RcppAlphahull

Student: Federico Airoldi - 892377
Tutor: professor Alessandra Menafoglio

December 16, 2019

Abstract

In this report I introduce the package I have built for my project of the exam Advanced Programming for Scientific Computing. The project is a re-realization of an existing R package, `alphahull`: since its routines are too slow, I used the ability of R to call C++ code to re-implement the main functions and increasing so the overall speed of the package. My work is named RcppAlphahull and is now a package that can be installed and downloaded from my public repository on github (`https://github.com/federicoairoldi/ProgettoPACS`).

Contents

1 Introduction and main definitions

Problems of reconstruction of a set A given a finite number of its points are found in different fields, in particular, in computational geometry and in statistics. In this last field the typical goal is to provide an estimate for the support of a probability distribution P_X starting from a random sample of points $X_n = \{x_1, ..., x_n\}$.

If we know that the set A is convex, then the convex hull of our sample is a quite natural estimator for our unknown set (see [1]), but when we don't have convexity then such estimator is not the most appropriate choice. Among the proposed solution to such estimation problems, there's the so called α-convex hull (and the α-shape) introduced in [2] by Edelsbrunner, Kirkpatrick, and Seidel in 1983.

Definitions and results of this paper are closely related to the Voronoi tessellation and the Delaunuay triangulation named after the Russian mathematicians Georgy Feodosevich Voronoy and Boris Delaunay. Here we report them.

Definition 1.1 (Voronoi diagram)**.** Given a set $X_n = \{p, ..., q\}$ of n points in $\mathbb{R}^2$ its Voronoi diagram $VD\left(X_n\right)$ is a covering of the plane by n regions V_p $p \in X_n$ such that

$$V_p = \left\{x \in \mathbb{R}^2 \mid d\left(x, p\right) \leqslant d\left(x, q\right) \ \ \forall q \in X_n\right\}$$

Two points p and q in X_n are Voronoi neighbours if V_p and V_q share a common edge and, on such edge, the points are equally distant from p and q.

Remark 1. All of the cells V_i are closed and convex, some of them are unbounded and correspond to those points which are on the boundary of the convex hull of X_n while the others are bounded and correspond to those points which are in its interior. Also, the edge v between two Voronoi neighbours points p and q lies on the bisector of such points.

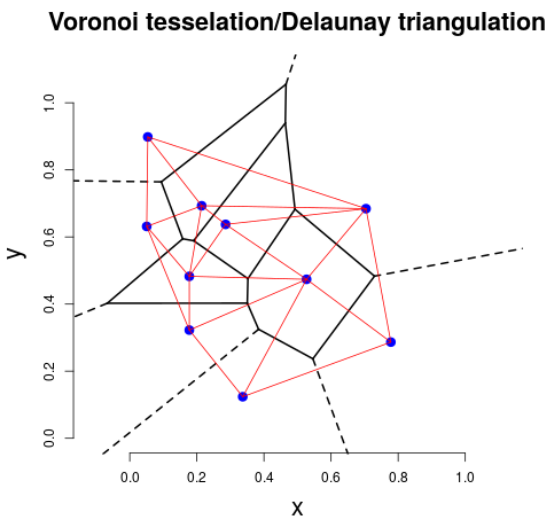


Figure 1: example of Voronoi tesselation (in black) and Delaunay triangulation (in red) of a randomized set of 10 points. Dotted lines are infinite edges of the Voronoi diagram. Points are generated with the following code:

```
n = 10
set.seed(325)
x = runif(n)
y = runif(n)
vor = RcppAlphahull::delvor(x,y)
```

Definition 1.2 (Delaunay triangulation)**.** The Delaunay triangulation of a set $X_n = \{p, ..., q\}$, denoted by $DT(X_n)$, is defined as the dual of its Voronoi diagram $VD(X_n)$, that is, two points p and q are connected in the triangulation if they are Voronoi neighbours.

1.1 α-convex hulls

In their paper [2] Edelsbrunner et al. introduced the definition of α-shape, α-convexity and α-convex hull for any arbitrary $\alpha \in \mathbb{R}$. However we will restrict ourself to the definition of such concepts for $\alpha > 0$ by slightly modifying the notation introduced by the authors. Pateiro-Lpez and Rodriguez-Casal have done the same in their reference paper for the package alphahull [6].

Before looking at the main definitions we first start by denoting with the notation $\mathring{\mathrm{B}}(x, r)$ and $B(x, r)$ respectively an open and a close ball in $\mathbb{R}^d$ with center in x and radius r and, as usual, with A^c and ∂A the complement and the boundary of a set A.

Definition 1.3 (α-convexity)**.** A set $A \subset \mathbb{R}^d$ is said to be α-convex for $\alpha > 0$ if

$$A = C_\alpha(A) = \bigcap_{\mathring{\mathrm{B}}(x,\alpha) \,\mid\, \mathring{\mathrm{B}}(x,\alpha)\cap A=\varnothing} \left[\mathring{\mathrm{B}}(x, \alpha)\right]^c$$

The set $C_\alpha(A)$ is the α-convex hull of A.

This implies that for any point y outside an α-convex set A there exists an open ball $\mathring{\mathrm{B}}(x, \alpha)$ that falls completely outside such set and that contains y.

Moreover this definition resembles that one of the convexity/convex hull, but it uses open balls instead of halfplanes. Also one can show the following:

Lemma 1.1. If a set A is convex, then it is α-convex for any $\alpha > 0$.

Performances of the α-convex hull as estimator are analysed in [3] where Rodriguez-Casal proved that if A is α-convex and satisfies a standardness condition (w.r.t. the distribution P_X of the sample points) and $X_n = \{p, ..., q\}$ is a random sample of its points, then using $C_\alpha(X_n)$ as an estimator for A leads to a convergence in Hausdorff distance like the following

$$d_H(S, C_\alpha(X_n)) = O\left(\left(\frac{ln(n)}{n}\right)^{\frac{1}{d}}\right)$$

Such convergence gets faster if A belongs to the Serra's regular models [3, 4].

In [2] there are some theoretical results useful to build the α-convex hull of a set of points; starting from the definition we can write

$$C_\alpha(A) = \bigcap_{\mathring{\mathrm{B}}(x,\alpha)\ \mid\ \mathring{\mathrm{B}}(x,\alpha)\cap A=\varnothing} \left[\mathring{\mathrm{B}}(x,\alpha)\right]^c = \left[\bigcup_{\mathring{\mathrm{B}}(x,\alpha)\ \mid\ \mathring{\mathrm{B}}(x,\alpha)\cap A=\varnothing} \mathring{\mathrm{B}}(x,\alpha)\right]^c$$

With such notation we can describe the α-convex hull via its complement and, as it's proved in [2], such set is constituted by open balls and halfplanes:

Lemma 1.2. The α-convex hull of a set $X_n = \{p, ..., q\}$ can be expressed as the complement of the union of $O(n)$ open discs with radius $\geqslant \alpha$ and with centers on the edges of $VD(X_n)$ and halfplanes.

The proof of this lemma is constructive and so it gives us all of the necessary information to find the open balls and halfplanes than we need, and in particular it states that we need only to consider appropriate discs centered on edges of $VD(X_n)$:

- if the Voronoi edge v (between points p and q) is infinite then we have to add to the complement an halfplane and, if there exist points on v at distance α from p, an open ball for each one of them;
- if the Voronoi edge $v = (e_1, e_2)$ (between points p and q) is finite then we have to check whether there are points on the bisector at distance α between p and q that lie in v too and distinguish the different cases (see subsection 3.2.5).

1.2 α-shapes

The α-shape instead relies on the concept of α-neighbour points, which in turn, is constructed by using the definition of α-extreme point.

Definition 1.4 (α-extreme)**.** Given $\alpha > 0$, a point p in a set A is said to be an α-extreme if there exists an open ball $\mathring{\mathrm{B}}(x, \alpha)$ such that $p \in \partial\mathring{\mathrm{B}}(x, \alpha)$ and it doesn't contain any other point of A.

A consequence of this statement is that the points lying on the convex-hull's boundary of a set are α-extreme for any $\alpha > 0$. Starting from this last definition we can introduce the one for α-neighbour points.

Definition 1.5 (α-neighbour points)**.** Given $\alpha > 0$, two α-extreme points p and q in A are said to be α-neighbours if there exists an open ball $\mathring{\mathrm{B}}(x, \alpha)$ such that $p, q \in \partial\mathring{\mathrm{B}}(x, \alpha)$ and it doesn't contain any other point of A.

And lastly we can state the definition of the α-shape.

Definition 1.6 (α-shape)**.** Given $\alpha > 0$, the α-shape of a set A is defined as the straight line graph connecting the pairs of its α-neighbour points.

Some useful results come from the definitions we have introduced so far in particular I report the main ones that I used in order to implement their construction in my package RcppAlphahull.

