APPLICATION OF THE COMPUTATIONAL GEOMETRY ALGORITHMS IN THE SOFTWARE PACKAGE MATHEMATICA

Aybeyan Selimi, page 29-46

ABSTRACT

Computational geometry is a branch of computer science that discusses algorithms for solving geometric problems. It considers such tasks as triangulation, building a convex hull, determining whether one object belongs to another, finding their intersection, etc. Computational geometry algorithms operate with the geometric objects with the point, a segment, a polygon, and circles. Two important algorithms of computational geometry that have many applications are Delaunay triangulation and the Voronoi diagram. The Voronoi splitting is used in computational materials science to create synthetic polycrystalline aggregates. In computer graphics is used for randomly splitting surfaces. These algorithms are used in a gold method or the area stealing method for interpolating a function in 2D. In this paper, are shown three implementations of the Delaunay triangulation and Voronoi diagram in the software package Mathematica.

Keywords. Computational geometry, Voronoi diagram, Delaunay triangulation, polygon, convex hull and algorithm.

Ass. Prof. Aybeyan Selimi Ph.D,

Faculty of Informatics, International Vision University, Gostivar - North Macedonia;

e-mail:
aybeyan@vizyon.edu.mk

UDK: 004.92.021:514.11

Date of received:
25.08.2019

Date of acceptance:
22.10.2019

Declaration of interest:
The authors reported no conflict of interest related to this article.

1. INTRODUCTION

Computational geometry is a descendant of classical geometry and computer science. It is part of algorithms that deal with the development and analysis of efficient algorithms and data structures suitable for solving geometric problems (Selimi et. al, 2019). Particularly are important the geometric problems where brute force solutions are not practically usable (Janičić, 2016). Computational geometry as a discipline is developed thanks to problems and applications in computer graphics, computer vision, robotics, databases, geographic information systems, CAD / CAM systems, molecular biology, etc. Some of the specific applications are applications in virtual reality, motion planning, collision detection, drug design, fluid dynamics, etc (Selimi et. al, 2018). The field of computational geometry usually deals with problems in the Euclidean plane or space and involves the availability of elementary operations such as: checking that the point belongs to a straight / circle, whether they are straight-cut, and the like.

In this paper we implement the computational geometry algorithms in software package Mathematica. Mathematica is the world’s only fully integrated environment for technical computing (Wolfram, 2013). It is developed in programming language Wolfram.

The Mathematica program is divided into two parts, the kernel, and the front end. The kernel interprets the expressions in Wolfram language and gives the resulting expressions. The front end, designed by Theodore Gray, has a graphical user interface, which allows the creation and modification of documents containing program code with prettyprinting, formatted text along with results including mathematics, graphics, graphical user interface components, tables, and sounds. All content and all formatting can be algorithmically generated and interactively modified. Most standard text editing options are supported. It also contains a spell checker but does not do this in parallel as the user enters the text.

Documents can be structured using a cell hierarchy, which allows for document sharing and sketching, and supports automatic number indexing. Documents can be displayed as a slide show for presentations. Notebooks and their contents are presented as expressions in Mathematica that can be created, modified or analyzed by Mathematica. This allows conversion to other formats such as TeX or XML.

The front features development tools like debuggers, add-ons, and automatic syntax coloring. Among the alternative front ends is Wolfram Workbench, an integrated Eclipse-based development environment, first shown in 2006. It provides tools for project-based Mathematica code development, including audit management, debugging, profiling and testing (https://www.macworld.com). The Mathematica Kernel also has a command line for the front end (https://reference.wolfram.com). Other interfaces include JMath, based on the GNU readline (http://robotics.caltech.edu), and MASH, which runs standalone Mathematica programs (with arguments) from the UNIX command line. Wolfram Research has published a series of tutorials for beginners to familiarize the user with the user interface and the launche .

2. COMPUTATIONAL GEOMETRY ALGORITHMS

Computational geometry is basically a discrete discipline. Problems of computational geometry are generally the subject of research in other fields of science such as “*geometric modeling*”. Computational geometry explores simple and easily approximating surfaces and geometric objects. The basic terms that appear in this discipline are the point and line segment, which then builds more complex structures.

Among the most important geometric figures of this discipline are the polygons in the plane, while in their space their generalization of polyhedra. Polygon is a closed geometric figure in a plane that is a finite collection of crossed line segments called the edges of the polygon. The points where the two edges meet are called vertices of the polygon. A set of all edges and vertices of a polygon is called the boundary of the polygon and is labeled with ∂P. In the Figure 1.1(a) is given polygon with nine edges joined at nine vertices. In the diagrams (b)–(d) are given objects that fail to be polygons. From Jordan curve theorem we know that the every simple closed planar curve separates the plane into a bounded interior region and an unbounded exterior. For this reason in this paper, polygons representing a special part of the Jordan curve theorem are analyzed.

