be written :

Theorem 2. *Forall triangulation tr of a nonempty data set d, triangle of this triangulation t, point strictly in this triangle p , if p is not in d, then* `split_triangle tr t p` *is a triangulation for the data set* $d \cup \{p\}$.

This is proven by proving the

4.2 Flipping edges

5 Instantiation of the model

The next step of the work was to do an instantiation of the model :

- a point is a vector of size 2 of reals : `Definition P :=` $'rV[R]_2$`.` Then it's very easy to define xCoord and yCoord : with ordXY being the X element of the Ordinal Y :
- `Definition xCoord (p : P) := p ord10 ord20.`
- `Definition yCoord (p : P) := p ord10 ord21.`
- `oriented surface a b c` can be defined as the determinant of the matrix :

$$\begin{matrix} 1 & \texttt{xCoord a} & \texttt{yCoord a} \\ 1 & \texttt{xCoord b} & \texttt{yCoord b} \\ 1 & \texttt{xCoord c} & \texttt{yCoord c} \end{matrix}$$

- A triangle is a vector of size 3 of points : `Definition T :=` $'rV[P]_3$`.`
- An edge is a vector of size 2 of points : `Definition E :=` $'rV[P]_2$`.`

6 Examples of proofs

7 Conclusion

Bibliography

[GAA$^+$13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin, and David Pichardie, editors, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, July 2013. Springer.

[GMB06] S. Garrido, L. Moreno, and D. Blanco. Voronoi diagram and fast marching applied to path planning. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3049–3054, May 2006.

[Gon08] Georges Gonthier. Formal Proof – The Four-Color Theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, December 2008.

[Knu92] Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer, 1992.

[PA08] Jose N Portela and Marcelo S Alencar. Cellular coverage map as a voronoi diagram. *Journal of Communication and Information Systems*, 23, 2008.

[PB01] David Pichardie and Yves Bertot. Formalizing Convex Hulls Algorithms. In *Proc. of 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'01)*, number 2152 in Lecture Notes in Computer Science, pages 346–361. Springer-Verlag, 2001.

[SD97] Peter Su and Robert L. Scot Drysdale. A comparison of sequential delaunay triangulation algorithms. *Computational Geometry*, 7(5):361 – 385, 1997.