Lemma 1.3. For each point p in a finite set X_n there exists a value $\alpha_{max}(p)$ such that p is an α-extreme for $0 < \alpha \leqslant \alpha_{max}(p)$.

The proof of this lemma highlights how we can find such values for all the points of our set, in particular:

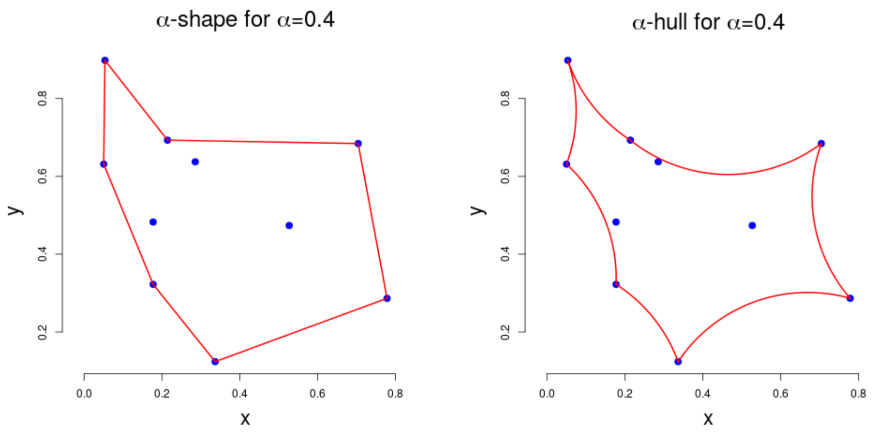
- if p lies on the boundary of the convex hull of X_n then it is α-extreme for any $\alpha > 0$;
- if p isn't on the convex hull's boundary then its limit value is $\alpha_{max}(p) = max\{d(x,p) \mid x \in V_p\}$ where V_p is the Voronoi cell of p and, since for non boundary points V_p is a closed convex polygon, we can find such maximum by looking at V_p's vertices: $\alpha_{max}(p) = max\{d(x,p) \mid x \text{ is a vertex of } V_p\}$.

This criterion helps us to check for a given value of α which of our points are α-extreme, and consequently we can use the next lemma to decide which pairs of them are α-neighbours; the next statement is based on the fact that the α-shape of a finite set of points A is a sub-graph of its Delaunay triangulation.

Lemma 1.4. For any edge $e = (p, q)$ of $DT(X_n)$ there exist two real values $\alpha_{min}(e)$ and $\alpha_{max}(e)$ such that e is a an edge of the α-shape for any $\alpha_{min}(e) \leqslant \alpha \leqslant \alpha_{max}(e)$. In this interval the two points p and q are so α-neighbours.

In this case too, the proof is constructive and tells us that in order to compute such limit values we have to watch the dual Voronoi edges:

- if $e = (p, q)$ is not a convex-hull edge then $\alpha_{min}(e) = min\{d(x,p) \mid x \in v\}$ and $\alpha_{max}(e) = max\{d(x,p) \mid x \in v\}$ where $v = (e_1, e_2)$ is the Voronoi dual edge of e;



(a) α-shape (b) α-convex hull

Figure 2: α-shape and α-convex hull for the value $\alpha = 0.4$ for the same 10 points of example in figure 1. Computation performed with functions `ashape` and `ahull` of package `RcppAlphahull` (see subsection 3.2).

- if $e = (p, q)$ is a convex-hull edge then there's no upper bound for $\alpha_{max}(e)$ so I intuitively fix it to be infinite, yet the lower bound is the same.

I used such lemmas in the implementation of my version of the function ashape (see subsection 3.2.3).

1.3 `alphahull`: an R package

The R package `alphahull` [5] implements some functions that compute the Voronoi tessellation/Delaunay triangulation, the α-shape and the α-convex hull for a set of points in $\mathbb{R}^2$. Its main functions are `delvor`, `ashape` and `ahull` and all of them return an `S3` object (that is a named list) that contains the information of the requested object that respectively are the Voronoi tessellation/Delaunay triangulation, the α-shape and the α-convex hull. Such functions are dramatically slow probably because of the R inefficient management of cycles. This inefficiency calls for an hybrid implementation of the package using C++ which is much more faster.

2 Geometrical classes

For the algorithms implementation of my package I defined some classes describing the $\mathbb{R}^2$ geometrical concepts that are used to define the theoretical structures described in subsections 1.1 and 1.2. All of them are template classes where the needed precision has to be specified. The starting point of such classes is the `MyGAL`'s Vector2 that describes a point/vector in $\mathbb{R}^2$ and they're stored, with the header file `geomUtil.h`,in the folder `RcppAlphahull/src/newClasses`. In figure 3 I report the UML class diagram of such folder.

2.1 Segment

This class describes a segment in $\mathbb{R}^2$: to describe a single segment I use the two points that it connects together. This structure is helpful in distinguish the different cases that can occur while computing the α-shape (see subsubsection 3.2.3), in particular the method `intersect`, that is implemented as both a member and non member function, returns true if two segment share a common point. The method relies on the construction of the two lines, on which the segments lie, and mutually checks the position of on their extremes with respect the other line.

Algorithm 1 Do segments $s_1 = (p_1, q_1)$ and $s_2 = (p_2, q_2)$ intersect?

```
1: retrieve rect $r_1$ of the first segment $s_1 = (p_1, q_1)$
2: retrieve rect $r_2$ of the first segment $s_2 = (p_2, q_2)$
3: evaluate position of $p_1$ w.r.t. $r_2$ with $val_1 = sign(a_1 \cdot y + b_1 \cdot x + c_1)$
4: evaluate position of $q_1$ w.r.t. $r_2$ with $val_2 = sign(a_1 \cdot y + b_1 \cdot x + c_1)$
5: evaluate position of $p_2$ w.r.t. $r_1$ with $val_3 = sign(a_2 \cdot y + b_2 \cdot x + c_2)$
6: evaluate position of $q_2$ w.r.t. $r_1$ with $val_4 = sign(a_2 \cdot y + b_2 \cdot x + c_2)$
7: if $val_1$ != $val_2$ and $val_3$ != $val_4$ then
8:    the segments do intersect
9: end if
10: if $val_1 = val_2 = 0$ (consequently $val_3$ and $val_4$ are 0 too) then
11:    if coordinates intersect then
12:       the segments do intersect (actually they overlap)
13:    end if
14: end if
15: the segments don't intersect
```

2.2 Line

This class describes a line in $\mathbb{R}^2$ using its implicit form $a \cdot y + b \cdot x + c = 0$; I choose this form instead of the explicit one, $y = m \cdot x + q$, since the latter fails in representing lines of the form $x = x_r$. However in the member functions there are two methods which can compute the slope m and the intercept q for those lines which can be represented in explicit form. Such methods returns a $+\infty$ and $-\infty$ for lines that cannot be represented in such form respectively for the slope and the intercept; for getting information on these lines one can rely on the member method `x_r()` that for non vertical lines returns a quiet NaN.