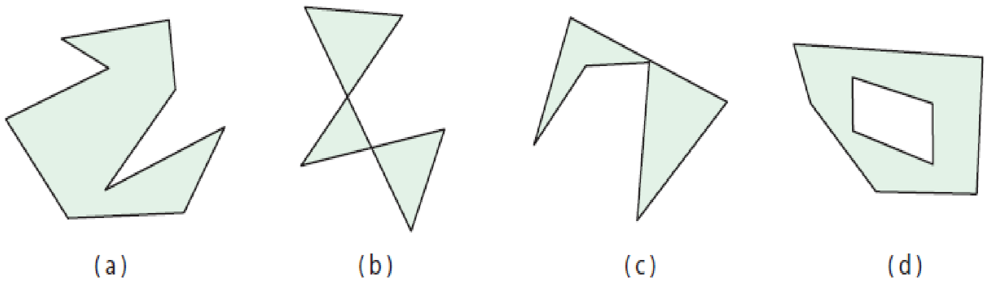


Figure 2.1. (a) A polygon. (b)–(d) Objects that are not polygons [17 ,pp.2].

Theorem 1. (**Polygonal Jordan Curve**) (Devadoss et al., 2011). *The boundary ∂P of a polygon P partitions the plane into two parts. In particular, the two components of $\mathbb{R}^2 \setminus \partial P$ are the bounded interior and the unbounded exterior.*

Proof. Let P be a polygon in the plane. We first choose a fixed direction in the plane that is not parallel to any edge of P. This is always possible because P has a finite number of edges. Then any point x in the plane not on ∂P falls into one of two sets:

1. The ray through x in the fixed direction crosses ∂P an even number of times: x is exterior. Here a ray through a vertex is not counted as crossing ∂P.
2. The ray through x in the fixed direction crosses ∂P an odd number of times: x is interior.

Notice that all points on a line segment that do not intersect ∂P must lie in the same set. Thus the even sets and the odd sets are connected. And moreover, if there is a path between points in different sets, then this path must intersect ∂P.

2.1 Triangulation

Definition 1. A *triangulation* of a polygon P is a decomposition of P into triangles by a maximal set of non-crossing diagonals (Devadoss et al., 2011).

Here the word *maximal* has the mean that there is no other noncrossing diagonal which is in the set of triangulations of a geometric figure. Figure 2.3 shows three different triangulations of the polygon. There are several questions asked for triangulation of the polygon.

- What is the number of different triangulations of a given polygon?
- How many triangles consist each triangulation of a given polygon?
- Does every polygon always have a triangulation?
- Must each polygon have at least one diagonal?
- We start with the last question.

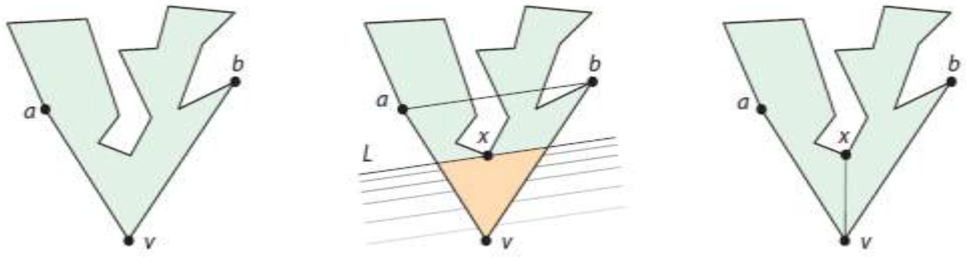


Figure 2.2. Finding a diagonal of a polygon through sweeping [17, pp.4].

Lemma 2. (Devadoss et al., 2011) *Every polygon with more than three vertices has a diagonal.*

Proof. Let v be the lowest vertex of P; if there are several, let v be the rightmost. Let a and b be the two neighboring vertices to v. If the segment ab lies in P and does not otherwise touch ∂P, it is diagonal. Otherwise, since P has more than three vertices, the closed triangle formed by a, b, and v contains at least one vertex of P. Let L be a line parallel to segment ab passing through v. Sweep this line from v parallel to itself upward toward ab; see Figure 1.4. Let x be the first vertex of the closed triangle abv, different from a, b, or v, that L meets along this sweep. The (shaded) triangular region of the polygon below line L and above v is empty of vertices of P. Because vx cannot intersect ∂P except at v and x, we see that vx is diagonal.

