
Delaunay triangulation

Joseph LIARD <joseph.liard@drawsvg.org>

Release 1.0

2026 October

Abstract

This article describes how to compute Delaunay triangulations using algorithms ranging from the simplest to the most complex and high-performance ones.

The final method presented highlights an innovative approach suitable for large datasets.

The algorithms presented are evaluated using several datasets derived from topographic surveys.

The evaluations rely on metrics to assess algorithm complexity, computational load, and memory usage.

This article is available online in HTML format.

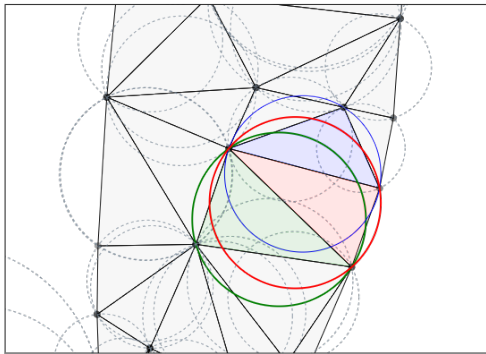
Table of Contents

Delaunay triangulation	1
Topological model of triangulations	2
Inserting an interior vertex	3
Inserting a vertex on a segment	3
Shift the adjacent side of two triangles	4
Delaunay criterion for two adjacent triangles	5
Base max and height of a triangle	5
The relative axis tool	5
Status of a point	7
Base criterion for two adjacent triangles	7
OOP implementation of the topological model	8
Evaluation metrics	8
Sample test dataset	9
Bowyer–Watson algorithm the reference	11
Flip algorithm	14
Optimization via tree-based sorting	18
Algorithm with convex hull	21
Convex ring algorithm	25
Evaluation with other datasets	30
Assessment of method complexity	33
Evaluating the strengths and weaknesses of the methods	37
General conclusion	40
References	40

Delaunay triangulation

Boris Delaunay was a Russian mathematician born in 1890, well known for Delaunay triangulation (1934).

The Delaunay triangulation of a set of points in the plane subdivides the space into triangles whose circumcircles contain none of the points.



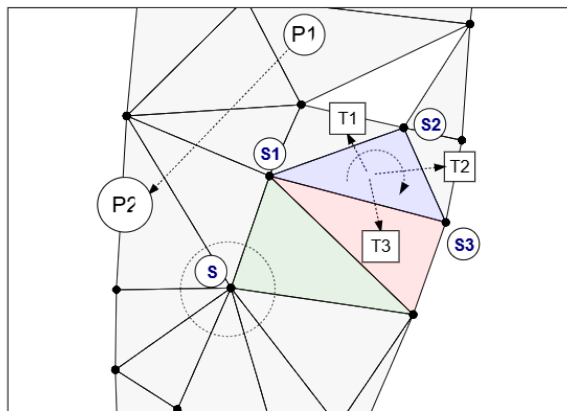
This method maximizes the size of the smallest angle among all the triangles and tends to avoid skinny triangles.

This method is used in numerous fields of application, notably surveying, to calculate the contour lines of a terrain based on a network of points with measured elevations.

The illustration highlights three adjacent triangles (red, green, blue) that satisfy the Delaunay criterion, along with their circumcircles.

Topological model of triangulations

The topological model of a triangulation allows for navigation from triangle to triangle and the performance of operations on the triangles while maintaining the consistency of the network.



Each triangle in the mesh references its three vertices (S1, S2, S3) as well as its adjacent triangles.

Up to three triangles (T1, T2, T3):

- T1 adjacent to edge S1-S2.
- T2 to edge S2-S3.
- and finally T3 to edge S3-S1.

In principle, all triangles are described in the same direction of rotation. For convenience, clockwise, so as to obtain a positive area:

$$\text{area}(S1, S2, S3) = (x2-x1)*(y1+y2)/2 + (x3-x2)*(y2+y3)/2 + (x1-x3)*(y3+y1)/2$$

This also simplifies the (strict) inclusion test for a point $p(x,y)$ within the triangle (S1,S2,S3) by determining the three areas (S1,S2,p), (S2,S3,p), and (S3,S1,p), all of which must be strictly positive:

$$\text{area}(S1, S2, p) > 0 \text{ and } \text{area}(S2, S3, p) > 0 \text{ and } \text{area}(S3, S1, p) > 0$$

Navigating the network allows you to determine the triangles **intersected by a segment (p1,p2)** or to traverse the triangles **around the same vertex (s)**.

The topological model of the triangulation supports several operations, such as:

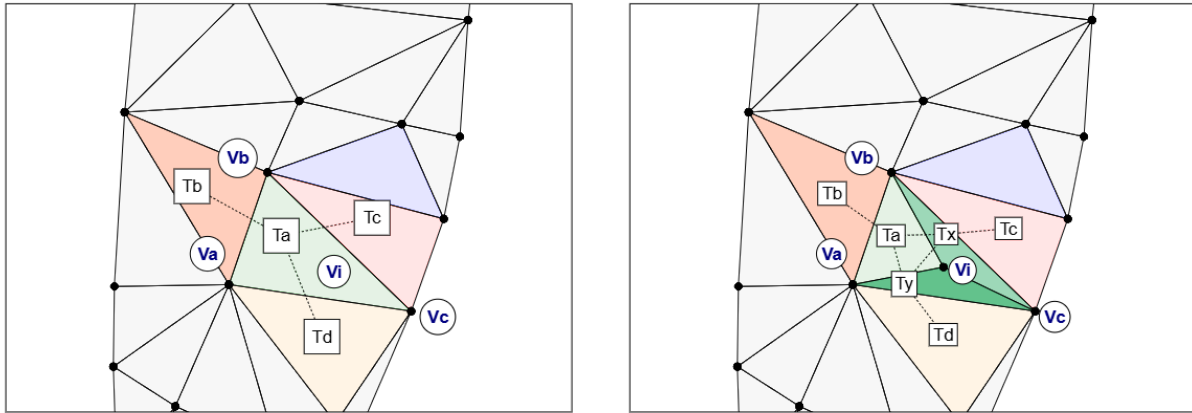
- Inserting an interior vertex

- Inserting a vertex on a segment
- Shift the adjacent side of two triangles

Inserting an interior vertex

This operation completes the triangulation by subdividing a triangle into three, using a new vertex strictly inside the triangle.

For convenience, this operation is referred to as "*split triangle*" in the remainder of this document, denoting the division of a triangle into three by inserting a point inside.



Before inserting vertex V_i into triangle T_a

T.	S1	S2	S3	T1	T2	T3
T_a	V_a	V_b	V_c	T_b	T_c	T_d

After inserting vertex V_i into triangle T_a

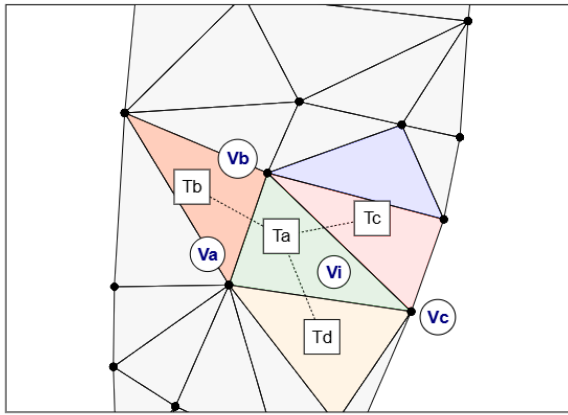
T.	S1	S2	S3	T1	T2	T3
T_a	V_a	V_b	V_i	T_b	T_x	T_y
T_x	V_i	V_b	V_c	T_a	T_c	T_y
T_y	V_i	V_c	V_a	T_x	T_d	T_a

The operation modifies triangle T_a and inserts the two triangles (T_x, T_y) .

Inserting a vertex on a segment

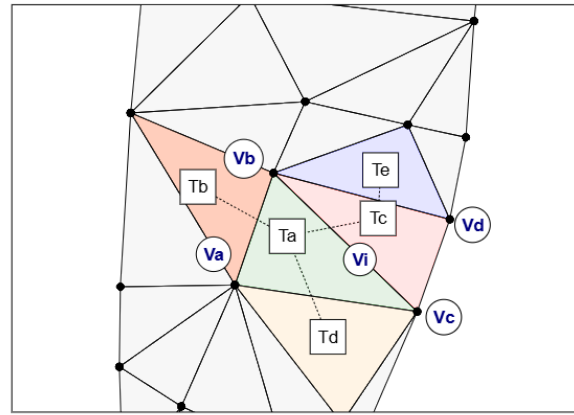
This operation completes the triangulation by inserting a vertex on the edge of a triangle, thereby subdividing that triangle into two, as well as the adjacent triangle sharing that edge, if one is defined.

For convenience, this operation is referred to as "*split segment*" in the remainder of this document, denoting the division of two triangles into four by inserting a point on the shared segment.



Before inserting vertex V_i onto the segment (V_b, V_c)

T.	S1	S2	S3	T1	T2	T3
Ta	Va	Vb	Vc	Tb	Tc	Td
Tc	Vc	Vb	Vd	Ta	Te	



After inserting vertex V_i onto the segment (V_b, V_c)

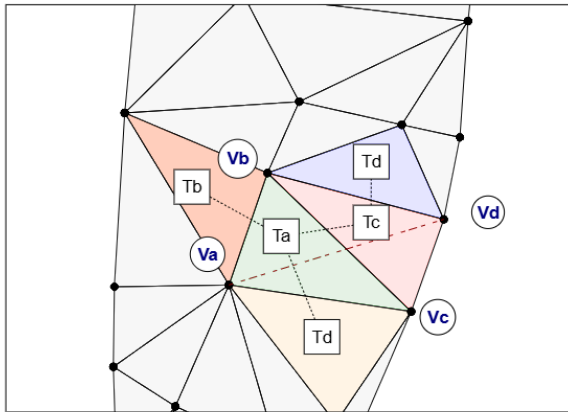
T.	S1	S2	S3	T1	T2	T3
Ta	Va	Vb	V_i	Tb	Tc	Ty
Tc	V_i	Vb	Vd	Ta	Te	Tx
Tx	V_i	Vd	Vc	Tc		Ty
Ty	V_i	Vc	Va	Tx	Td	Ta

The operation modifies triangles (T_a, T_c) and inserts the two triangles (T_x, T_y) .

Shift the adjacent side of two triangles

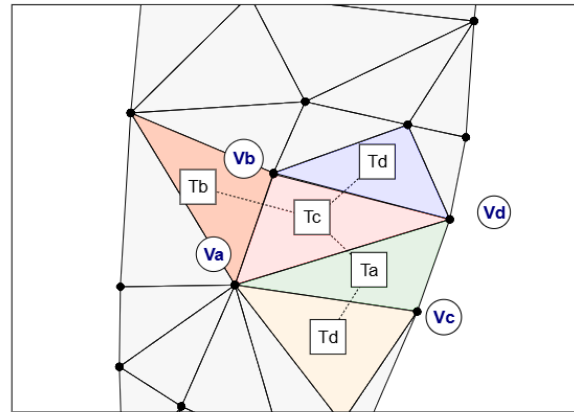
This operation transforms two adjacent triangles by flipping the vertices of their shared side onto the opposite vertices.

Transformation of triangles (T_a, T_c) by flipping the common edge (V_b, V_c) to (V_a, V_d) :



Before transformation

T.	S1	S2	S3	T1	T2	T3
Ta	Va	Vb	Vc	Tb	Tc	Td
Tc	Vc	Vb	Vd	Ta	Td	

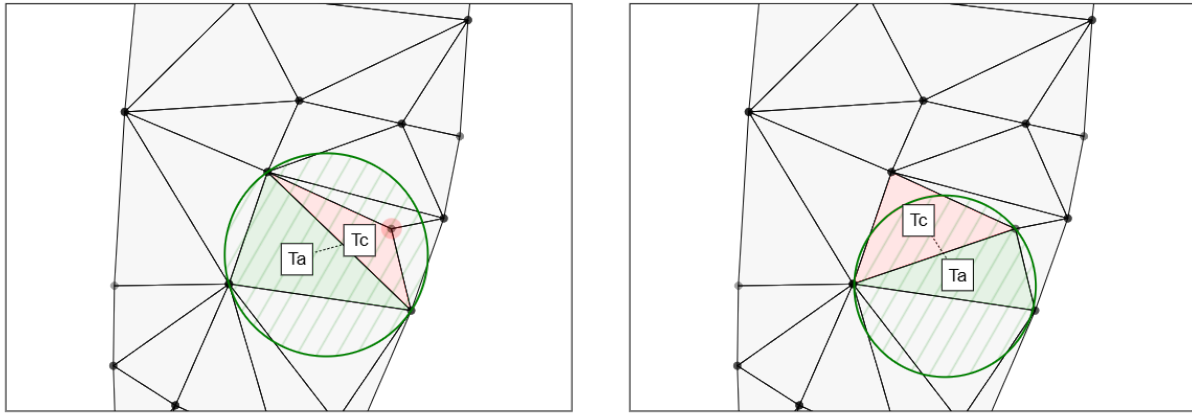


After transformation

T.	S1	S2	S3	T1	T2	T3
Ta	Va	Vd	Vc	Tc		Td
Tc	Va	Vb	Vd	Tb	Td	Ta

Delaunay criterion for two adjacent triangles

Two adjacent triangles do not satisfy the Delaunay condition if one of the two vertices opposite the shared edge lies within the circumcircle of the triangle formed by the other vertex.