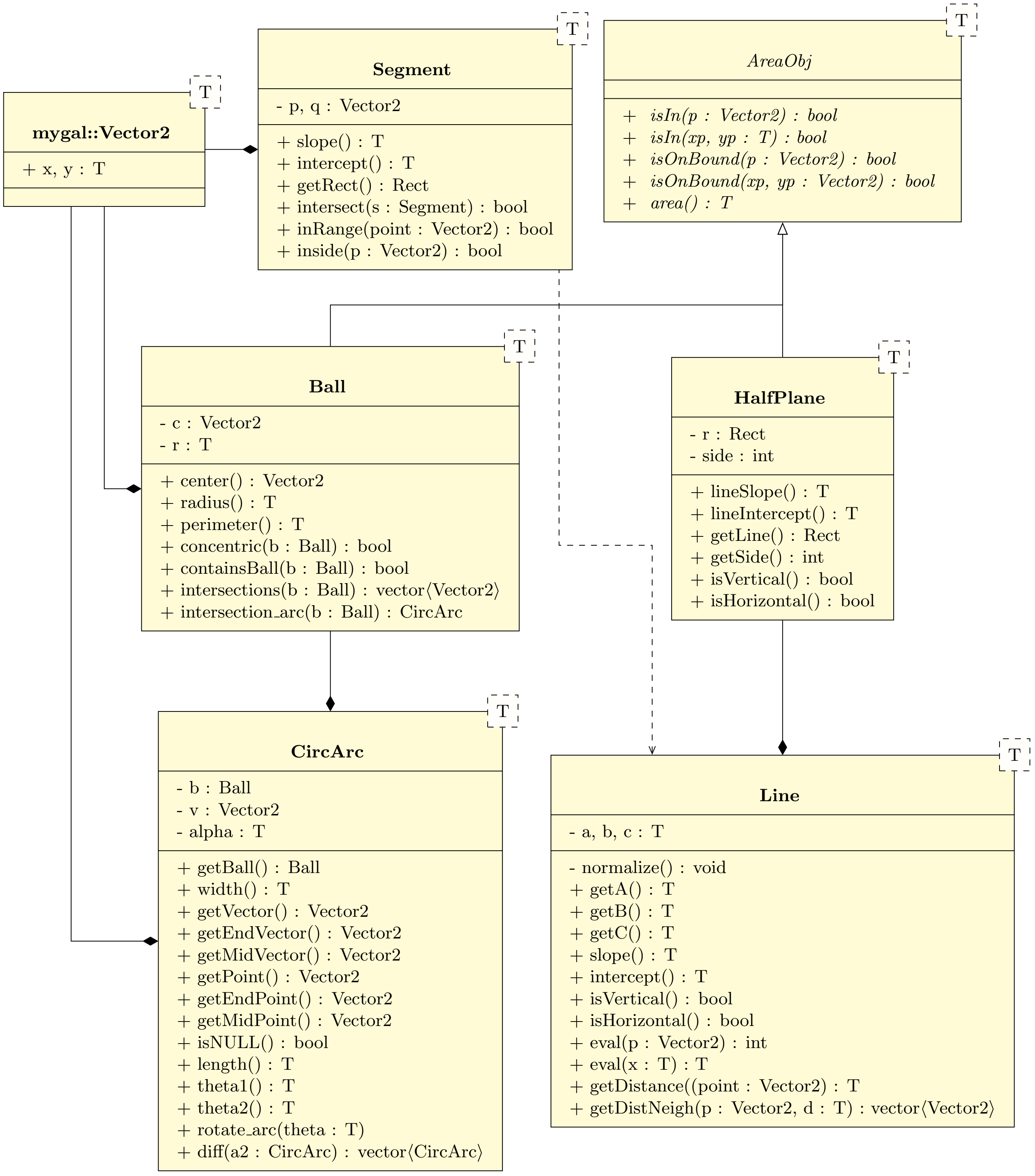


Figure 3: UML diagram of my geometrical classes. Here I report the member method, all of them implement a non member `operator<<` function to insert them in a stream.

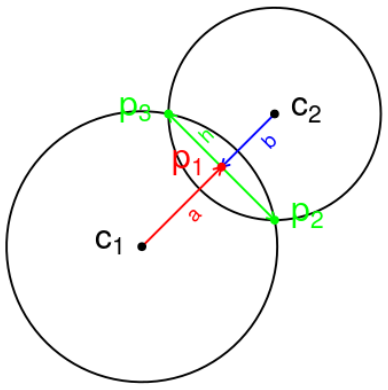


Figure 4: intersection point between two balls in $\mathbb{R}^2$. The red vector has direction $v = c_2 - c_1$ and magnitude a, the blue one has direction $-v$ and magnitude $b = d - a$ and the greens have direction v_{norm} and magnitude h.

2.3 AreaObj

This abstract class defines some basic methods that geometrical objects with an area should implement, for instance a method that check whether or not a point falls in its interior. I used this class as a base for the following classes HalfPlane and Ball.

2.4 HalfPlane

This class describes an halfplane in $\mathbb{R}^2$: each halfplane is described by a line and by an integer that denotes which side is covered by the halfplane, in particular:

- `side = 1`, then the halfplane is $a \cdot y + b \cdot x + c > 0$;
- `side = -1`, then the halfplane is $a \cdot y + b \cdot x + c < 0$.

For such distinction I decided to use an integer value instead of a boolean since in this way I can check whether a point belongs to the halfplane or not by using the function `sign<T>` in `geomUtil.h` and by using the method `eval(const vectro2&)` of the class `Line`.

2.5 Ball

This class describes a ball in $\mathbb{R}^2$ by its center and radius. One of the main useful method I implemented for this class is the one that returns the intersection points of two non concentric balls, here I report the computations in algorithm 2 and an example in figure 4.

Algorithm 2 Intersection points between two balls b_1 and b_2

```
1: if $b_1$ and $b_2$ are concentric then
2:     return an empty vector of Vector2
3: end if
4: compute the distance between the centers $d = d(c_1; c_2)$
5: if $d > r_1 + r_2$ or $d < |r_1 - r_2|$ then
6:     return an empty vector of Vector2
7: end if
8: compute $a = \frac{d^2 + (r_1^2 - r_2^2)}{2d}$
9: compute $h = \sqrt{r_1^2 - a^2}$
10: compute $v = c_2 - c_1$ vector from $c_1$ to $c_2$
11: compute $v_{norm}$ normal vector of $v$
12: compute vector/point $p_1 = c_1 + \frac{a}{d}(c_2 - c_1)$
13: compute vectors/poinst $p_2 = p_1 - \frac{h}{d}v_{norm}$ and $p_3 = p_1 + \frac{h}{d}v_{norm}$
14: if $h > 0$ then
15:     return both $p_2$ and $p_3$
16: else
17:     return just $p_2$
18: end if
```

2.6 CircArc

This class describes a circular arc in $\mathbb{R}^2$ and is denoted by the ball on which upon is constructed, the vector pointing to its starting point on the circumference and by its angle width. In this way it's easier to handle the construction of new arcs since the only requirement I need to check is that the angle is in $[0; 2\pi]$. Angles of value 0 denote a null arcs; I used that in the methods that computes the differences of arcs on a circumference.

In this class I implemented a method to compute the set difference between arcs on the same circumference, it's description can be seen in algorithm 3. By difference $a_1 - a_2$ of two arcs a_1 and a_2 I mean the part of a_1 that does not belong to a_2.

2.7 geomUtil.h

In this header file I have written some functions that can be useful when working with the previously reported Classes. It includes some functions that works on objects of class Vector2 that, unluckily, weren't implemented in the library `MyGAL` and an important method that computes the boundary of the union of a set of balls employing methods and functions that works on the previously describes class `CircArc`.

Algorithm 3 Difference of arcs a_1 and a_2

```
if $\alpha_2 = 2\pi$ then
    nothing remains of arc $a_1$
end if
if $\alpha_2 = 0$ then
    return $a_1$
end if
if $\theta_1$ of $a_1$ is not 0 then
    rotate both $a_1$ and $a_2$ by the angle $-\theta_1$ of $a_1$
    compute the difference for the rotate arcs
    rotate the difference by the angle $\theta_1$ of $a_1$
    return the final result
end if
select the appropriate case
case 1: $0 < \alpha \leqslant \theta_1^{a_2} < \theta_2^{a_2}$:
    $a_1$ has no common part with $a_2$
    return $a_1$
case 2: $0 < \theta_1^{a_2} < \alpha \leqslant \theta_2^{a_2}$:
    return arc with starting vector $v = (0; 1)$ with angle $\theta_1^{a_2}$
case 3: $0 < \theta_1^{a_2} < \theta_2^{a_2} < \alpha$:
    if $\alpha == 2\pi$ then
        return an arc starting from the ending point of $a_2$ with angle $\alpha - \theta_2^{a_2} + \theta_1^{a_2}$
    else
        return a couple of arcs one starting at $v = (0; 1)$ with angle $\theta_1^{a_2}$ and starting from the ending point of $a_2$ with angle $\alpha - \theta_2^{a_2}$
    end if
case 4: $0 = \theta_1^{a_2} < \alpha \leqslant \theta_2^{a_2}$:
    $a_1$ is entirely in $a_2$
    return a null arc
case 5: $0 < \theta_1^{a_2} < \theta_2^{a_2} < \alpha$:
    return an arc starting from the ending point of $a_2$ with angle $\alpha - \theta_2^{a_2}$
case 6: $0 < \theta_2^{a_2} < \alpha \leqslant \theta_1^{a_2}$:
    return an arc starting from the ending point of $a_2$ with angle $\alpha - \theta_2^{a_2}$
case 7: $0 < \alpha \leqslant \theta_2^{a_2} < \theta_1^{a_2}$:
    $a_1$ is entirely in $a_2$
    return a null arc
case 8: $0 \leqslant \theta_1^{a_2} < \theta_2^{a_2} \leqslant \alpha$:
    return an arc starting from the ending point of $a_2$ with angle $\theta_1^{a_2} - \theta_2^{a_2}$
```

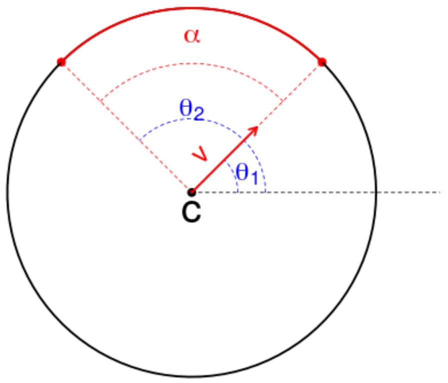


Figure 5: example of an arc on a circumference, the vector v has unitary norm and points in the direction of the starting point of the arc. angles θ_1 and θ_2 are used for the computation of difference of arcs.