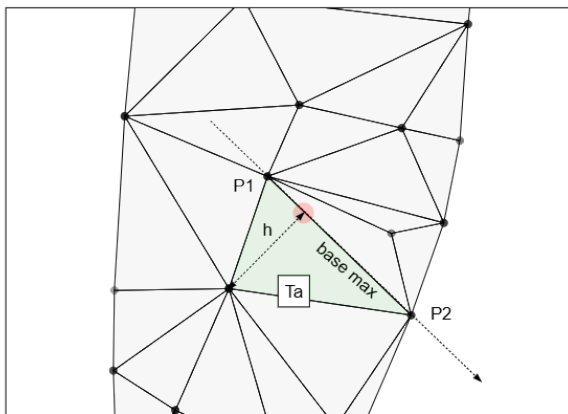


The vertex of (T_c) opposite the common side lies within the circumcircle of (T_a). After the shift transformation the two triangles meet the criterion.

The two triangles (T_a , T_c) do not satisfy the Delaunay criterion.

Base max and height of a triangle

The maximum base of a triangle is its longest side, with the opposite vertex situated orthogonally to it.

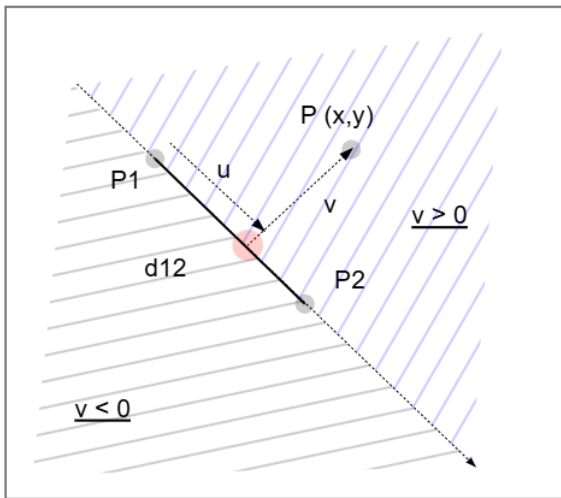


Measure the height from the longest base to the opposite vertex, and finally, the height-to-base-length ratio.

To determine the maximum base, the relative axis measurement tool ($P1$, $P2$) is used.

The relative axis tool

The relative axis is to geometric calculation what the ruler and set square are to technical drawing. That is, an indispensable tool.

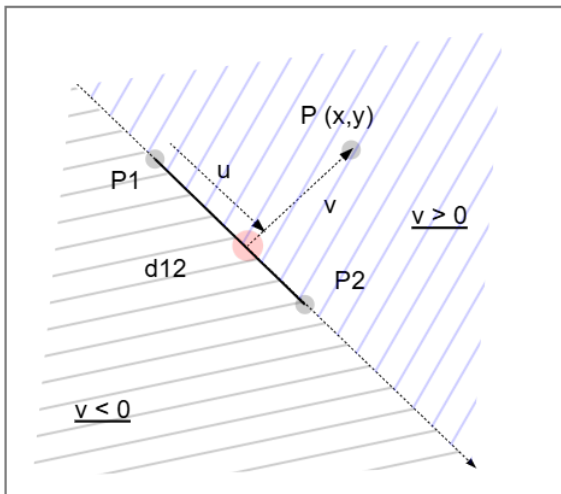


The relative axis, defined by two points (P_1 , P_2), transforms absolute coordinates (x , y) into relative coordinates (u , v).

The value t , equal to u divided by the distance d_{12} between P_1 and P_2 , lies between 0 and 1 when point P projects onto the segment (P_1 , P_2).

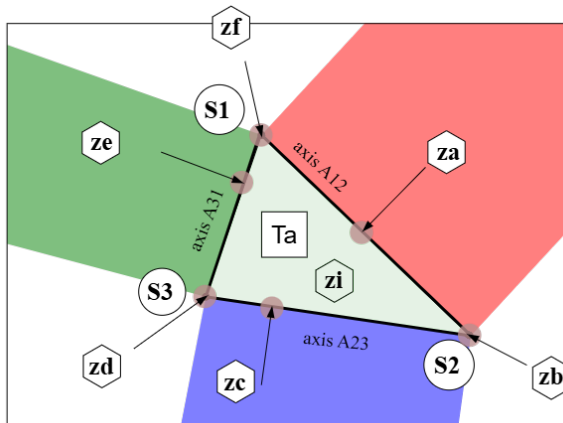
The axis divides the space into two sectors (positive v , negative v).

Examples of using the relative axis to calculate a triangle's circumcircle and its distance to a point.



The circumcenter of triangle T_a is determined by the intersection of the perpendicular bisectors (A_{12} , A_{31}) of sides (S_1 , S_2) and (S_3 , S_1).

Calculation of the distance from a point to the triangle, the axes of the 3 sides divide the space into 7 zones:

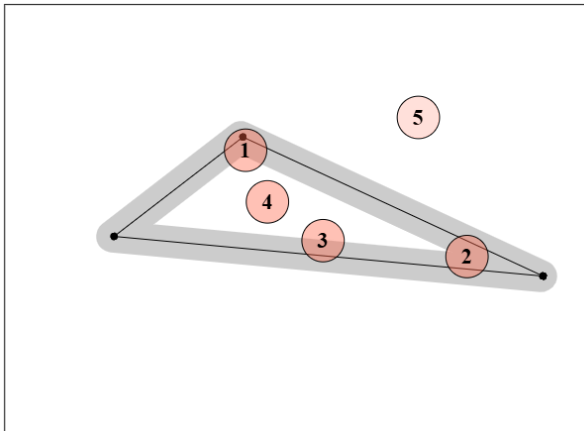


- **Za:** the distance from a point to the triangle is equal to its coordinate relative to axis A12.
- **Zb:** the distance from a point to the triangle is equal to its distance to vertex S2.
- **Zc:** the distance from a point to the triangle is equal to its coordinate relative to axis A23.
- **Zd:** the distance from a point to the triangle is equal to its distance to vertex S3.
- **Ze:** the distance from a point to the triangle is equal to its coordinate relative to axis A31.
- **Zf:** the distance from a point to the triangle is equal to its distance to vertex S1.
- **Zi:** the point is inside the triangle; its distance is equal to 0.

Status of a point

The triangulation incorporates the computational **precision** required to handle **edge cases**, such as point-to-point or point-to-segment proximity.

A point's position relative to a triangle is characterized by a status.



The status is an enumeration used to distinguish between cases, the radius represents the calculation precision of the triangulation:

1. Point close to a vertex (edge case "*near vertex*").
2. Point within the angular sector of a vertex defined by the precision (edge case "*near angular sector*").
3. Point near a segment of the triangle (edge case "*near segment*").
4. Point inside the triangle.
5. Point outside the triangle.

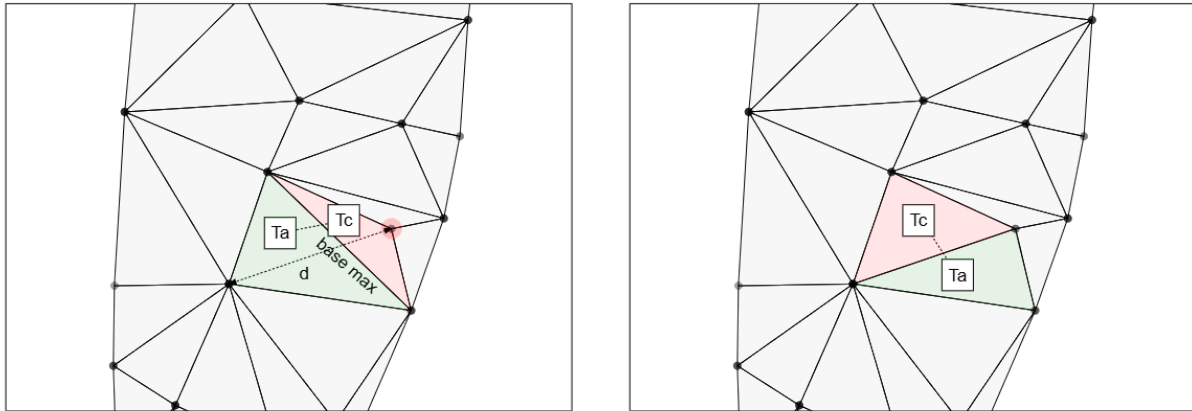
Determining point status significantly impacts triangulation calculation methods, particularly when handling edge cases in operations such as inserting a point inside a triangle or onto a segment.

In particular, points with "*near vertex [7]*" status are excluded from the triangulation, listed, and flagged as a dataset anomaly.

Base criterion for two adjacent triangles

Two triangles sharing a common base max, with the distance between their opposite vertices being less than the length of the common base, are candidates for transformation by swapping the common base with the edge connecting their opposite vertices.

This criterion makes it possible to improve triangle quality without calculating circumcircles. It can be used prior to Delaunay criteria.



OOP implementation of the topological model

An object-oriented implementation (OOP) of the triangulation's topological model establishes a logical foundation that enables the development of various methods.

The kernel consists of three classes of objects:

- Vertex

It may be identical to the dataset's point class, or constitute an enriched class containing a link to a triangle built on that vertex, serving as an entry point into the triangulation.

Provides its (x, y) coordinates.

- Triangle

Class holding links to its 3 vertices and its one to three adjacent triangles.

Its public methods are read-only for the object.

- Triangulation

Class that holds the triangles and contains the logic for creating and transforming them.

All the methods described in this document are based on this model.

Evaluation metrics

To compare methods, several indicators are measured:

- **Duration** (Ms), the execution time (indicative) in milliseconds

This metric is provided for informational purposes; due to the execution platform (Desktop Intel Core i5 32Go, Windows 11/Java), it may vary from one execution to another.

- **Memory** (Kb), the amount of memory in kilobytes.

This measure is calculated by summing the sizes of the objects' data members.

- **Classes**, the number of OOP classes.

Only application classes are counted. Technical classes, such as lists, are not included in the count.

- **Objects**, the number of allocated objects.
- **Triangle**, the number of created triangles.
- **Conformity** (C. %), the Delaunay conformity of the triangulation in percentage after point insertion step.
- **Distance** the number of distance calculations in **thousands**.

Only distances calculated between points.

- **Axis** tool, the number of axis calculations in **thousands**.
- **Point** status (P. status), the number of point status calculations in **thousands**.
- **Circle**, the number of circumcircle calculations in **thousands**
- **Insert**, the number of triangle point insertion
- **Split** segment, the number of segment point insertion
- **Shift**, the number of triangle transformations resulting from the application of the Delaunay criterion and the base-max criterion.
- **Scan** (Sc.), the number of full scans of the triangulation triangles.
- **Near vertex** status (N. v, the number of points with "near vertex [7]" status excluded from the triangulation.
- **Near segment** status (N. s), the number of cases with "near segment [7]" or "near angular sector [7]" status.

This two last metrics characterizes the quality of the dataset.

- **Op** ($\sum$ op), the total number of geometric operations counted (distance [9], axis [9], P. status [9], circle [9]) in **thousands**.

Axis and circle: counts only the number of constructed objects and encompasses all method calls on those objects.

Distance: only distances calculated between points; metrics distinct from axis and circle.

- $\sum$ **pt** the total number of geometric operations per point.
- **B. p** the amount of memory in bytes per point.
- **Ratio** calculated as the number of operations [9] per point of the reference method *DtRef* divided by that of the method under consideration.

Provides a comparison criterion that is inversely related to the method's computational load.

Sample test dataset

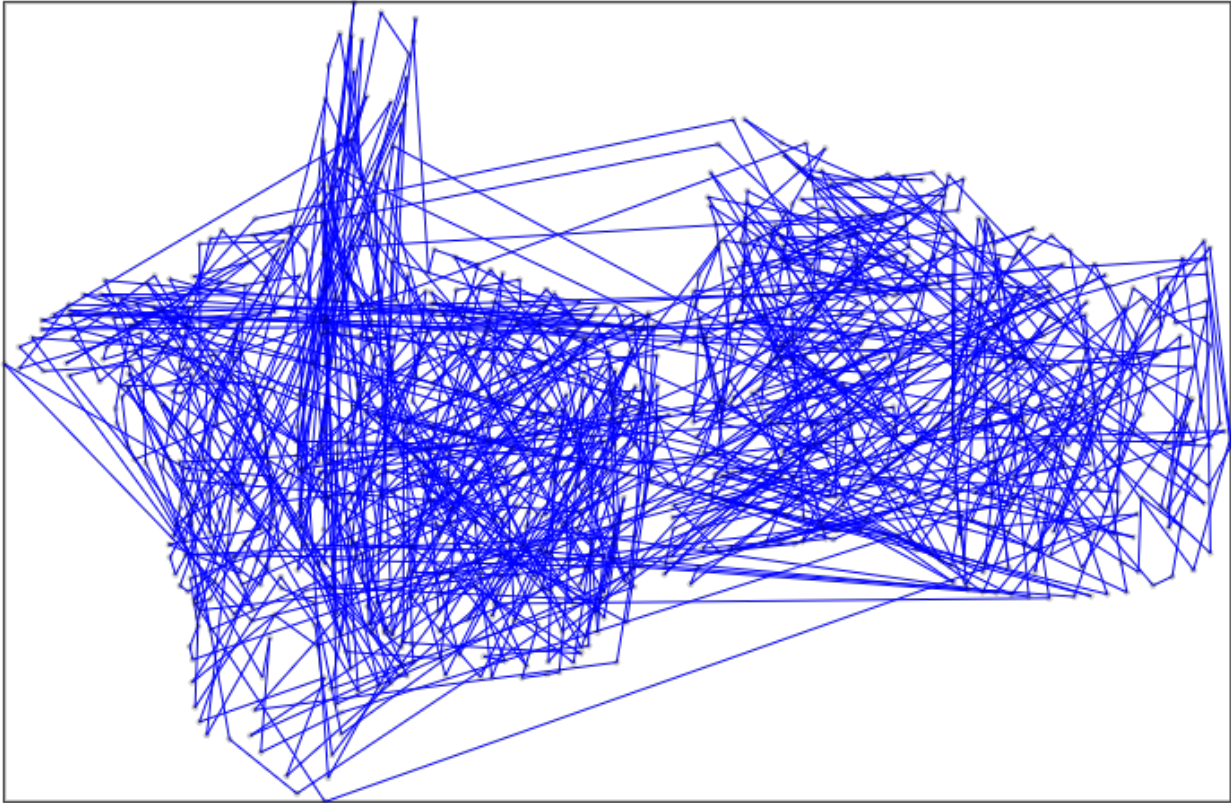
The calculation methods presented below are illustrated using a sample of real points. This consists of a topographic survey of leveling points.