3 RcppAlphahull

In order to rewrite the `alphahull` package I created another one with the help of the R package `Rcpp` [9] that allows to add additional C++ scripts and headers to an R package. I designed my R and C++ functions of `RcppAlphahull` in order to get a set of objects describing the different geometrical concepts as close as possible to the ones returned by the `alphahull` methods, in particular the functions `delvor`, `ashape` and `ahull` return objects of the same `S3` classes of those returned by the corresponding original versions. Moreover, the objects of type `ashape` and `ahull` can be plotted by the existing plot function implemented in the latter package.

Documentation on how to use the functions can be seen in the .pdf reference manual *RcppAlphahull_1.0.pdf* or by typing `help(*function_name*)` in R/`Rstudio` after one has load the library. Both of them are synthesized by using the packages `devtools` [10] and `roxygen2` [11], that automatically create the help and .pdf documentation once the R code has been commented.

3.1 Structure of the package

I decided to implement the package's functions to be as close as possible to the corresponding ones in the package alphahull in terms of syntax: for each of the following functions (except `plot.delvor`) I have created an R wrapper to handle the different input cases and a .cpp file that takes care of the actual computations. The R files are contained in the folder `RcppAlphahull/R` of the package while the .cpp are in the `RcppAlphahull/src` one. The package directory structure is shown in figure 6.

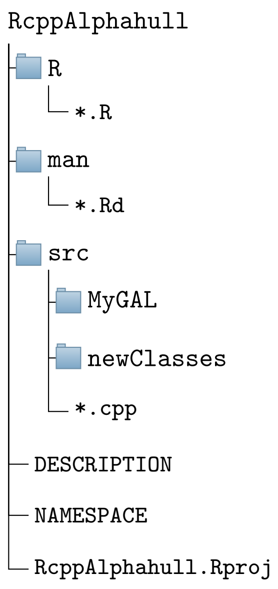


Figure 6: directory tree of the `RcppAlphahull` package, folder R contains the R function wrappers, folder man contains the .Rd file for the inline documentation and folder src contains the .cpp files. DESCRIPTION, NAMESPACE and RcppAlphahull.Rproj are files generate by R for the installation.

3.2 Re-implemented functions

3.2.1 delvor

The `alphahull` function `delvor` employs routines from another package, `tripack` [7], to compute the Voronoi diagram and, starting from its result, it extracts a matrix that describes both the tessellation and the triangulation. Its output is a list with three components (see [5]):

- `mesh`: a matrix describing the Voronoi tesselation and the Delaunay triangulation;
- `x`: a matrix containing the input points coordinates;
- `tri.obj`: an object of class `tri` returned by the tripack package.

My re-implementation of this function relies on an external C++ library named `MyGAL` [12] that implements the Fortune Algorithm to compute in $O\left(n \cdot ln(n)\right)$ time the Voronoi tessellation and the Delanuay triangulation of a finite set of points in $\mathbb{R}^2$ (details can be found in [8]). Headers of such library are included in the folder `RcppAlphahull/R` of my package. Before looking at the construction I have to say that I needed to overload one of the library's methods: the author implemented the function `bound` in the class `FortuneAlgorithm` in order to cut the tightest as possible the infinite edges of the Voronoi diagram. For my goals this was not convenient so I wrote a modified version of such function that takes in input a pointer to an object `Box` and a reference to a distance value and next updates its limits by assuring that the bounding bigger than the smallest one needed to contain all of the sites. Once the Voronoi diagram is computed the matrix `mesh` can be constructed by iterating the edges of the variable `diagram`. I suggest to look at subsection 3.1 in [6] for a complete description of such matrix. As already said such function is in the class `FortuneAlgorithm` and its declaration is `bool bound(Box<T>* box, const T& dist)`.

I decided to include the whole MyGAL library in the source files since in general R users like to install libraries with just one command (or at most with very few commands) and this eases the installation procedure.

In my version of the function I built a `tri` object slightly different from the original one since the new version of this structure is an `S3` object of class `tri.mod` with the following components:

- `n`: number of sites of the triangulation;
- `x`: x coordinates of the sites;
- `y`: y coordinates of the sites;
- `neighbours`: list of n integer vectors where the i-th vector contains the indices of the neighbours sites of the i-th site.

So the final output of my function is an `S3` list of class `delvor` with the following components:

- `mesh`: a matrix describing the Voronoi tesselation and the Delaunay triangulation;
- `x`: a matrix containing the input points coordinates;
- `tri.obj`: an object of class `tri.mod` describing the Delaunay triangulation.

The computations are performed in the `computeVoronoiRcpp` function in file `voronoi_Rcpp.cpp`. Instructions on how to use the R corresponding function can be found in the .pdf manual in the repository or by typing `help(delvor)` in R's command line.

3.2.2 plot.delvor

Since my `delvor` object is slightly different to the original one I had to re-implement also the corresponding method for the visualization to handle the different object that stores the Delaunay triangulation. The function, however, preserves the attribute and so its use in coding.

Instructions on how to use this function can be found in the .pdf manual in the repository or by typing `help(plot.delvor)` in R's command line.

3.2.3 ashape

The output of the ashape function in alphahull is an `S3` list of class `ashape` with the following components:

- `edges`: number of sites of the triangulation;
- `length`: length of the α-shape;
- `alpha`: value of α for which the shape is computed;
- `alpha.extremes`: vector containing indices of those sites which are α-extreme for the provided value of α;
- `delvor.object`: a `delvor` object like the one returned by the function delvor describing the Voronoi/Delaunay diagram underlying the α-shape.

In my version of this function the output has the same form, yet I recall again that actually my `delvor` object is slightly different from the original one (see subsection 3.2.1).
As specified in subsection 1.6, the α-shape edges are a subset of those of the Delaunay triangulation and so, in order to select the appropriate ones, I relied on lemmas 1.3 and 1.4 and implemented two functions:

- `computeAlphaLimits`, that for each site p of the Voronoi tesselation compute the limit value $\alpha_{max}(p)$;
- `getAlphaNeighbours`, that, knowing which sites are α-extreme, returns the rows of the `delvor` mesh matrix that compose the α-shape.

The first functions just check each row of the matrix `delvor$mesh` and evaluates the limit values by checking whether or not the cell's sites have one infinite edge, and in such cases $\alpha_{max}(p)$ is set to infinity, or if they're bounded, and in this cases $\alpha_{max}(p) = max\{d(x,p) \mid x \text{ is a vertex of } V_p\}$.

For the second function, the computation of $\alpha_{min}(e)$ and $\alpha_{max}(e)$ for the candidate Delaunay edge $e = (p,q)$, where p and q are α-extreme, we have to distinguish some cases. We denote with $v = (e_1, e_2)$ the finite version of the dual Voronoi edge of e, with r the rect passing per e and with r_{bis} the bisector of p and q (note that segemet v is by construction on r_{bis}):

1. Segment (p,q) intersect (e_1, e_2) and v is finite (figure 7a), then

$$\alpha_{min}(e) = \frac{d(p,q)}{2} = d(p, r_{bis}) \text{ and } \alpha_{max}(x) = max\,(d(p,e_1), d(p,e_2))$$

2. Segment (p,q) doesn't intersect (e_1, e_2) and v is finite (figure 7b), then

$$\alpha_{min}(e) = min\,(d(p,e_1), d(p,e_2)) \text{ and } \alpha_{max}(x) = max\,(d(p,e_1), d(p,e_2))$$

3. Segment (p,q) intersect (e_1,e_2) and v is infinite (figure 7c), then

$$\alpha_{min}(e) = \frac{d(p,q)}{2} = d(p, r_{bis}) \text{ and } \alpha_{max}(x) = \infty$$

4. Segment (p,q) doesn't intersect (e_1,e_2) and v is infinite (figure 7d), then

$$\alpha_{min}(e) = min(d(p,e_1), d(p,e_2)) \text{ and } \alpha_{max}(x) = \infty$$

Once that for an edge we have computed such limits then we have only to check the condition reported in 1.4.

The computations (see algorithm 4) are performed in the `computeAshapeRcpp` and its auxiliary functions in file `ashape_Rcpp.cpp`. Instructions on how to use the R corresponding function can be found in the .pdf manual in the repository or by typing `help(ashape)` in R's command line.

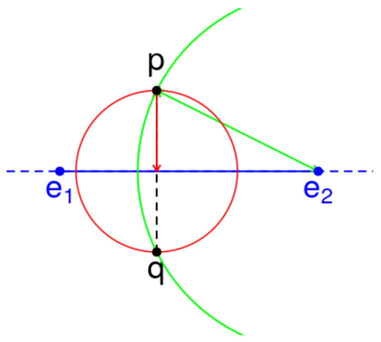
Algorithm 4 Computation of the α-shape

```
1: finding $\alpha$-extreme sites
2: for each site $p$ do
3:     if $V_p$ is unbuonded then
4:         $\alpha_{max}(p) = \infty$
5:     else
6:         $\alpha_{max}(p) = max\{d(x,p) \mid x \text{ is a vertex of } V_p\}$
7:     end if
8:     if $\alpha \leqslant \alpha_{max}(p)$ then
9:         $p$ is an $\alpha$-extreme site
10:    end if
11: end for
12: checking which couples of $\alpha$-extreme sites are $\alpha$-neighbours
13: for each couple/edge of $\alpha$-extreme sites $e = (p,q)$ do
14:     compute $\alpha_{min}(e)$ and $\alpha_{max}(e)$ according to the case
15:     if $\alpha_{min}(e) \leqslant \alpha_{max}(e)$ then
16:         $e = (p,q)$ is an edge of the $\alpha$-shape
17:     end if
18: end for
```

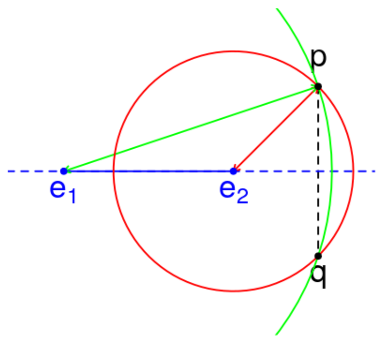
3.2.4 complement

The output of `alphahull`'s function `complement` is not an `S3` object, but simply a matrix containing the information about halfplanes and balls that form the α-hull. For details on such matrix see subsection 3.3 in [6] or the documentation of alphahull [5].

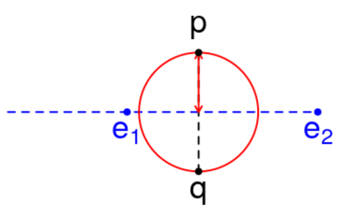
To define which are the balls and halfplanes that form the α-hull complement



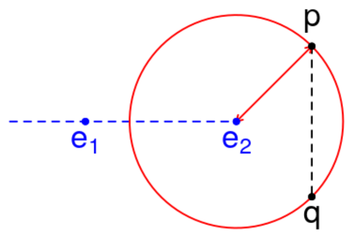
(a) case 1



(b) case 2



(c) case 3



(d) case 4

Figure 7: This figure illustrates the possible case that can occur in the evaluation of edges for the α-shape computation: p and q are two α-extreme and Voronoi neighbours sites, e_1 and e_2 describe the common Voronoi edge in blue and the green and red arrows show respectively $\alpha_{max}\,(e)$ and $\alpha_{min}\,(e)$.

I relied on the final results of [2] (see consequences of lemma 5) and used the methods and classes I introduced in section 2. The results allow us to say that, to retrieve those balls and halfplanes that constitute the complement, we just have to consider appropriate halfplanes or balls centered on the Voronoi edges of our sites. This turns into a classification of the points on a Voronoi edges as is shown in algorithm 5.

Algorithm 5 Computation of the α-hull complement

```
1: for each row of matrix mesh do
2:     retrieve coordinate of $p$, $q$, $e_1$ and $e_2$
3:     compute rect $r$ passing through $p$ and $q$
4:     compute rect $r_{bis}$ bisectrix of $p$ and $q$
5:     build the finite Voronoi edge $v = (e_1, e_2)$
6:     compute the distance from $p$ of $e_1$ (infinite if $bp1 = 1$)
7:     compute the distance from $p$ of $e_2$ (infinite if $bp2 = 1$)
8:     find the points on $r_{bis}$ at distance $\alpha$ from $p$ (can be 0, 1 or 2)
9:     if $bp1 = 1$ then
10:        add an halfplane to the complement description
11:    else if $e_1$ is distant at least $\alpha$ from $p$ then
12:        add a ball centered in $e_1$ to the complement description
13:    end if
14:    if $bp2 = 1$ then
15:        add an halfplane to the complement description
16:    else if $e_2$ is distant at least $\alpha$ from $p$ then
17:        add a ball centered in $e_1$ to the complement description
18:    end if
19:    for each point on $r_{bis}$ at distance $\alpha$ do
20:        if point falls in the Voronoi edge then
21:            add a ball centered in point to the complement description
22:        end if
23:    end for
24: end for
```

The algorithm 5 is implemented in the `computeComplementRcpp` function in file `complement_Rcpp.cpp`. Instructions on how to use the R corresponding function can be found in the .pdf manual in the repository or by typing `help(ashape)` in R's command line.

3.2.5 ahull

The function `ahull` in package `alphahull` returns an `S3` list of class `ahull` with the following components:

- `arcs`: a matrix describing the arcs that form the α-hull boundary;

- `xahull`: a matrix containing coordinates of original set of sites beside new end points for the arcs of the boundary;
- `length`: length of the α-hull boundary;
- `complement`: output matrix of function `complement`;
- `alpha`: value of α for which the hull is computed;
- `ashape.obj`: object of class `ashape` returned by the function `ashape`.

In my version the complement computation is delegated to the function `computeComplementRcpp` and so what remains is to find the arcs that form the α-hull boundary and I have done it in two steps:

Algorithm 6 Computation of the α-hull boundary

```
1: compute the boundary of the union of balls
2: remove those arcs that falls in the union of halfplanes
```

The first step is done by employing the apposite function `union_boundary` defined in the file `geomUtil.h` and the latter is done by considering the fact that the arcs that need to be remove have their extreme points at most on the boundary of the halfplanes, therefore I select them by checking whether or not the point at the middle of the arc is inside any of the halfplanes. To perform such task I put the arcs that describes the union's boundary in a list and then, by iterating over all the halfplanes and by using the function `remove_if` and a lambda function, I discard those that fall outside.

As last remark I'd like to highlight the fact that in the paper [6] the authors state that the boundary of the α-hull is constituted only by circumference arcs with radius α, but, since I didn't found any confirm or proof of such statement, I decided to compute it by using all the arcs that I get from the matrix describing the complement. In this way I get a description with arcs with radius $\geqslant \alpha$ differently from the original package and this prevented the possibility to compare results automatically. Moreover in my `xahull` component just includes the extreme points of the arcs that form the α-hull's boundary.

The computations are perfomed in the `computeAhullRcpp` function in file `ahull_Rcpp.cpp`. Instructions on how to use this function can be found in the .pdf manual in the repository or by typing `help(ahull)` in R's command line.

3.2.6 inahull

Looking at the codes in `alphahull` functions I noted that this particular one, `inahull`, was implemented in a non optimal way: its task is the one of saying whether or not a given point falls in inside an α-hull by spanning the halfplanes and the balls that constitute the complement and by checking if one of them contains it. It's implementation however does a lot of non-needed work since it doesn't stop when it encounters an halfplane or a ball of the complement that contains the point, but keeps checking the remaining components and so, in my own, version I decided to do it. My implementation takes as input the matrix describing the complement and derives from it the set of balls and halfplanes; starting from such object and from their methods it checks whether or not the point is in the α-hull.

The computations are perfomed in the `inahullRcpp` and `inahullPoint` functions in file `inahull_Rcpp.cpp`. Instructions on how to use this function can be found in the .pdf manual in the repository or by typing `help(inahull)` in R's command line.

4 Tests

I tested both the validity of result for the different functions I wrote and the speed on which they are obtained with respect the package `alphahull`. The script I used sample randomly the set of points used as a test and then compare the results of the two packages or confront the speed of the functions. Scripts of such tests are saved in the folder `test_scripts` of the repository.

4.1 Validity of results

Since the packages work with floating point numbers I decided to compare results by looking at those quantities that can be compared exactly when possibile and to used tolerance comparisons while it wasn't the case. All the validity tests randomly sample the number of sites `n` and their coordinates and next call the function for which they want to compare the results. Such procedure is done `n.test` times.

While checking the results about the Voronoi diagram and the α-shape, I simply checked that both matrices mesh have the same number of rows, the same number of finite edges and infinite edges is the same and that they connect the same points (denoted by their indices).

For the confrontation of the `complement` functions, however, I had to relate on a comparison with tolerance on the quantities that defines the balls and the halfplanes; such tolerance `eps` can be modified by the user, I tested the code and found out that, up to a tolerance of $1e-11$ the complement sets are constituted by the same objects.

Lastly, for the function `ahull` I wasn't able to perform such comparison since, as already mentioned in subsubsection 3.2.5, the structure of my matrix `arc` is widely different from the one returned by the `alphahull` package due to the fact that in my implementation the boundary is described by circumference arcs with radius $\geqslant \alpha$ while in the original implementation such matrix is formed only by arcs of radius α.