The sample consists of **981** points covering an area of **9,700 x 6,300** meters. The calculation precision is set to **1** centimeter.

The non-convex perimeter is irregular because it borders rivers.



The distribution of the points is not random as shown in the illustration below, but proceeds step-by-step following the operators' movements in the field.



This property is important for the subsequent calculations.

Bowyer–Watson algorithm the reference

The Bowyer–Watson algorithm is an iterative algorithm.

It proceeds by adding points, one at a time, to a valid Delaunay triangulation of the subset of points. After each insertion, all triangles whose circumcircle contains the new point are removed, leaving a convex polygonal hole that is then re-triangulated using the new point.

The triangulation is initialized with a **fictitious** "super-triangle" enclosing all the points by adding specific points, which can be constructed as follows:

```
Determine the bounding box of the points (center (xc, yc), width, height)
Set the radius  $r = \text{Max}(\text{width}, \text{height}) / 2$ , this value can be increased to move away from the limit points.
Set  $p1 = (xc, yc + 2*r)$ 
Set  $p2 = (xc + 3*r, yc - r)$ 
Set  $p3 = (xc - 3*r, yc - r)$ 
```

After creating the triangulation, the triangles located along the edges of the fictitious envelope must be removed.

Finally, the triangulation boundary may extend beyond the actual perimeter of the point set, resulting in triangles with abnormal lengths (or ratios). In such cases, the boundary must be reduced by removing the abnormal triangles along the edges.

The Bowyer–Watson algorithm can be summarized as follows:

- 1/ Create the "super triangle"
- 2/ Insert the scatter points

```

For each point (p):
- find the triangle (t) containing point (p)
- remove the triangles adjacent to (t) whose circumcircle contains the new point (p)
- triangulate the polygonal hole using the new point (p)
End loop
3/ Remove the triangles at the edge of the fictitious envelope
4/ Remove abnormal triangles on the boundary

```

Simplification of the algorithm using the DtRef method

The Bowyer–Watson algorithm can be simplified by leveraging the topological model of the triangulation and its navigation [2] functions, such as traversing the triangles sharing a common vertex.

To simplify the algorithm, we replace the removal of triangles and the triangulation of the hole with the transformation of non-conforming triangles around [2] the insertion point, which leads to the same result.

The modified algorithm only creates triangles, thereby avoiding memory fragmentation issues and calls to the garbage collector.

This method is referred to as **DtRef** in the remainder of the document.

The algorithm of the DtRef method is :

```

1/ Create the "super triangle"
2/ Insert the scatter points
  For each point (p):
    - find the triangle (t) containing point (p)
    - split triangle (t) into three at point (p)
    // Keep Delaunay's conformity
    - As long as there is a non-conforming triangle (ts) around vertex (p):
      - flip the triangle (ts) and its relevant adjacent triangle
    - End of loop (ts)
  End of loop (p)
3/ Remove the triangles at the edge of the fictitious envelope
4/ Remove abnormal triangles on the boundary

```

Insert process including edge cases

Edge cases (see point status) arising from calculation precision must be taken into account, which somewhat complicates the algorithm.

The triangle search process iterates through all triangles and, for each one, determines the point's status to handle edge cases:

- "*near vertex [7]*" case: triangle ignored and point excluded from the triangulation calculation.
- "*near segment [7]*" and "*near angular sector [7]*" cases: triangle ignored and point set aside for later processing.
- "*outside*" case: triangle ignored.
- "*inside [7]*" case: triangle retained.

The detailed insertion [12] step of the DtRef method becomes:

```

2/ Insert the scatter points
  For each point (p):
    For each triangle (t)
      - evaluate the state of point (p) for triangle(t):
        - case near vertex [7]:
          - exclude point (p)

```

```

- next point (p)
- case near segment [7] and near angular sector [7]:
- set point (p) aside
- next point (p)
- case "inside [7]":
- split triangle (t) into three at point (p)
- As long as there is a non-conforming triangle (ts) around vertex (p) :
  - flip the triangle (ts) and its relevant adjacent triangle
- End of loop (ts)
- next point (p)
End of loop (t)
End of loop (p)

```

"Near segment [7] and near angular sector [7]" points are set aside for processing after the insertion phase.

Processing them may result in:

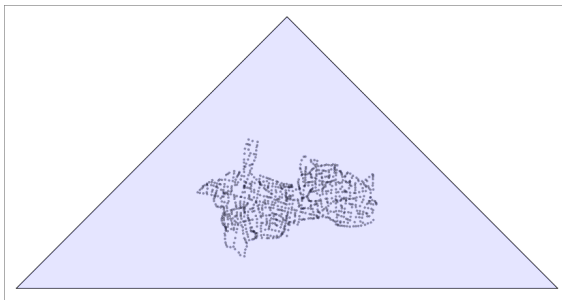
- Insertion onto a segment.
- Insertion into a triangle.
- Exclusion due to proximity to a vertex [7].

Step (2) is performed using nested loops with $O(n^2)$ complexity, due to the search for triangle (t), which sequentially iterates over the created triangles.

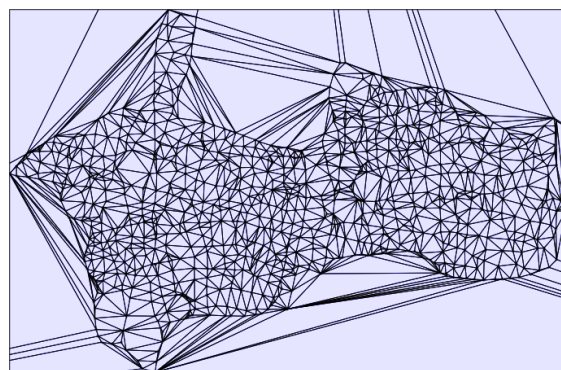
Sample processing

Illustration of each sample processing step of the *DtRef* method. The triangles conforming to the Delaunay triangulation are shaded in blue:

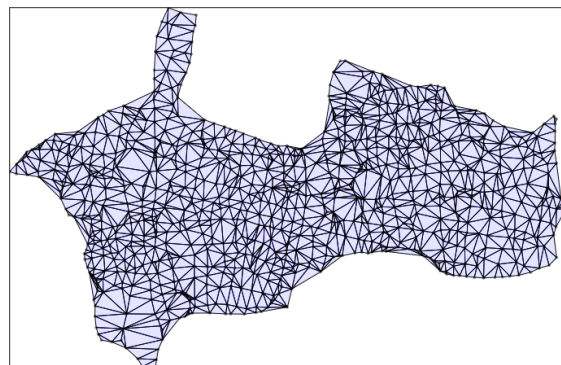
Step 1 Create the super triangle

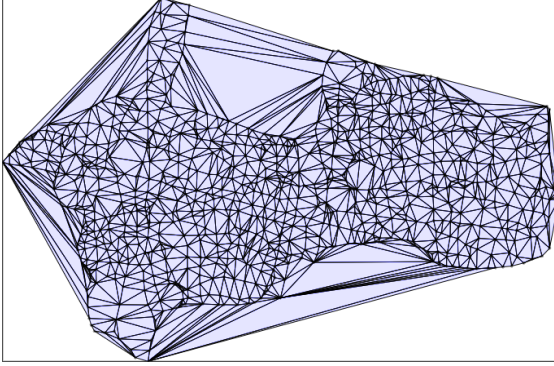


Step 2 Insert the scatter points, conformity is **100%**



Step 3 Remove the triangles at the edge of the fictitious envelope





The final step removes boundary triangles whose longest side exceeds a maximum length or whose ratio (height/base length) is below a threshold:

- **680** meters for our sample.
- **10%** in our case study.

Flip algorithm

The majority of Delaunay triangulation generation algorithms rely on the "Flip" principle that is:

- Transforming a triangulation through successive iterations on its non-conforming triangles **converges** toward conformity.

The algorithms boil down to a two-step construction:

- 1/ Create the triangulation
- 2/ Flip adjacent triangles that do not meet the Delaunay criterion

The DtFlipRef method

The "flip principle [14]" is applied to the *DtRef* method by eliminating the maintenance of Delaunay conformity during each point insertion, replacing it with iterations performed after all points have been inserted.

The method described below is referred to as **DtFlipRef** in the remainder of the document. It is a simplified version of the *DtRef* method with the Flip principle [14].

The most immediate method for creating the triangulation of a point set is to construct the "super triangle", and then to insert the points of the set into it.

Step (1) becomes:

- 1/ Create the triangulation:
 - 1.1 Create the super triangle
 - 1.2 Insert the scatter points

We conclude with the same steps (3 and 4) of the Bowyer–Watson algorithm [12] for removing unnecessary triangles.

The algorithm of the DtFlipRef method can be summarized as follows:

- 1/ Create the triangulation:
 - 1.1 Create the super triangle
 - 1.2 Insert the scatter points
- 2/ Flip adjacent triangles that do not meet the Delaunay criterion
- 3/ Remove the triangles at the edge of the fictitious envelope
- 4/ Remove abnormal triangles on the boundary

Point **insertion** step (1.2) is performed using nested loops with $O(n^2)$ complexity, due to the search for the insertion triangle, which sequentially traverses the created triangles:

- 1.2 Insert the scatter points
 - For each point (p):
 - find the triangle (t) containing point (p)
 - split triangle (t) into three at point (p)

After insertion, the percentage of triangles satisfying the Delaunay criterion is recorded in order to compare the methods.

Insert process including edge cases

The detailed insertion Flip insert step (1.2) with edge cases is:

- 1.2 Insert the scatter points
 - For each point (p):
 - For each triangle (t)
 - evaluate the state of point (p) for triangle(t):
 - **case** *near vertex [7]*:
 - exclude point (p)
 - next point (p)
 - **case** *near segment [7] and near angular sector [7]*:
 - set point (p) aside
 - next point (p)
 - **case** *"inside [7]"*:
 - split triangle (t) into three at point (p)
 - next point (p)

"Near segment [7] and near angular sector [7]" points are set aside for processing [13] after the insertion phase.

Flipping triangles

The **Flipping** step (2) is performed through successive iterations over the set of triangles while processing the points that were set aside:

- 2/ Flip adjacent triangles that do not meet the Delaunay criterion
 - Do:
 - 2.1/ Process non-compliant triangles:
 - For each triangle (t) in the triangulation:
 - If non-conforming flip the triangle (t) and its relevant adjacent triangle
 - 2.2/ Process the points set aside
 - Continue until 100% conformity is reached

The status of the set-aside points is redetermined when the triangles are transformed to reduce them to two cases:

- "inside [7]" case: to split triangle (t) into three at point (p).
- "near segment [7]" case: to insert the point onto the segment.

The detailed process of the points set aside is:

```

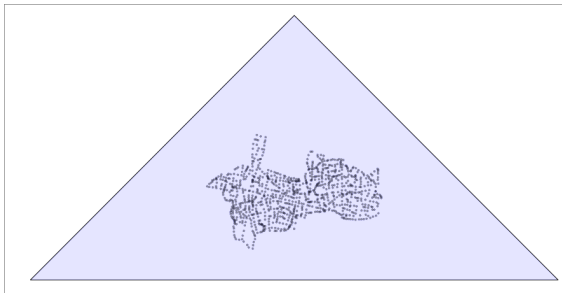
2.2/ Process the points set aside
For each point(p) set aside:
  For each triangle (t):
    - evaluate the status of point (p)
    - "inside [7]" case: split triangle (t)
    - "near vertex [7]" case: exclude point (p)
    - "near segment [7]" case: insert point(p) on segment
    - "near angular sector [7]" case: set point (p) aside
  End of loop (t)
End of loop (p)
  
```

Sample processing

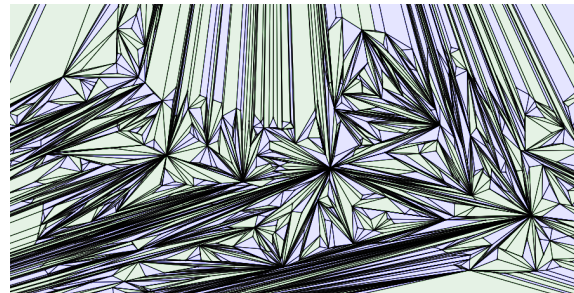
Illustration of each sample processing step of the *DtFlipRef* method.

Triangles that do not conform to the Delaunay triangulation are shaded in green:

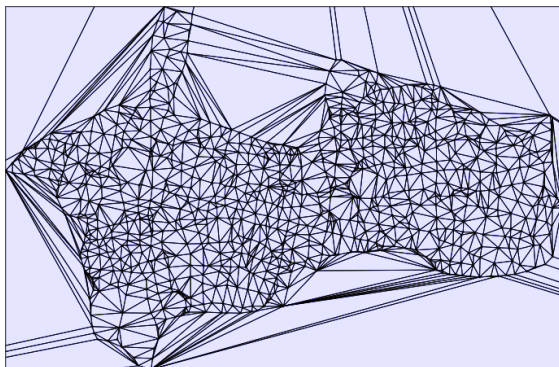
Step 1.1 Create the super triangle



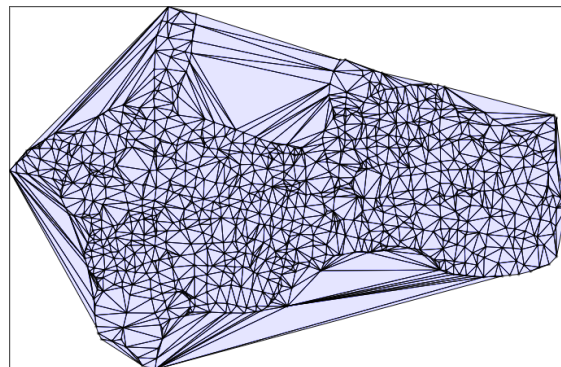
Step 1.2 Insert the scatter points, conformity is 40%



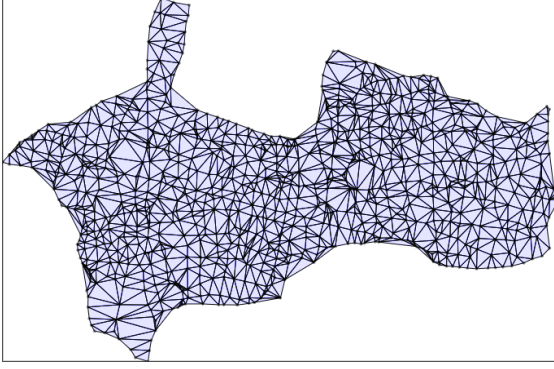
Step 2 Flip adjacent triangles that do not meet the Delaunay criterion



Step 3 Remove the triangles at the edge of the fictitious envelope



Step 4 Remove abnormal triangles on the boundary



***DtRef* and *DtFlipRef* metrics for the test dataset**

Here is the table of metrics produced by the *DtRef* and *DtFlipRef* methods using the test set (**981** points).

Metric	<i>DtRef</i>	<i>DtFlipRef</i>
Ms [8]	113	60
Kb [8]	265	265
C. [9]	5	5
Objects [9]	6,898	6,898
Triangle [9]	1,963	1,963
C. % [9]	100	40
Distance [9]	5	5
Axis [9]	1,547	1,412
P. status [9]	476	441
Circle [9]	8	6
Insert [9]	981	981
Split [9]		
Shift [9]	2,928	3,447
Sc. [9]		9
N. v [9]		
N. s [9]		
Σ op [9]	2,038	1,867
Σ pt [9]	2,077	1,903
B. p [9]	276	276
Ratio [9]	1	1

Metrics N. v [9] and N. s [9] are empty, indicating good dataset quality.

Similarly, the metric Split [9] indicates the absence of collinear points.

Therefore, this dataset does not allow for testing **edge cases**.

Comparative analysis of the *DtRef* and *DtFlipRef* methods

Both methods yield similar metrics:

1. The significant **difference** lies in the number of permutations (Shift [9]), which is higher for DtFlipRef + 17 % (3,447 to 2,928).

This confirms the **advantage** of maintaining conformity during each insertion in the Bowyer-Watson algorithm,

compared to enforcing conformity after point insertion via successive iterations, as in the flip algorithm.

2. However, the DtRef method involves a greater number of operations [9] + 9 % (2038,421 to 1867,104)

and is therefore more computationally intensive in terms of geometric calculations.

3. The DtFlipRef method iterates through the entire set of triangles 9 times to achieve Delaunay conformity of the triangulation.

4. Calculating point status (**476,463**) and handling edge cases add complexity to these methods.

5. Even though the execution time is relatively short (**113 ms**),

the metrics regarding the number of operations by point (**2,077**) reveal the **O(n²)** complexity of this method.

In fact, these methods are heavily **dependent on the distribution** of the points. The execution time for this phase can vary significantly depending on the dataset.

These methods are useful for creating and validating the implementation of the fundamental triangulation model (relationships between triangles, point status, split segment, split triangle, shift methods, etc.).

Optimization via tree-based sorting

Insert steps of DtRef [12] and DtFlipRef [15] of the previous method is improved by applying the famous "**divide and conquer**" principle.

To break the **O(n²)** complexity and reduce the number of point status, points are distributed among the triangles during the point insertion stage.

Each triangle contains the list of included points and is divided into three by the centroid of that list, which is then redistributed among the three new triangles.

With this sorting of the points, the insertion step becomes independent of their distribution.

The algorithm [12] of the DtRef method with tree-based sorting becomes (edge cases [12] are not represented):

- 1/ Create the "super triangle"
- 2/ Insert the scatter points

Add all points to the "super-triangle"

```
For each non-empty triangle(t)
- calculate the midpoint(pc) of the list of included points of triangle(t)
- split triangle (t) into three at point (pc)
- distribute the list of included points among the 3 triangles, excluding the central point(pc)
// Keep Delaunay's conformity
- As long as there is a non-conforming triangle (ts) around vertex (p):
  - flip the triangle (ts) and its relevant adjacent triangle
```

- End of loop (ts)
- Continue as long as there is a non-empty triangle(t)

- 3/ Remove the triangles at the edge of the fictitious envelope
- 4/ Remove abnormal triangles on the boundary

This method is referred to as **DtRefTree** in the remainder of the document.

The algorithm [14] of DtFlipRef method with tree-based sorting becomes (egde cases [12] are not represented):

- 1/ Create the triangulation:
 - 1.1 Create the super triangle
 - 1.2 Insert the scatter points

Add all points to the "super-triangle"

For each non-empty triangle(t)

 - calculate the midpoint(pc) of the list of included points of triangle(t)
 - split triangle(t) into three at point(pc)
 - **distribute** the list of included points among the 3 triangles, excluding the central point(pc)

Continue as long as there is a non-empty triangle(t)
- 2/ Flip adjacent triangles that do not meet the Delaunay criterion
- 3/ Remove the triangles at the edge of the fictitious envelope
- 4/ Remove abnormal triangles on the boundary

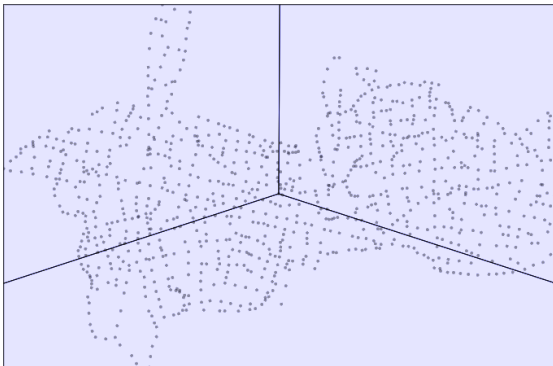
This method is referred to as **DtFlipRefTree** in the remainder of the document.

The process of finding the triangle for a point is identical to the previous one including edge cases:

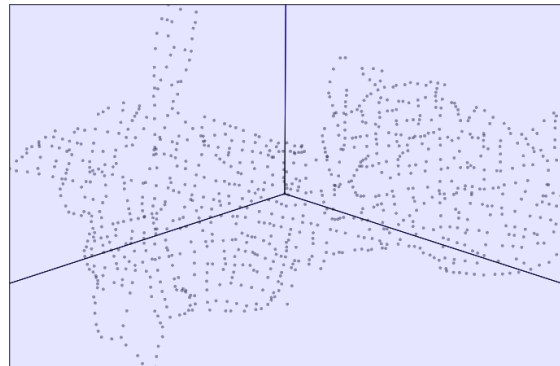
- Points with "near vertex [7]" status are excluded.
- Those classified as "near segment [7]" or "near angular sector [7]" are set aside for processing in step (2) of the triangle transformation.

Triangle subdivision

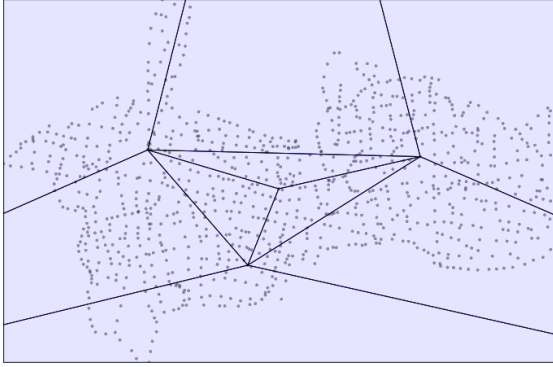
Sorting orders the points and subdivides the triangles by depth, as shown in the illustrations below for the two methods:



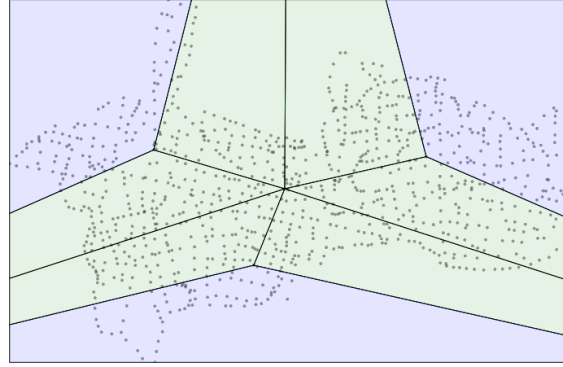
DtRefTree division of triangles after one iteration



DtFlipRefTree [19] division of triangles after one iteration



DtRefTree division of triangles after two iterations


DtFlipRefTree [19] division
of triangles after two iterations

Triangles that do not meet the Delaunay criterion are light green.

Parallel processing

These two methods pave the way for parallelizing the triangulation calculation.

For instance, one can limit the depth of triangle subdivision:

- Treat each triangle and its interior points as an independent triangulation to be constructed,.
- Then merge them to form the final triangulation.

Merging the triangles requires ensuring the alignment of adjacent triangles along their boundaries.

You can find publications on the "divide and conquer" principle for computing a Delaunay triangulation, such as this one:

- DeWall: A fast divide and conquer Delaunay triangulation algorithm.

DtRefTree and *DtFlipRefTree* metrics for the test dataset

Updating the metrics table with the two new methods DtRefTree and DtFlipRefTree [19]:

Metric	<i>DtRef</i>	<i>DtFlipRef</i>	<i>DtRefTree</i>	<i>DtFlipRefTree</i> [19]
Ms [8]	113	60	22	10
Kb [8]	265	265	310	288
C. [9]	5	5	6	6
Objects [9]	6,898	6,898	9,773	8,393
Triangle [9]	1,963	1,963	1,963	1,963
C. % [9]	100	40	100	38
Distance [9]	5	5	54	27
Axis [9]	1,547	1,412	165	103
P. status [9]	476	441	27	13
Circle [9]	8	6	8	5

Insert [9]	981	981	981	981
Split [9]				
Shift [9]	2,928	3,447	2,669	2,936
Sc. [9]		9		8
N. v [9]				
N. s [9]			1	
Σ op [9]	2,038	1,867	256	149
Σ pt [9]	2,077	1,903	261	152
B. p [9]	276	276	323	301
Ratio [9]	1	1	8	14

Comparative analysis of the DtRefTree and DtFlipRefTree methods

The point insertion steps of DtRefTree [18] and DtFlipRefTree becomes independent of their distribution.

Sorting the points yields an immediate gain:

1. The number of point status is divided by :
 - **17** and **33** for DtRefTree and DtFlipRefTree [19] methods.
2. The number of operations [9] is divided by:
 - **8** and **12** for DtRefTree and DtFlipRefTree [19] methods.

Furthermore, the same difference as before is observed between the two methods, just as with the DtRef and DtFlipRef methods:

- The number of permutations (Shift [9]) being higher for DtFlipRefTree [19] + **10** % (2,936 to 2,669).
- The DtRefTree method still [18] involves a greater number of operations [9] + **71** % (256,630 to 149,796).

Algorithm with convex hull

The "super-triangle" is replaced by a triangulation based on the convex hull of the points.

Benefits are:

- No more notional points.
- No more reallocation; the method applies directly to the scatter points.

After calculating the convex hull, the triangulation is initialized by determining the point closest to its centroid, and successive triangles are constructed from this center to each side of the boundary.

The initial triangulation is transformed using the "flip principle [14]" to become Delaunay-compliant.

Based on the same principles as the previous methods, two new methods are defined.

- The *DtCvxTree* method, which uses depth-based sorting of points on triangles and transforms non-conforming triangles upon each point insertion.
- The *DtFlipCvxTree* method, which uses depth-based point sorting on triangles and the "flip principle [14]" at the end of insertion.