Additionally to the scripts in folder `test_scripts` I provide here a graphical comparison (see figure 8) based on a randomized set of 300 with an annulus, code can be seen below. Relative error of α-shape and α-hull lengths are respectively 0 and $3.476673e-16$.

```
set.seed(3)
n = 300
theta = runif(n,0,2*pi)
r = sqrt(runif(n,0.25^2,0.5^2))
x = 0.5+r*cos(theta)
y = 0.5+r*sin(theta)
vorcpp = RcppAlphahull::delvor(x,y)
vorR =alphahull::delvor(x,y)
alpha = 0.1
asR = alphahull::ashape(vorR, alpha = alpha)
ascpp = RcppAlphahull::ashape(vorcpp, alpha = alpha)
ahR = alphahull::ahull(vorR, alpha = alpha)
ahcpp = RcppAlphahull::ahull(vorcpp, alpha = alpha)

# ashape length relative error
asR$length
ascpp$length
abs(asR$length-ascpp$length)/asR$length

# ahull length relative error
ahR$length
ahcpp$length
abs(ahR$length-ahcpp$length)/ahR$length

par( mfrow = c(1,2))
plot(vorcpp, wlines = "vor", pch = 19,
         col = c("blue","black","black","black"),
         main = "RcppAlphahull", xlab = "x", ylab = "y",
         cex.lab = 1.8, asp = 1)
plot(vorR, wlines = "vor", pch = 19,
         col = c("blue","black","black","black"),
         main = "alphahull", xlab = "x", ylab = "y",
         cex.lab = 1.8, asp = 1)

par( mfrow = c(1,2) )
plot(ascpp, col = c("red", "blue"), asp = 1, pch = 19,
         main = "RcppAlphahull", xlab = "x", ylab = "y",
         cex.lab = 1.8)
plot(asR, col = c("red", "blue"), asp = 1, pch = 19,
         main = "alphahull", xlab = "x", ylab = "y",
         cex.lab = 1.8)
```

	RcppAlphahull				alphahull			
n	delvor	ashape	complement	ahull	delvor	ashape	complement	ahull
10	0.001	0.000	0.000	0.001	0.002	0.003	0.002	0.012
100	0.002	0.001	0.001	0.001	0.003	0.006	0.008	0.035
1000	0.011	0.001	0.003	0.020	0.017	0.035	0.102	0.770
10000	0.130	0.010	0.034	0.079	0.508	0.337	1.079	2.382

Table 1: speed comparison in seconds of the two packages for the functions `delvor`, `ashape`, `complement` and `ahull`. The number of sites n here spans powers of 10, the script for this test is `speed_table.R`.

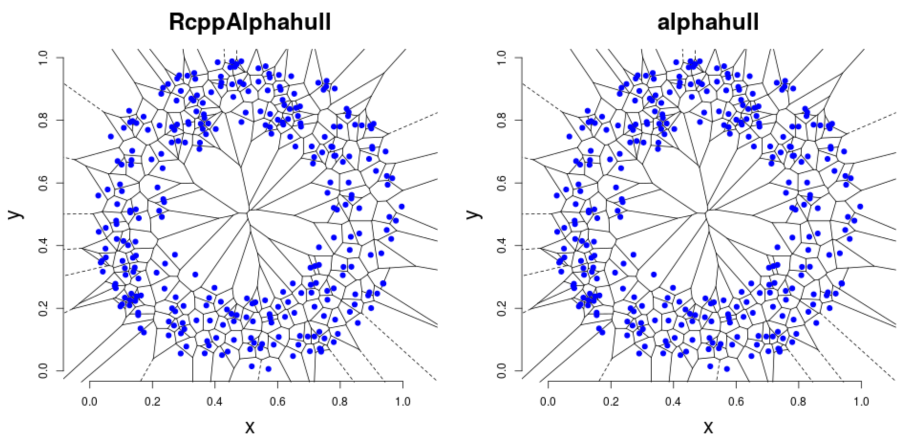
```r
par( mfrow = c(1,2) )
plot(ahcpp, col = c("red", "blue", "blue"), asp = 1,
         pch = 19, cex.lab = 1.8,
         main = "RcppAlphahull", xlab = "x", ylab = "y")
plot(ahR, col = c("red", "blue", "blue"), asp = 1,
         pch = 19, cex.lab = 1.8,
         main = "alphahull", xlab = "x", ylab = "y")
```

4.2 Speed tests

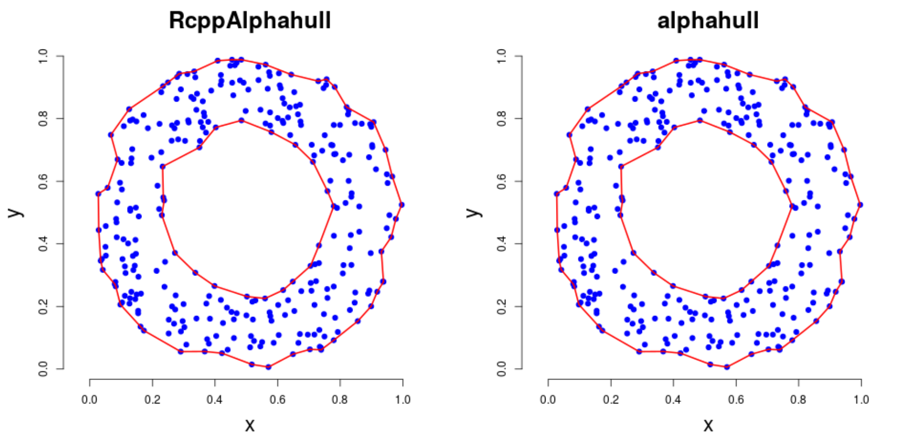
Since the goal of the project was to improve the speed of the computation of the geometrical structures introduced in section 2, I also created some scripts to test the performances of my package `RcppAlphahull` against the original one, `alphahull`. In each of them the code randomly samples the sites for different values of `n`, as well as the value of parameter α for those functions which require it, and then compares the time that the two packages spend in order to get the results. The comparison is used by employing the function `benchmark` package `rbenchmark` and looking at the `user.self` entry of the returned matrix that gives the CPU time spent by the R processes in seconds. Results can be seen in table 1, table 2 and figure 9.

4.3 A practical example

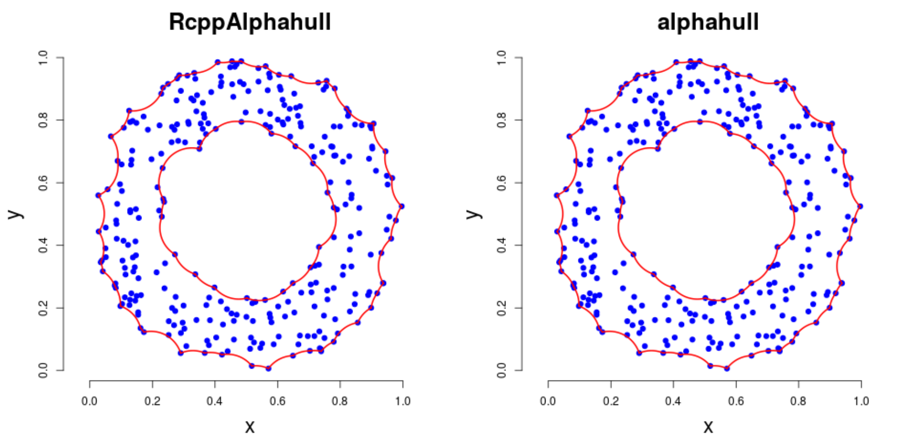
As last example, I provide an analysis performed on an egg printed with 3D printing. After the building process, the eggshell is scanned with tomography techniques and the resulting points of the surface are divided randomly into 10 sets. Each section is then projected on a 2-dimensional regression plane with PCA (Principal Component Analysis), in this way we have 10 sets of points in $\mathbb{R}^2$ as shown in figure 10. Finally on the approximating sets we compute the α-convex hulls and the α-shapes: goal of the analysis is to find and describe imperfections, such as holes or material excesses, in the eggshell. Graphical results of the computations are shown in figures 11 and 12. In table 3 I report the number of points, the α-shapes' length and the α-convex hulls' length for each section; the value of the parameter used is $\alpha = 0.85$.



(a) graphical comparison of functions `delvor`



(b) graphical comparison of functions `ashape`



(c) graphical comparison of functions `ahull`

Figure 8: Graphical comparison of the results of the two packages on the same set of points, a randomized annulus in $[0;1] \times [0;1]$. α-shape and α-hull are computed for the value $\alpha = 0.1$.