By analogy with the algorithm [18] of the *DtRefTree* method, that of the method *DtCvxTree* becomes (edge cases [12] are not represented):

```

1/ Create the initial triangulation based on the convex hull and conforming Delaunay.
2/ Insert the scatter points

    - Distribute the points across the triangles

    For each non-empty triangle(t)
    - calculate the midpoint(pc) of the list of included points of triangle(t)
    - split triangle (t) into three at point (pc)
    - distribute the list of included points among the 3 triangles, excluding the central point(pc)
    // Keep Delaunay's conformity
    - As long as there is a non-conforming triangle (ts) around vertex (p):
      - flip the triangle (ts) and its relevant adjacent triangle
    - End of loop (ts)
    Continue as long as there is a non-empty triangle(t)

3/ Remove the triangles at the edge of the fictitious envelope
4/ Remove abnormal triangles on the boundary

```

Similarly, for the *DtFlipCvxTree* method, by analogy with the algorithm of the *DtFlipRefTree* [19], the algorithm becomes (edge cases are not shown):

```

1/ Create the triangulation:
  1.1 Create the initial triangulation based on the convex hull and conforming Delaunay.
  1.2 Insert the scatter points

    - Distribute the points across the triangles

    For each non-empty triangle(t)
    - calculate the midpoint(pc) of the list of included points of triangle(t)
    - split triangle(t) into three at point(pc)
    - distribute the list of included points among the 3 triangles, excluding the central point(pc)
    Continue as long as there is a non-empty triangle(t)

2/ Flip adjacent triangles that do not meet the Delaunay criterion
3/ Remove the triangles at the edge of the fictitious envelope
4/ Remove abnormal triangles on the boundary

```

Convex hull precision

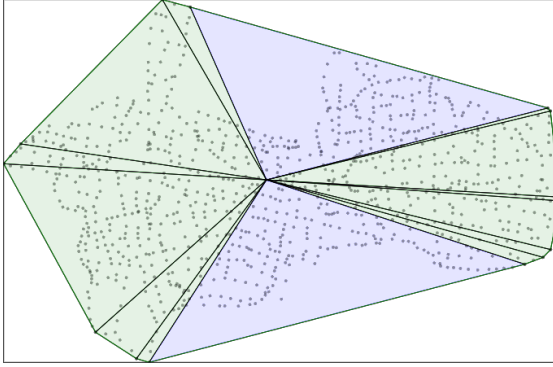
The convex hull is calculated at the same resolution as the precision and filters out points close to its vertices ("near vertex [7]" status).

Triangle subdivision with convex hull

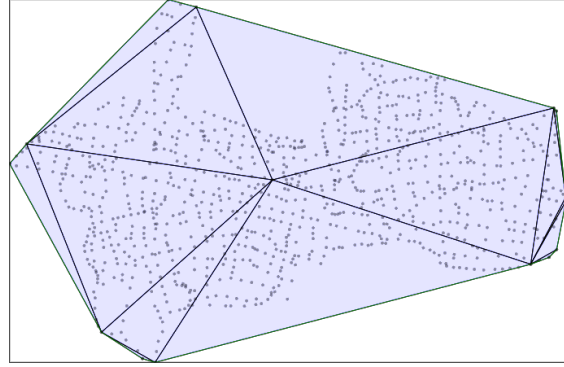
The initial triangulation created with the convex hull comprises **14** triangles.

While the previous method starts with only one triangle, this one uses a finer initial triangulation with 14 triangles and conforms more closely to the perimeter of the point set.

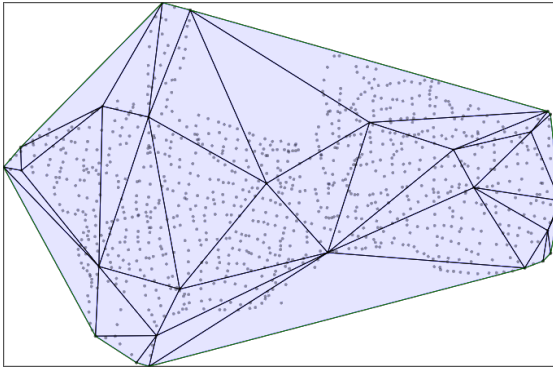
The subdivision of the triangles is more precise, as illustrated by the figures below:



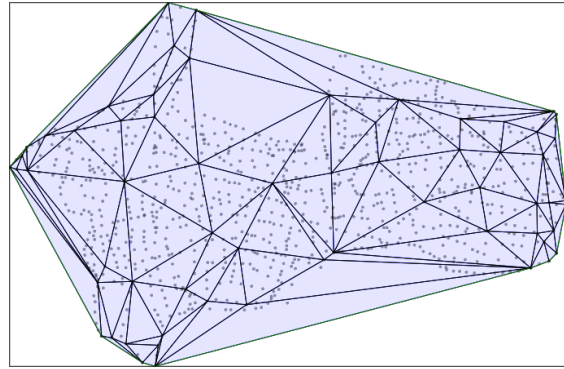
Triangulation calculated on the convex hull



Delaunay conformal triangulation



Division of triangles after one iteration



Division of triangles after two iterations

DtCvxTree and DtFlipCvxTree metrics for the test dataset

The convex hull is calculated in a process (*Cvx*) separate from the triangulation calculation.

The "# ms" column displays the total duration of the method.

Updating the metrics table with the two new methods DtCvxTree [22] and DtFlipCvxTree [22]:

Metric	<i>Cvx</i>	<i>RefTree</i>	<i>Flip</i> . [19]	<i>CvxTree</i> [22]	<i>Flip</i> . [22]
Ms [8]	1	22	10	37	12
Kb [8]	0.71	310	288	281	260
C. [9]	3	6	6	9	9
Objects [9]	22	9,773	8,393	8,702	7,363
Triangle [9]		1,963	1,963	1,946	1,946
C. % [9]		100	38	100	39
Distance [9]	.003	54	27	53	24
Axis [9]	.049	165	103	131	
P. status [9]		27	13	31	21
Circle [9]		8	5	8	6
Insert [9]		981	981	966	966

Split [9]					
Shift [9]		2,669	2,936	2,628	3,129
Sc. [9]			8	8	
N. v [9]					
N. s [9]		1			2
Σ op [9]	.052	256	149	283	183
Σ pt [9]		261	152	288	187
B. p [9]	0.74	323	301	293	271
Ratio [9]		8	14	7	11

Comparative analysis of the DtCvxTree and DtFlipCvxTree methods

Observations on metrics:

1. Calculating the convex hull is very fast (**1 ms**) and has no impact on the total computation time.
2. The indicators of DtCvxTree [22] and DtFlipCvxTree [22] remain equivalent to those of the previous methods DtRefTree and DtFlipRefTree [19] respectively.
3. The advantage of the Bowyer-Watson algorithm (DtCvxTree [22] method) is preserved with a reduced number - **16 %** (from 2,628 to 3,129) of transformations (Shift [9]).
4. The DtCvxTree [22] method still [18] involves a greater number of operations [9] + **35 %** (from 283,157 to 183,886).
5. Reduces the number of unnecessary triangles along the boundary.

Calculation of a convex hull

The calculation is performed at a specific resolution, and the algorithm eliminates points that coincide with the selected vertices.

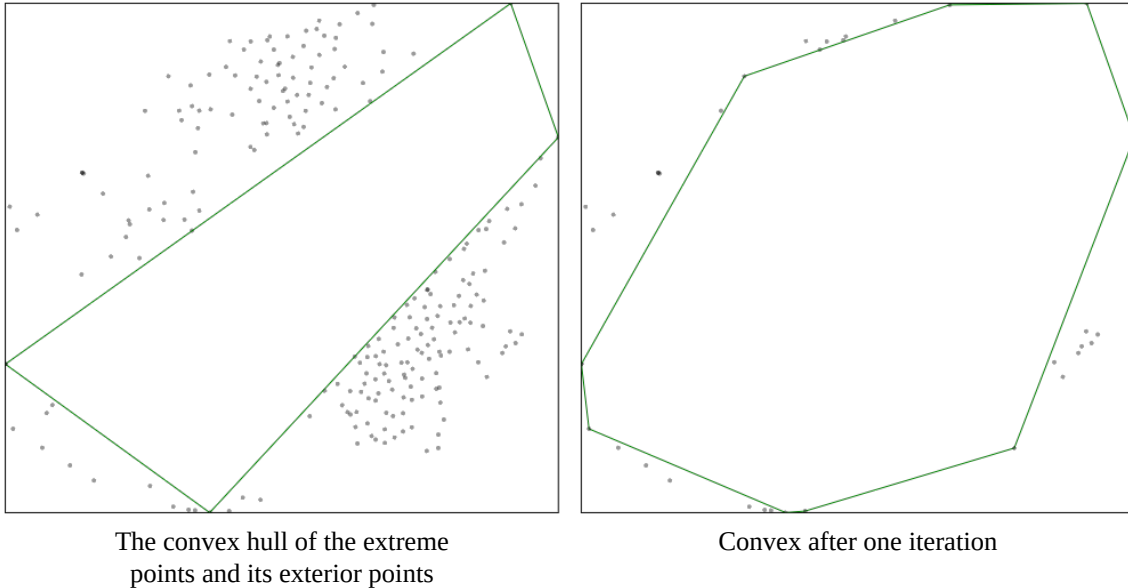
Here is a simple and efficient algorithm for calculating the convex hull of a set of points that uses the "divide and conquer" principle:

```

Initialize the convex hull with the extreme points (x_min, y_max, x_max, y_min).
Distribute each point located outside the convex hull onto its corresponding projected segment.
For each non-empty segment (s):
- Determine the highest point
- Split the segment (s) at that point
- Redistribute the points of (s) across the two segments
End loop (s)

```

The complexity of the algorithm is $O(n \log n)$.



Convex ring algorithm

This method offers an original approach using a classification of points by concentric rings to carry out the first step [14] of creating the triangulation.

The *CvxSurface* structure

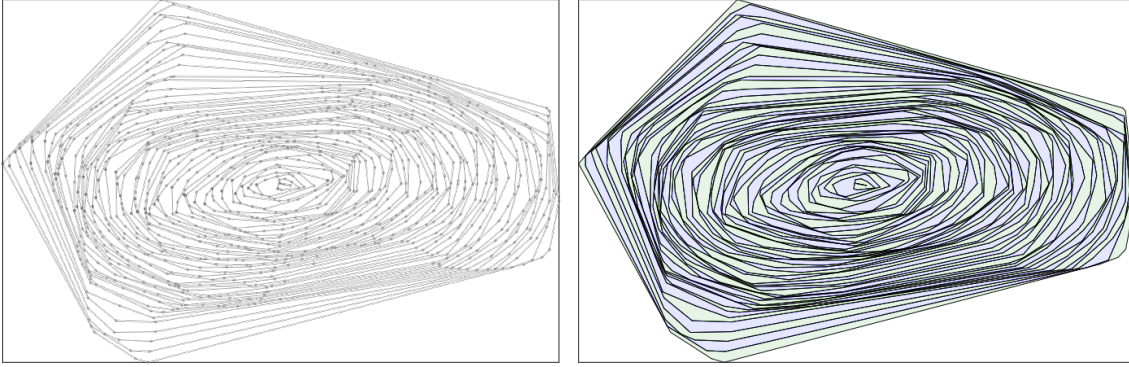
To achieve this, the convex hull calculation is used to accumulate within a structure (named *CvxSurface*) all the hulls of the points, proceeding from the outside toward the center; each hull is determined using the points located inside the preceding one. The final contour may consist of three or more points, or it may be flattened into two points or even a single point.

The calculation method for the *CvxSurface* structure is as follows:

```
Calculate the convex hull (cvx1) of the set of points
While (cvx1) contains points
  Calculate the convex hull (cvx2) of the points inside (cvx1)
  Continue with (cvx2), which becomes (cvx1)
```

The *CvxRing* structure

A second structure (named *CvxRing*) is derived from the *CvxSurface* and consists of adjacent concentric rings delimited by two successive convex hulls (an inner one and an outer one), plus a final **solid** ring with no inner contour.



The structure *CvxSurface* classifies the **981** points into **50** contours. The *CvxRing* structure is composed of **49** rings and a solid last one.

The rings (represented here by alternating shades of green and light blue) cover the space and pass through every point without intersecting.

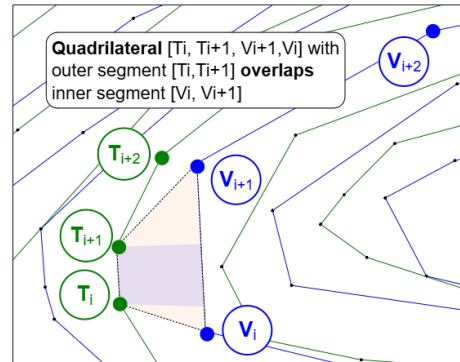
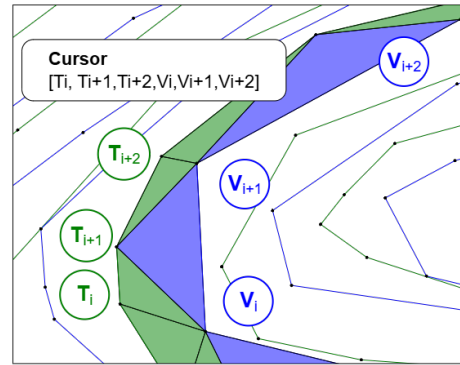
Triangulation cursor

The area of each ring, bounded by its two boundaries, can be triangulated using triangles adjacent to either the outer (green) boundary or the inner (blue) boundary (Triangulation of rings).

To triangulate a ring, a **cursor** is used on three successive points of its outer (T_i) and inner (V_i) contours.

The cursor defines a **quadrilateral** $[T_i, T_{i+1}, V_{i+1}, V_i]$ which, as far as possible, must exhibit an **overlap** between its outer segment $[T_i, T_{i+1}]$ and its inner segment $[V_i, V_{i+1}]$.

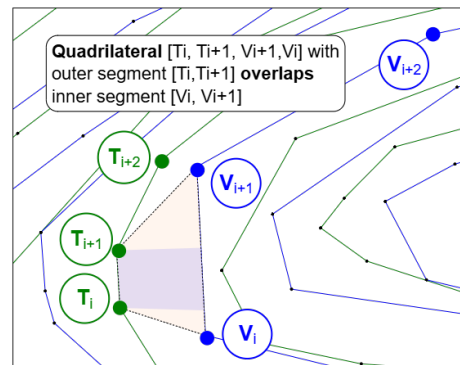
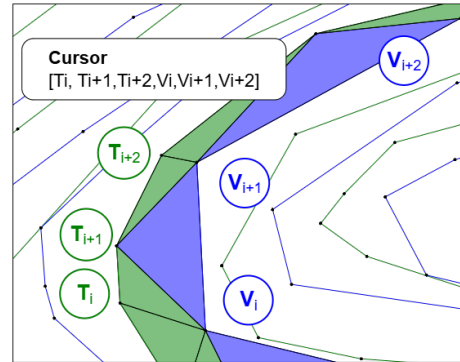
The first two vertices define either the triangle adjacent to the outer contour $[T_i, T_{i+1}, V_i]$ or the triangle adjacent to the inner contour $[V_{i+1}, V_i, T_i]$ (both being consistently oriented clockwise).



Optimal triangle

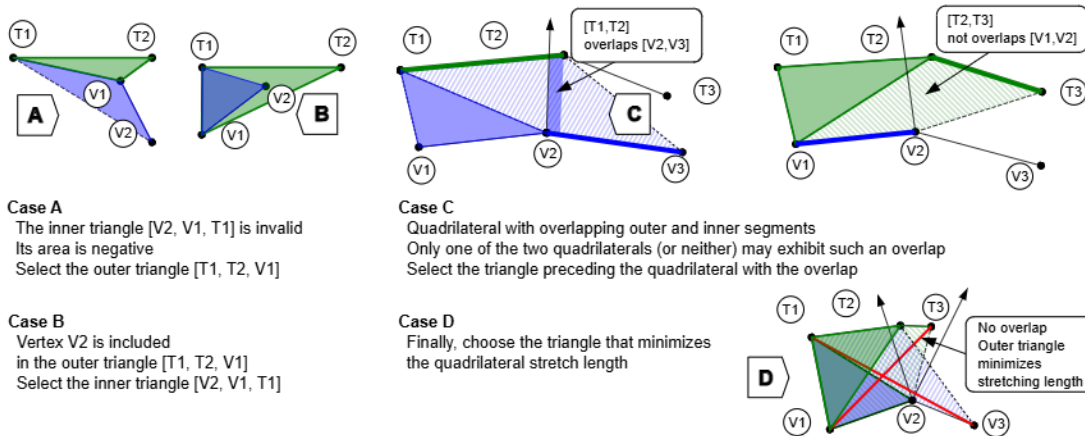
As for the third vertices T_{i+2} and V_{i+2} they serve to select the **optimal** triangle by measuring the **overlap** and the **elongation** length of the two subsequent quadrilaterals.

- The cursor starts at the vertex (T_1) with the minimum x-coordinate on the outer contour and the vertex (V_1) closest to the inner contour
- At each step, the cursor determines which triangle to select **outer** $[T_i, T_{i+1}, V_i]$ or **inner** $[V_{i+1}, V_i, T_i]$ and advances by one vertex along the contour adjacent to the triangle



Choice of triangle

The choice of triangle is made based on the analysis of the vertices (4 cases):



A. The inner triangle $[V_{i+1}, V_i, T_i]$ is invalid (negative area).

Select the **outer** triangle $[T_i, T_{i+1}, V_i]$

B. The vertex V_{i+1} is included in the outer triangle $[T_i, T_{i+1}, V_i]$.

Select the **inner** triangle $[V_{i+1}, V_i, T_i]$.

C. Only **one** of the two

quadrilaterals $[T_i, T_{i+1}, V_{i+2}, V_{i+1}]$ and $[T_{i+1}, T_{i+2}, V_{i+1}, V_i]$,

or **neither** of them, may exhibit an overlap between the outer and inner segments.

Select the triangle preceding the quadrilateral with the **overlap**.

D. Finally, choose the triangle that minimizes the quadrilateral **elongation** length.

Triangulation of the Cvx Ring structure

The *CvxRing* structure thus allows the triangulation to be partitioned ring by ring:

- Each ring constitutes a triangulation.
- The triangles located on the outer edge of a ring are all adjacent, along their longest side, to the triangles on the inner edge of the preceding ring.
- The relationships between the triangles of two consecutive rings are easy to establish.

The triangulation of each ring primarily produces flat triangles, however:

- The transformation of triangles converges rapidly, **ring by ring**.
- Starting with the application of the base-max criterion to the inner boundary of each ring.

The transformation of the triangles is carried out in **five** steps:

```
4/ Transform the triangles on each ring
  For each ring (r) other than the last one:
    - Flip triangles that do not meet the base-max criterion on the inner boundary of the ring (r)
  End of loop (r)
  For each ring (r) other than the last one:
    - Flip triangles that do not meet the Delaunay criterion on the inner boundary of the ring (r)
  End of loop (r)
  For each ring (r):
    - Flip triangles that do not meet the Delaunay criterion on the outer boundary of the ring (r)
  End of loop (r)
  - Flip triangles that do not meet the Delaunay criterion on the last ring
```

Following the calculation of the *CvxSurface* structure, the calculation method for the *CvxRing* structure is as follows:

Compute the (*CvxRing*) structure from the *CvxSurface* structure:

```
1/ Create the rings for each contour
2/ Triangulate each ring
3/ Establish relationships between the triangles at the edges of each ring
4/ Transform the triangles on each ring
```

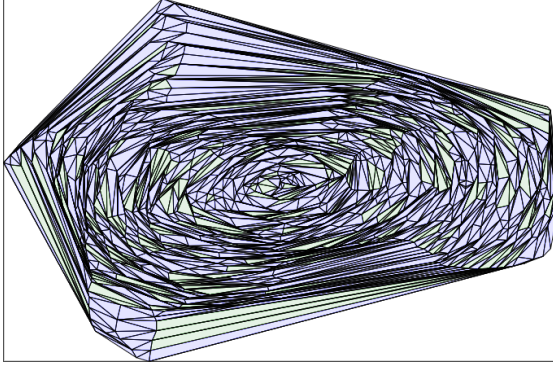
Several steps can be **parallelized**:

- Step 2, Triangulate each ring can be processed in parallel ring by ring.

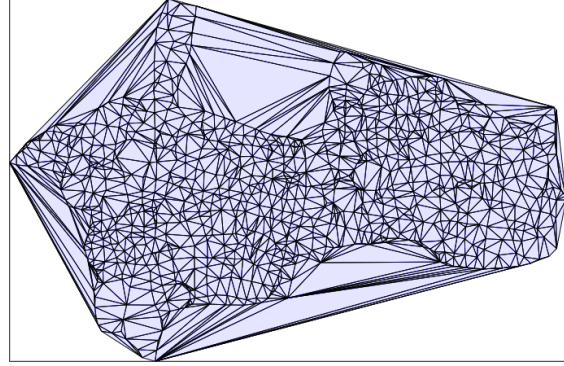
It is also possible to transform the ring triangles at this stage if processed in parallel.

This makes it possible to reduce the number of transformations to be handled after the links between rings are created.

- Step 3, Establishing the relationships between rings can be handled in parallel, group by group of adjacent rings.



Triangulate each ring



Transform triangles on each ring, conformity is **100%**

The algorithm with the CvxRing structure

Once the *CvxRing* structure has been calculated, the triangulation can be formed by accumulating the triangles from the rings.

The new algorithm can be summarized as follows:

- 0/ Calculate the two structures *CvxSurface* and *CvxRing* (which can be processed during a separate upstream)
- 1/ Create the triangulation by accumulating the rings (after this step, the triangulation is **Delaunay** conforming)
- 2/ Remove abnormal triangles on the boundary

Edge cases

Edge cases are naturally handled by the method that does not require calculating the point status:

- The computation of convex hulls eliminates nearby points ("near vertex [7]").
- The *CvxRing* triangulation is performed without insertions, thus eliminating the need to analyze edge cases ("near segment [7]", "near angular sector [7]"), which are resolved by flipping maximal bases.

DtCvxRing metrics for the test dataset

Updating the metrics table with the new method DtCvxRing with calculations of the *CvxSurface* and *CvxRing* structures prior to the triangulation calculation:

Metric	<i>Cvx</i>	<i>CvxSurface</i>	<i>CvxRing</i>	<i>CvxTree</i>	<i>FlipCvxTree</i>	<i>DtCvxRing</i>
Ms [8]	1	42	34	37	12	4
Kb [8]	0.71	49	303	281	260	318
C. [9]	3	5	10	9	9	12
Objects [9]	22	1,386	6,783	8,702	7,363	6,800
Triangle [9]			1,946	1,946	1,946	1,946
C. % [9]				100	39	100
Distance [9]	.003	.352	.548	53	24	
Axis [9]	.049	3	39	189	131	35

P. status [9]				31	21	
Circle [9]			4	8	6	
Insert [9]				966	966	
Split [9]						
Shift [9]			2,291	2,628	3,129	2,291
Sc. [9]					8	
N. v [9]						
N. s [9]					2	
Σ op [9]	.052	4	48	283	183	84
Σ pt [9]		4	49	288	187	86
B. p [9]	0.74	51	316	293	271	332
Ratio [9]		4	49	288	187	86

Analysis of the DtCvxRing method

Observations on metrics:

1. The execution of the *DtCvxRing* method is very fast (4 ms) because it simply involves importing the ring triangles into its triangulation and then removing the anomalous triangles located on the boundary.
2. The conformity of the triangles after creating the *CvxRing* structure is **100%**.
3. The *CvxRing* structure calculation uses only the Axis tool.
4. The number of transformations (Shift [9]) performed by *CvxRing* is low (2,291 compared to 2,928 for the DtRef method) representing a **21 %** reduction.

Triangle transformations are performed **without traversing** the entire triangulation, but solely on the boundaries of the rings.

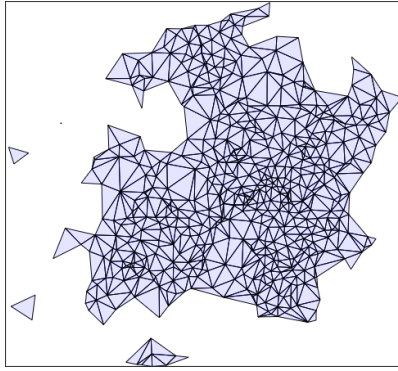
5. The number of operations per point (Σ pt [9]) of *DtCvxRing* is also the lowest (86 compared to 2,077 for the DtRef method) representing a **95 %** reduction or a ratio of **1/24**.
6. While the amount of memory in bytes per point (332 compared to 276 for the DtRef method) is the greatest, representing a **20 %** increase.

In conclusion:

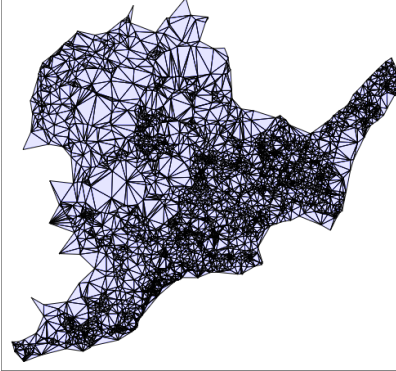
- The method is more complex to implement and consumes more memory **318/265 Kb**.
- The method's logic is distributed across three components (*CvxSurface*, *CvxRing*, *DtCvxRing*) rather than concentrated in a single monolithic component.
- The scalability of the method, achieved by decomposing the set of points into concentric rings, combined with its parallelizability, enables it to handle larger volumes.

Evaluation with other datasets

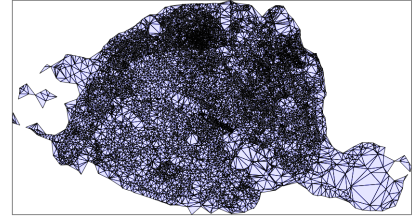
To complement the evaluation of the methods, **three** additional datasets, all derived from topographic surveys, are used.



569 points



3,839 points



9,811 points

The second dataset

The second dataset is a topographic survey with 569 points.

Metric	<i>DtRef</i>	<i>DtFlipRef</i>	<i>DtRefTree</i>	<i>DtFlip..</i>	<i>DtCvxTree</i>	<i>DtFlip..</i>	<i>DtCvxRing</i>
Ms [8]	58	34	16	10	21	13	3
Kb [8]	154	154	180	167	161	149	192
C. [9]	5	5	6	6	9	9	12
Objects [9]	4,014	4,014	5,675	4,894	4,998	4,235	4,176
Triangle [9]	1,139	1,139	1,139	1,139	1,118	1,118	1,118
C. % [9]	100	38	100	39	100	39	100
Distance [9]	3	3	29	15	29	12	
Axis [9]	585	511	90	56	106	65	20
P. status [9]	173	153	14	7	17	9	
Circle [9]	4	3	4	3	4	3	
Insert [9]	569	569	569	569	550	550	
Split [9]							
Shift [9]	1,599	2,034	1,514	1,571	1,473	1,645	1,134
Sc. [9]		8		6		8	
N. v [9]							
N. s [9]							
Σ op [9]	767	672	138	82	158	90	44
Σ pt [9]	1,347	1,181	243	144	279	158	78
B. p [9]	277	277	324	302	291	269	347
Ratio [9]	1,347	1,181	243	144	279	158	78

The third dataset

The third dataset contains **3,839** points (**1** of which are excluded due to their proximity to another point).