	`RcppAlphahull`				`alphahull`			
n	delvor	ashape	complement	ahull	delvor	ashape	complement	ahull
100	0.002	0.000	0.002	0.002	0.003	0.006	0.012	0.037
200	0.003	0.001	0.002	0.004	0.005	0.009	0.023	0.085
300	0.004	0.000	0.003	0.008	0.006	0.012	0.034	0.139
400	0.005	0.001	0.003	0.005	0.008	0.015	0.044	0.116
500	0.006	0.000	0.004	0.014	0.009	0.022	0.052	0.312
600	0.006	0.001	0.005	0.006	0.010	0.021	0.062	0.199
700	0.008	0.001	0.005	0.010	0.013	0.025	0.075	0.224
800	0.009	0.001	0.006	0.019	0.014	0.030	0.081	0.407
900	0.010	0.002	0.007	0.014	0.015	0.030	0.099	0.278
1000	0.011	0.001	0.003	0.016	0.018	0.035	0.111	0.413

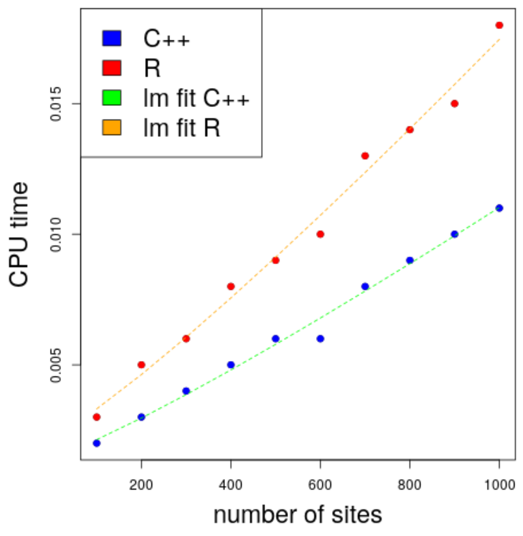
Table 2: speed comparison in seconds of the two packages for the functions `delvor`, `ashape`, `complement` and `ahull` as reported in figure 9. n denotes the number of sites randomly sampled for each test with seed set at 3 (with command `set.seed(3)`). The script for this tests are in folder `test_scripts`.

Section	n	α-shape length	α-hull length
1	880	304.27043	321.53047
2	1014	311.46888	333.09387
3	976	298.92926	312.78471
4	442	163.02208	169.69186
5	892	278.87189	293.74374
6	1714	462.85202	496.78368
7	1296	403.70531	433.54239
8	726	227.09480	244.81235
9	1062	361.92424	385.30071
10	1646	424.34720	451.81903

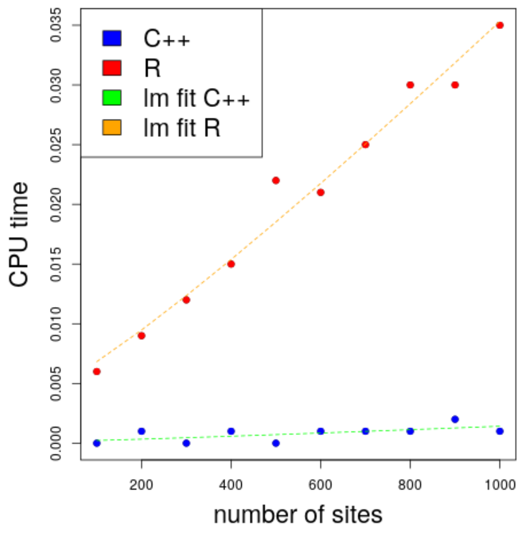
Table 3: main quantities of the eggshell analysis, for each section the number of points, the α-shape and the α-hull lengths are reported.

	`RcppAlphahull`	`alphahull`
ashape	0.504	1.632
ahull	3.782	76.397
total	4.286	78.029

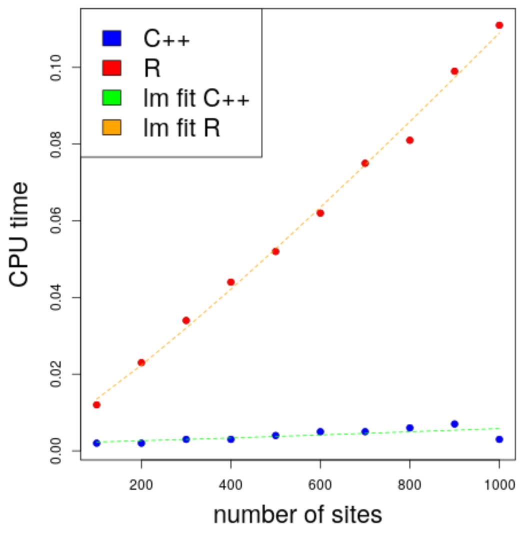
Table 4: eggshell analysis speed comparison between `alphahull` and `RcppAlphahull`. Each row report the time spent in seconds by the package to retrieve the respective object for all the ten sections.



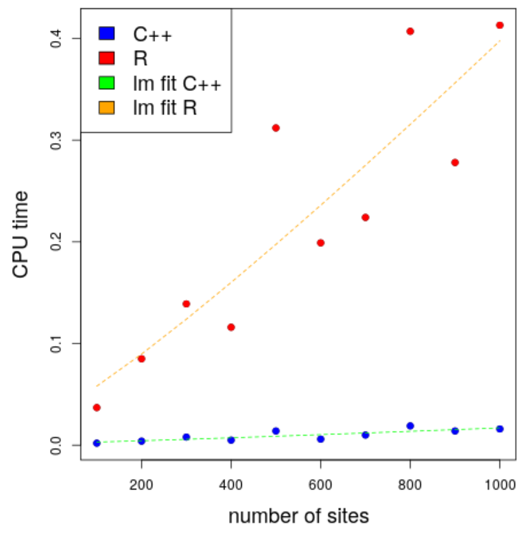
(a) performances of functions `delvor`



(b) performances of functions `ashape`



(c) performances of functions `complement`



(d) performances of functions `ahull`

Figure 9: speed comparison in seconds of the two packages for the functions `delvor`, `ashape`, `complement` and `ahull`. n denotes the number of sites randomly sampled for each test with seed set at 3 (with command `set.seed(3)`). The script for this test are in folder `test_scripts` of my repository. The blue dots refers to `RcppAlphahull`, the reds to `alphahull`. Dotted lines represent a linear model fit performed with R command `lm` using a formula of the form `Time` $\sim$ `n.nodes*log(n.nodes)`. Detailed values are in table 2.

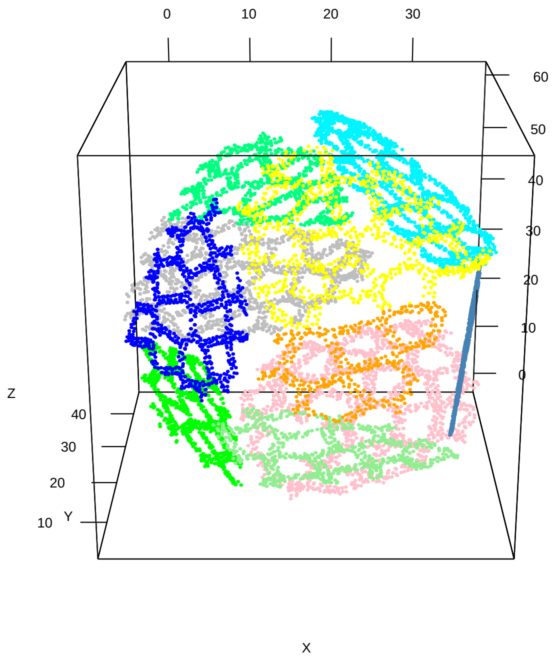


Figure 10: eggshell's section projected on the approximation planes. Sections are selected randomly and then approximated via a regression plane.