Metric	<i>DtRef</i>	<i>DtFlipRef</i>	<i>DtRefTree</i>	<i>DtFlip..</i>	<i>DtCvxTree</i>	<i>DtFlip..</i>	<i>DtCvxRing</i>
Ms [8]	1,053	848	63	50	143	49	10
Kb [8]	1,035	1,035	1,211	1,126	1,103	1,020	1,186
C. [9]	5	5	6	6	9	9	12
Objects [9]	26,902	26,900	38,143	32,763	34,202	28,871	24,978
Triangle [9]	7,677	7,677	7,677	7,677	7,660	7,660	7,660
C. % [9]	100	37	100	38	100	39	100
Distance [9]	23	23	270	124	288	115	
Axis [9]	22,086	20,442	1,152	882	1,520	697	127
P. status [9]	7,147	6,695	202	189	291	132	
Circle [9]	34	26	32	23	32	25	
Insert [9]	3,832	3,838	3,830	3,832	3,813	3,821	
Split [9]	12		16	12	20	4	
Shift [9]	11,338	14,372	10,502	11,871	10,414	13,569	11,675
Sc. [9]		9		9		10	
N. v [9]	1	1	1	1	1	1	
N. s [9]	13	6	24	20	29	11	
Σ op [9]	29,291	27,187	1,657	1,219	2,133	970	370
Σ pt [9]	7,629	7,081	431	317	555	252	96
B. p [9]	276	276	323	300	294	272	316
Ratio [9]	7,629	7,081	431	317	555	252	96

The fourth dataset

The fourth dataset contains **9,811** points (**1** of which are excluded due to their proximity to another point).

Metric	<i>DtRef</i>	<i>DtFlipRef</i>	<i>DtRefTree</i>	<i>DtFlip..</i>	<i>DtCvxTree</i>	<i>DtFlip..</i>	<i>DtCvxRing</i>
Ms [8]	6,359	5,263	284	141	514	253	16
Kb [8]	2,644	2,644	3,097	2,878	2,828	2,607	2,853
C. [9]	5	5	6	6	9	9	12
Objects [9]	68,706	68,706	97,675	83,650	87,922	73,765	58,048
Triangle [9]	19,621	19,621	19,621	19,621	19,593	19,593	19,593

C. % [9]	100	39	100	39	100	39	100
Distance [9]	59	59	805	338	815	307	
Axis [9]	140,658	134,492	5,269	3,126	5,802	3,479	325
P. status [9]	46,233	44,464	1,048	687	1,126	806	
Circle [9]	87	72	83	66	83	68	
Insert [9]	9,800	9,804	9,792	9,796	9,764	9,772	
Split [9]	20	12	36	28	40	24	
Shift [9]	28,991	40,983	26,831	36,045	27,062	37,383	31,936
Sc. [9]		10		10		10	
N. v [9]	1	1	1	1	1	1	
N. s [9]	24	29	57	45	64	44	
Σ op [9]	187,038	179,088	7,206	4,218	7,829	4,662	972
Σ pt [9]	19,064	18,253	734	430	797	475	99
B. p [9]	276	276	323	300	295	272	297
Ratio [9]	19,064	18,253	734	430	797	475	99

Assessment of method complexity

To evaluate the complexity of the methods, their metrics are analyzed across the **four** datasets, based on:

- The number of operations (Σ op [9]) required to compute the triangulation for each dataset. independently of the language and execution platform.
- The number of points of each dataset (np).

Method	Number of operations (Σ op [9]) in thousands by dataset			
<i>DtRef</i>	767	2,038	29,291	187,038
<i>DtFlipRef</i>	672	1,867	27,187	179,088
<i>DtRefTree</i>	138	256	1,657	7,206
<i>DtFlipRefTree</i> [19]	82	149	1,219	4,218
<i>DtCvxTree</i> [22]	158	283	2,133	7,829
<i>DtFlipCvxTree</i> [22]	90	183	970	4,662
<i>DtCvxRing</i>	44	84	370	972
<i>Dataset points (np)</i>	569	981	3,839	9,811

Method for evaluating correlation equations for the number of operations

By treating the measurements (Σ op [9], np) obtained for each method as a series, probabilistic techniques (calculation of standard deviation, variance, and variation) are applied to evaluate the correlation equations.

Subsequently, to evaluate a correlation equation $\sum op [9]=f(np)$, its percentage variation (standard deviation divided by the mean value) is calculated across the five datasets.

A percentage variation of less than 15% is accepted as an indicator of the equation's plausibility.

Evaluation of the complexity of the DtRef method

For the *DtRef* method, as expected given its $O(n^2)$ complexity, we observe that the value $2*(np)^2$ is close to the value $\sum op [9]$, as shown by the calculation for the last dataset:

$$2*(np)^2 = 2*(3,839)^2 = 29475,842 \quad op = 29291,302.$$

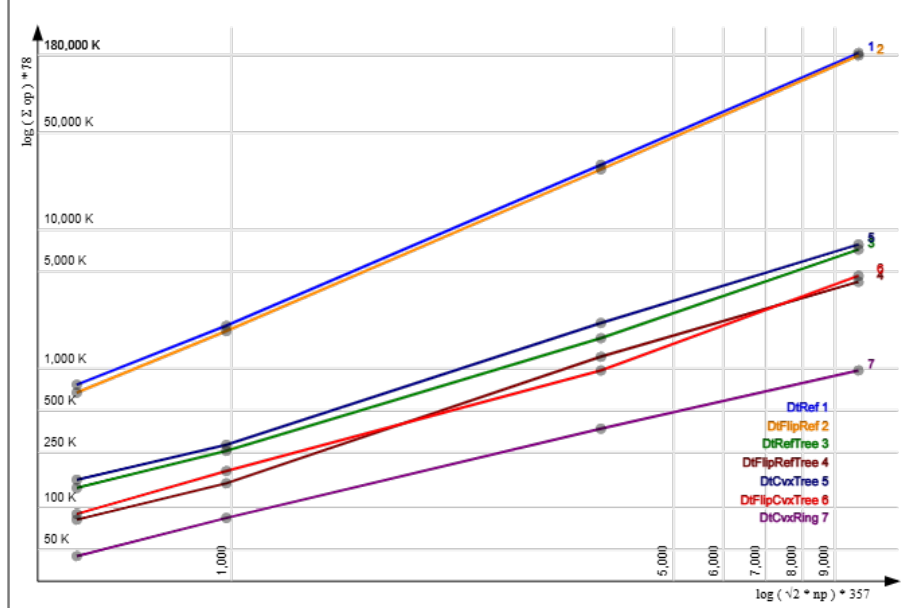
To confirm this equation for this method and *DtFlipRef*, we calculate its percentage variation [34]:

Method	Equation	Variation [34]%
<i>DtRef</i>	$\sum op \approx 2.01 * (np \wedge 2.01)$	8.65
<i>DtFlipRef</i>	$\sum op \approx 2.0 * (np \wedge 2.0)$	4.37

The value of the variation [34] is less than 15%, confirming the equation.

To proceed, we represent this equation on a graph with for each method:

- On the x-axis the number of dataset points (np), expressed as $\log(\sqrt{2} * np)$.
- On the y-axis the number of operations ($\sum op [9]$) to calculate the Delaunay triangulation of the dataset, expressed as $\log(\sum op [9])$.



It can be seen from the graph that:

- The correlation for the *DtRef* and *DtFlipRef* methods is confirmed by the alignment of the points on its curve (1).
- The curves corresponding to the four methods *DtRefTree* (3), *DtFlipRefTree* [19] (4), *DtCvxTree* [22] (5), *DtFlipCvxTree* [22] (6) are close to one another, and their points are nearly aligned.

- In contrast, the *DtCvxRing* curve (7) deviates from the previous four, and its points are not aligned.

It is assumed that:

- The four methods *DtRefTree*, *DtFlipRefTree* [19], *DtCvxTree* [22], *DtFlipCvxTree* [22], which are optimizations of *DtRef* and *DtFlipRef*, follow the same law with different coefficients.
- Whereas *DtCvxRing* follows a different one.

Complexity evaluation of the *DtRefTree*, *DtFlipRefTree*, *DtCvxTree* and *DtFlipCvxTree* methods

We generalize the *DtRef* equation to determine the equations for the other four by calculating:

- The ratio $K(f) = \log(\sum op [9]) / \log(f * \sqrt{2} * np)$, for (f) values from 1 to 10.
- The percentage variations $V(f)$ of the ratio $K(f)$ for each value (f), in order to select the value of (f) with the lowest variation.
- The percentage variation of the correlation $\sum op [9] = (f * \sqrt{2} * np) ^ K(f)$.

Its selection of the best correlation for each method leads to the following equations :

Method	(f)	$K(f)$	$V(f) \%$	$\sum op$	Variation %
<i>DtRef</i>	1	2.01	0.55	$\approx 2.01 * (np ^ 2.01)$	8.65
<i>DtFlipRef</i>	1	2.0	0.29	$\approx 2.0 * (np ^ 2.0)$	4.37
<i>DtRefTree</i>	6	1.39	0.58	$\approx 19.41 * (np ^ 1.39)$	7.75
<i>DtFlipRefTree</i> [19]	4	1.4	0.59	$\approx 11.23 * (np ^ 1.4)$	7.05
<i>DtCvxTree</i> [22]	7	1.38	0.53	$\approx 23.63 * (np ^ 1.38)$	6.46
<i>DtFlipCvxTree</i> [22]	6	1.34	0.77	$\approx 17.65 * (np ^ 1.34)$	10.84

Following this calculation, we observe:

- For the *DtRef* method, the factor of 2 is indeed recovered, as expected.
- For the *DtCvxTree* [22] method, the exponent of 1.38 is close to the square root of 2, which suggests an equation of the order of $np^{\sqrt{2}}$.

Next, the $(\sum op [9] / np^{\sqrt{2}})$ ratio of the equation is determined for the three methods, followed by the variation:

Method	Equation	Variation [34]%
<i>DtRef</i>	$\sum op \approx 2.01 * (np ^ 2.01)$	8.65
<i>DtFlipRef</i>	$\sum op \approx 2.0 * (np ^ 2.0)$	4.37
<i>DtRefTree</i>	$\sum op \approx 15.79 * (np ^ \sqrt{2})$	8.28
<i>DtFlipRefTree</i> [19]	$\sum op \approx 9.79 * (np ^ \sqrt{2})$	6.86
<i>DtCvxTree</i> [22]	$\sum op \approx 18.19 * (np ^ \sqrt{2})$	7.03
<i>DtFlipCvxTree</i> [22]	$\sum op \approx 10.27 * (np ^ \sqrt{2})$	11.64

Complexity evaluation of the *DtCvxRing* method

For the *DtCvxRing* method, and in order to confirm the assumed $O(n \log n)$ complexity, we evaluate the following ratio $R(p)$:

- $R(p) = \log(\sum \text{op}[9]) / \log(p * np * \log(np))$.
- We are looking for the value of (p) for which the ratio $R(p)$ is close to **1**, with a variation $V(p)$ of less than **1%**.

A bisection search on the interval $[1, 100]$ for (p) :

- Yields a value of (p) equal to **11.897**, with a variation $V(p)$ of **0.47%**.
- Ultimately, the variation of the correlation equation $\sum \text{op}[9] \approx 11.897 * np * \log(np)$ is **5.83%**, confirming the hypothesis of $O(n \log n)$ complexity.

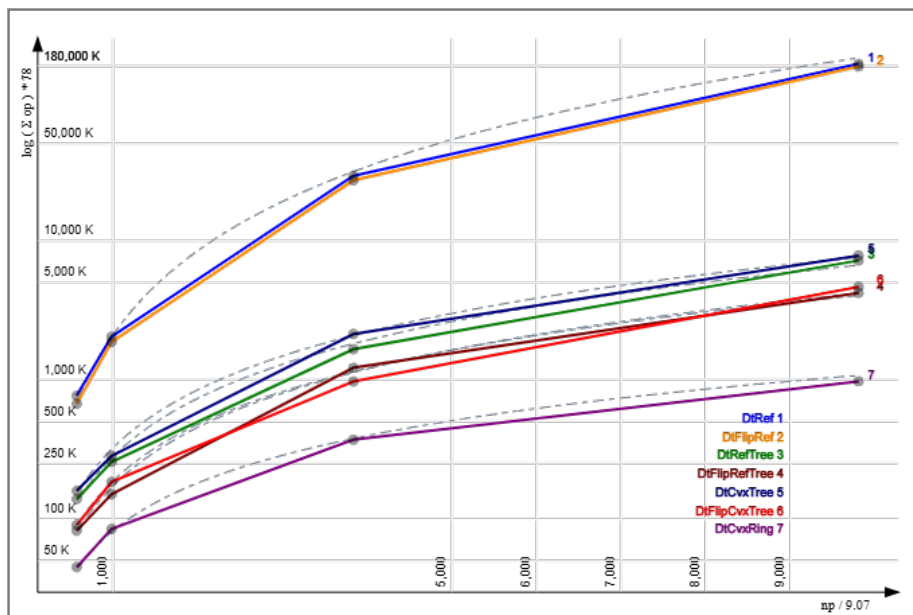
Summary of the method complexity assessment

The table below summarizes the correlation equation ($\sum \text{op}[9] \approx f(np)$), the percentage of variation, and the complexity for each method:

Method	Equation	Variation %	Complexity
<i>DtRef</i>	$\sum \text{op} \approx 2.01 * (np^{2.01})$	8.65	$O(np^2)$
<i>DtFlipRef</i>	$\sum \text{op} \approx 2.0 * (np^{2.0})$	4.37	$O(np^2)$
<i>DtRefTree</i>	$\sum \text{op} \approx 15.79 * (np^{\sqrt{2}})$	8.28	$O(np^{\sqrt{2}})$
<i>DtFlipRefTree</i> [19]	$\sum \text{op} \approx 9.79 * (np^{\sqrt{2}})$	6.86	$O(np^{\sqrt{2}})$
<i>DtCvxTree</i> [22]	$\sum \text{op} \approx 18.19 * (np^{\sqrt{2}})$	7.03	$O(np^{\sqrt{2}})$
<i>DtFlipCvxTree</i> [22]	$\sum \text{op} \approx 10.27 * (np^{\sqrt{2}})$	11.64	$O(np^{\sqrt{2}})$
<i>DtCvxRing</i>	$\sum \text{op} \approx 11.897 * np * \log(np)$	5.83	$O(np * \log(np))$

To complement this table, the following graph shows:

- The curves ($y=\log(\# \text{op}[9]), x=np$) for each method.
- Along with their correlation equations (gray dashed line).