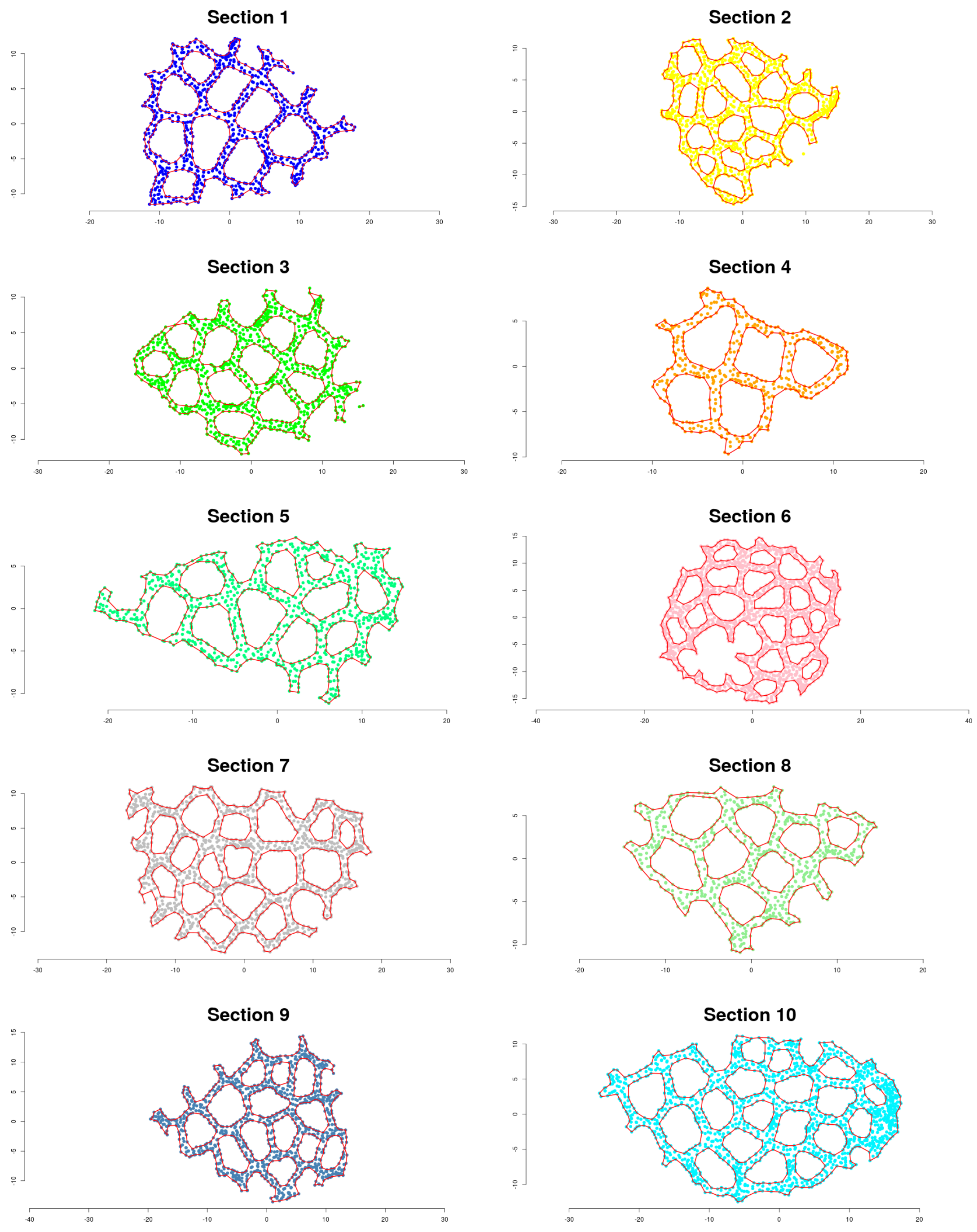


Figure 11: α-shapes of the eggshell sections for the value $\alpha = 0.85$.

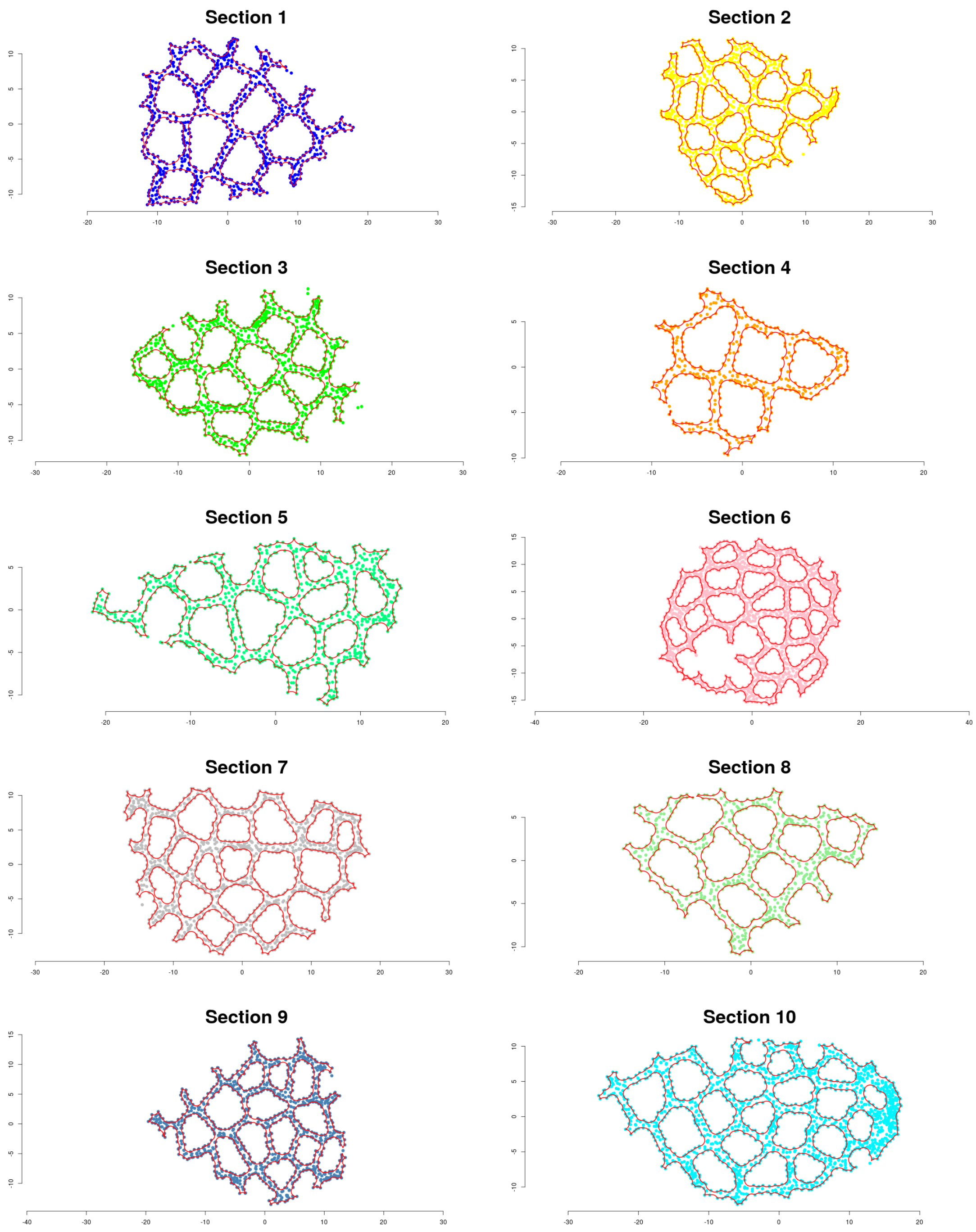


Figure 12: α-convex hulls of the eggshell sections for the value $\alpha = 0.85$.

5 Computational environment

The develompent of package `RcppAlphahull` and all of the reported tests, tables and plots, were carried out on an Acer Extensa5230E mounting Ubuntu 18.04.3, with 3GB RAM and an Intel Celeron 900 processor with 1 core at 2.20 GHz, and in within an R framework: R version 3.4.4 and RStudio 1.2.1335. The compiler used is gcc version 7.4.0.

6 Acknowledgement

I'd like to thank Professor Alessandra Menafoglio for her help in defining such project and testing its installation and functionalities.

7 Bibliography

References

[1] Schneider, R. 1988 *'Random Approximation of Convex Sets'*, Journal of Microscopy, vol. 151, pp. 211-227.

[2] Edelsbrunner, H., Kirkpatrick, D. G., Seidel, R. 1983 *'On the Shape of a Set of Points in the Plane'*, IEEE Transactions on Information Theory, vol. 29, no. 4, pp. 551-559.

[3] Rodriguez-Casal, A. *'Set Estimation under Convexity Type Assumptions'*, Annales de l'I.H.P.- Probabilits & Statistiques, no. 43, pp. 763-774.

[4] Serra, J. 1984, *'Image Analysis and Mathematical Morphology'*, Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London. ISBN 0-12-637240-3. English version revised by Noel Cressie.

[5] Pateiro-Lpez, B., Rodriguez-Casal, A. 2019 *'alphahull: Generalization of the Convex Hull of a Sample of Points in the Plane'*, R package version 2.2, `https://CRAN.R-project.org/package=alphahull`.

[6] Pateiro-Lpez, B., Rodriguez-Casal, A. 2010 *'Generalizing the Convex Hull of a Sample: The R Package alphahull'*, Journal of statistical software, vol. 34, no. 5.

[7] Fortran code by R. J. Renka. R functions by Albrecht Gebhardt. With contributions from Stephen Eglen <stephen@anc.ed.ac.uk>, Sergei Zuyev and Denis White 2016, *'tripack: Triangulation of Irregularly Spaced Data'*, R package version 1.3-8. `https://CRAN.R-project.org/package=tripack`

[8] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O. 2000, *'Computational Geometry (2nd revised ed.)'*, Springer-Verlag, ISBN 3-540-65620-0 Section 7.2: Computing the Voronoi Diagram: pp.151160.

[9] Eddelbuettel, D., Francois, R. 2011, *'Rcpp: Seamless R and C++ Integration'*, Journal of Statistical Software, vol. 40, no. 8, pp. 1-18. URL.

[10] Wickham, H., Hester, J., Chang, W. 2019, *'devtools: Tools to Make Developing R Packages Easier'* R package version 2.1.0, `https://CRAN.R-project.org/package=devtools`.

[11] Wickham, H., Danenberg, P., Eugster, M. 2019, *'roxygen2: In-Line Documentation for R'*, R package version 6.1.1.9000, `https://github.com/klutometis/roxygen`.

[12] MyGAL: `https://github.com/pvigier/MyGAL`.