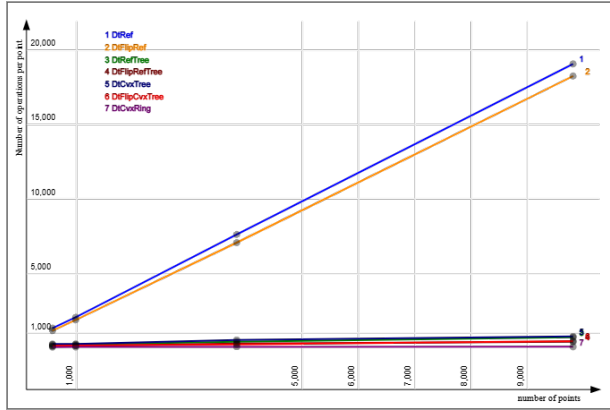
A scale factor is applied to each axis to fit the 4:3 ratio.

Evaluating the strengths and weaknesses of the methods

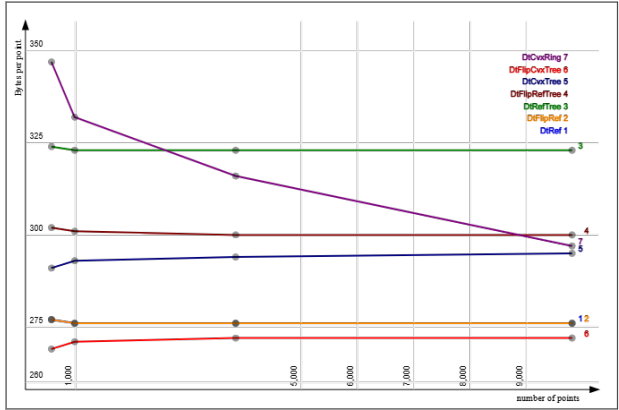
To evaluate the strengths and weaknesses of the methods, we rely on criteria based on metrics.

When weighed against the requirements of the application context, these criteria help guide the choice.

The two metrics, number of operations per point (Σ pt [9]) and number of bytes per point (B. p [9]), are decisive in the evaluation of the methods, as shown by their graphs:



Number of operations per point



Number of bytes per point

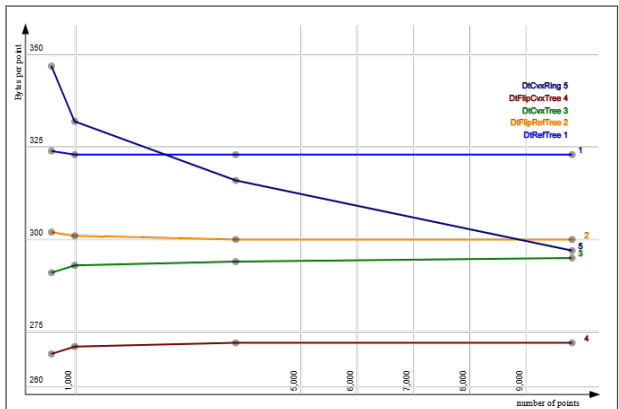
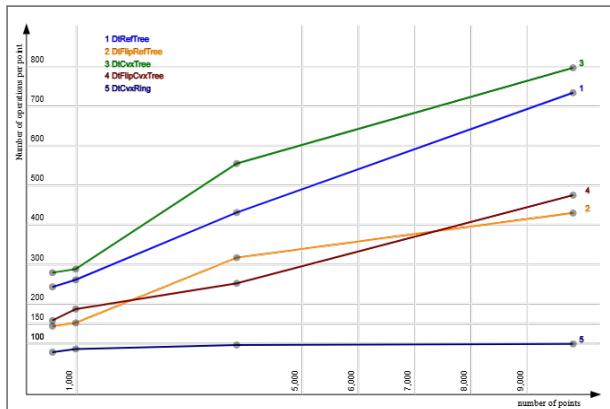
For production deployment, the DtRef and DtFlipRef methods are unacceptable due to their $O(n^2)$ complexity, as shown by the graph of the number of operations per point.

The choice therefore falls on the other five methods.

The second graph, showing the number of bytes per point, reveals:

- that this value remains practically **constant** for the four methods other than DtCvxRing
- whereas for the latter, it decreases to **297** bytes for the final dataset containing **9,811** points.

We will therefore use the measurements obtained on the final dataset to compare the last five methods.



Number of operations per point

Number of bytes per point

After narrowing the choice down to the five last methods, the graph showing the number of operations per point reveals:

- that the metric for the first four methods ranges from 144 to **797**,
 - whereas for the DtCvxRing method, the metric remains close to 100, from 78 to **99**,
- combined with a byte count per point of 347 to **297**,
- demonstrating the method's high execution **stability**.

The evaluation of the methods is based on quantified criteria and design-related criteria.

Quantified evaluation criteria

We use the measurements obtained on the final dataset to compare the last five methods.

Each criterion results in a score from 1 to 10.

Criteria

1. Reduced **memory** consumption

This criterion is quantified by measuring the number of bytes per point (B.p [9]).

The number of bytes per point ranges from **272** (score 10) to **323** (score 1).

2. Few geometric **calculations** are performed

This criterion is quantified by measuring the number of operations per point (Σ pt [9]).

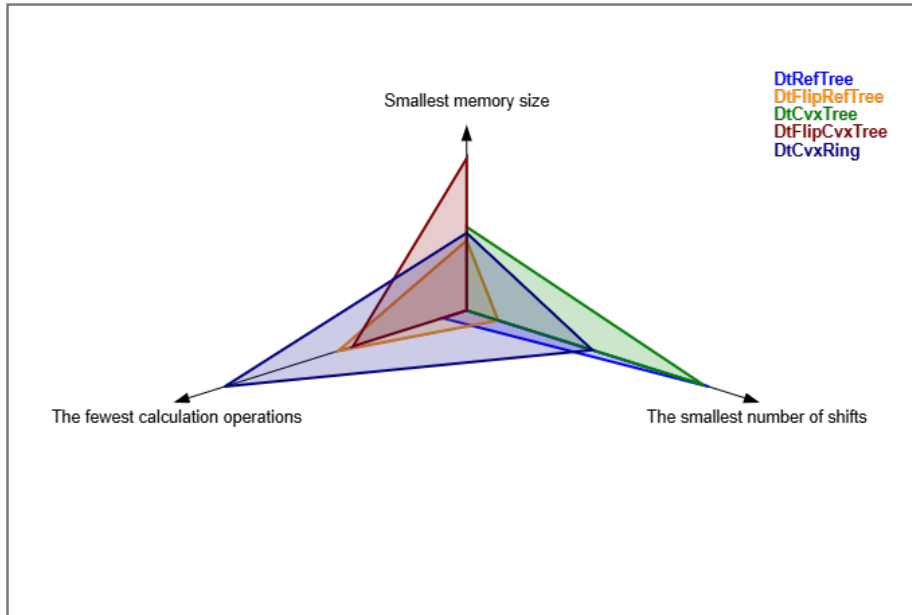
The number of operations per point ranges from **99** to **797**.

3. Reduced **shifts** [9] count

The number of shifts per point ranges from **2.73** (score 10) to **3.81** (score 1).

Evaluation of methods using quantified criteria

Criteria	DtRefTree	DtFlip..	DtCvxTree	DtFlip..	DtCvxRing
Memory	1	5	6	10	6
Calculations	1	6	1	5	10
Shifts count	10	2	10	1	6
Σ	12	13	17	16	22



Design criteria

These criteria do not result in a measured score but identify a characteristic classified as a strength.

Strength / Weakness

1. **Flexible** design

One design indicator is the number of OOP classes (C. [9]) involved in the method.

A simple design with few classes is typically viewed as a weakness, as it suggests monolithic code.

In contrast, a design with a larger number of classes reflects logic distributed across multiple concepts, resulting in source code that is more maintainable and adaptable.

The number of classes ranges from **6** (DtRefTree) to **12** (DtCvxRing).

2. **Operates** directly on the data set points

Does not require reallocating them or adding dummy points.

3. Some stages of the processing can be **parallelized**.

This characteristic is considered a strength for handling large volumes and fully utilizing processor capabilities.

4. An algorithm **complexity** of $O(np \cdot \log(np))$.

Greater complexity consumes more resources during production.

Evaluation of methods using design criteria

Strength / Weakness	DtRefTree	DtFlip..	DtCvxTree	DtFlip..	DtCvxRing
Flexible design processing	No	No	No	No	Yes

Operates on dataset points	<i>No</i>	<i>No</i>	Yes	Yes	Yes
Parallel processing	<i>No (1)</i>	<i>No (1)</i>	<i>No (1)</i>	<i>No (1)</i>	Yes
Complexity $O(np \cdot \log(np))$	<i>No</i>	<i>No</i>	<i>No</i>	<i>No</i>	Yes

(1) Parallelizing depth-first sorting methods is possible, provided the algorithm undergoes significant modifications.

General conclusion

The last five methods can be distinguished as follows:

- The group of the two methods (DtRefTree, DtCvxTree [22]) based on the Bowyer-Watson algorithm.

These methods proceed by iteration, without traversing all the triangles.

The second method (DtCvxTree [22]) has two advantages, as it consumes less memory and directly uses the points from the dataset.

- The second group of the two methods (DtFlipRefTree [19], DtFlipCvxTree [22]) that uses the flip principle [14].

The flip principle is simpler to implement than the Bowyer-Watson iteration.

However, these methods involve more shifts than the previous ones.

The second method (DtFlipCvxTree [22]) retains its advantages, just as in the previous group.

- And finally, the DtCvxRing method, which is orthogonal to the previous ones.

This last method differs from the others in that:

1. A **concentric ring design**, as opposed to the point-injection design of other methods.

And indeed, it does not use complex operations for inserting a point into a triangle or onto a segment.

Is using only the axis tool for all these calculations.

2. A simple and effective treatment for edge cases.

3. **Stable** execution,

evidenced by metrics indicating an operation-per-point count close to 100 (from 78 to **99**) and a byte-per-point count approaching 300 (from 347 to **297**).

4. A more **flexible** design built on **12** classes.

5. An algorithm **complexity** of $O(np \cdot \log(np))$.

6. Some stages of the processing can be **parallelized**.

References

Delaunay triangulation

- **Delaunay, B.** (1934). Sur la sphère vide. A la mémoire de Georges Voronoi. Bulletin de l'Académie des Sciences de l'URSS, Class. Sci. Nat.

- **De Berg, M., Cheong, O., van Kreveld, M., Overmars, M.** (2008). Computational Geometry: Algorithms and Applications (3rd ed.). Springer.
- **Cheng, S. W., Dey, T. K., Shewchuk, J.** (2012). Delaunay Mesh Generation. CRC Press.

Bowyer-Watson algorithm

- **Bowyer, A.** (1981). Computing Dirichlet tessellations. The Computer Journal, 24(2), 162-166.
- **Watson, D. F.** (1981). Computing the n-dimensional Delaunay tessellation with applications to Voronoi polytopes. The Computer Journal, 24(2), 167-172.

Lawson's Flip

- **Lawson, C. L.** (1977). Software for C1 surface interpolation. In Mathematical Software III (pp. 161-194). Academic Press.
- **Edelsbrunner, H., Shah, N. R.** (1996). Incremental topological flipping works for regular triangulations. Algorithmica, 15(3), 223-241

Divide and Conquer algorithm

- **Guibas, L., Stolfi, J.** (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics (TOG), 4(2), 74-123.
- **Shamos, M. I., Hoey, D.** (1975). Closest-point problems. In 16th Annual Symposium on Foundations of Computer Science (FOCS) (pp. 151-162). IEEE.

Convex hull of a point set

- **F. P. Preparata et S. J. Hong.** (1977). Convex hulls of finite sets of points in two and three dimensions.
- **David G. Kirkpatrick et Raimund Seidel.** (1986). The Ultimate Planar Convex Hull Algorithm.