Theorem 3. (Devadoss et al., 2011) *Every polygon has a triangulation.*

Proof. We prove this by induction on the number of vertices n of the polygon P. If $n = 3$, then P is a triangle and we are finished. Let $n > 3$ and assume the theorem is true for all polygons with fewer than n vertices. Using Lemma 1.2, find a diagonal cutting P into polygons P_1
and P_2. Because both P_1 and P_2 have fewer vertices than n, P_1 and P_2 can be triangulated by the induction hypothesis. By the Jordan curve theorem (Theorem 1.1), the interior of P_1 is in the exterior of P_2, and so no triangles of P_1 will overlap with those of P_2. A similar statement holds for the triangles of P_2. Thus P has a triangulation as well.

We know that every polygon has at least one triangulation. Next, we show that the number of triangles in any triangulation of a fixed polygon is the same. The proof is essentially the same as that of Theorem 1.4, with more quantitative detail.

Theorem 4. (Devadoss et al., 2011) *Every triangulation of a polygon P with n vertices has $n - 2$ triangles and $n - 3$ diagonals.*

Proof. We prove this by induction on n. When $n = 3$, the statement is trivially true. Let $n > 3$ and assume the statement is true for all polygons with fewer than n vertices. Choose a diagonal d joining vertices a and b, cutting P into polygons P_1 and P_2 having n_1 and n_2 vertices, respectively. Because a and b appear in both P_1 and P_2, we know $n_1 + n_2 = n + 2$. The induction hypothesis implies that there are $n_1 - 2$ and $n_2 - 2$ triangles in P_1 and P_2, respectively. Hence P has $(n_1 - 2) + (n_2 - 2) = (n_1 + n_2) - 4 = (n + 2) - 4 = n - 2$ triangles. Similarly, P has $(n_1 - 3) + (n_2 - 3) + 1 = n - 3$ diagonals, with the +1 term counting d.

In many algorithms and proofs, a special triangle should be included which is often used in the start of recursion or initial induction. The place of these special triangles in computational geometry is often used "*ears*". Three consecutive vertices a, b, c form an ear of a polygon if ac is a diagonal of the polygon. The vertex b is called the ear tip.

Corollary 5. (Devadoss et al., 2011) *Every polygon with more than three vertices has at least two ears.*

Proof. Consider any triangulation of a polygon P with $n > 3$ vertices, which by Theorem 1.4 partitions P into $n - 2$ triangles. Each triangle covers at most two edges of ∂P. Because there are n edges on the boundary of P but only $n - 2$ triangles, by the pigeonhole principle at least two triangles must contain two edges of P. These are the ears.

The proof of Theorem 4 leads to one approach for the triangulation of polygons. One diagonal is detected at each step and the problem is reduced to a smaller problem. If the polygon has n vertices, then this step has complexity $O(n)$. After this step, the problem can be reduced to a problem by $n - 1$, so the total complexity is $O(n^2)$. In some special cases,

for example, for a convex polygon, triangulation is trivial - for a convex polygon, it can be performed in time $O(n)$.
The idea, then, may be to first decompose it into convex parts and then apply triangulation to them. However, it is not easy to decompose a polygon into convex parts and, instead, the polygon will be decomposed into so-called monotone parts.

If the point set in general position is in convex position with all points on the hull then the number of triangulations of these points is the Catalan number (Saracevic, 2019).

2.2 Voronoi Diagram

A Voronoi diagram is the computational geometry concept that represents a partition of the given space onto regions, with bounds determined by distances to a specified family of objects. For example, Voronoi diagrams are useful in developing the system that needs to quickly respond to the customer which restaurant is closest to it at that moment. Many natural and social phenomena can be described and explained in terms of Voronoi diagrams: in anthropology, areas influenced by culture, in crystallography, the structure of crystals and metals, in ecology, competition between plants, in the economy, models of the market, etc (Janičić, 2016).

Knut's formulation: there are n posts in the city and each user uses the mail closest to him (for simplicity, it is assumed that the travel time to the mail is proportional to the Euclidean distance). The question is what an area served by a post looks like. If a new post is to be opened, where would it best be located? etc.

Definition 2. Let $P = \{p_1, p_2, \ldots , p_n\}$ be a set of points in the two-dimensional Euclidean plane. These are called the *sites*. Partition the plane by assigning every point in the plane to its nearest site. All those points assigned to p_i form the *Voronoi region* $V(p_i)$. $V(p_i)$ consists of all the points at least as close to P_i as to any other site:

$$V(p_i) = \{x: |p_i - x| \leq |p_j - x|, \forall j \neq i\}.$$

This set is closed. Some points do not have a unique nearest site or nearest neighbor. The set of all points that have more than one nearest neighbor form the Voronoi diagram V(*P*) for the set of sites (O'Rourke, 1997). A Voronoi diagram is a set of branches and vertices of decomposition. Then when it is said that the Voronoi diagram is connected - it is meant that the set of branches is connected. The Voronoi diagram provides a wealth of useful information. For example, if the two areas are adjacent, then is expected the appropriate sites are to compete for clients in the border region (Janičić, 2016).

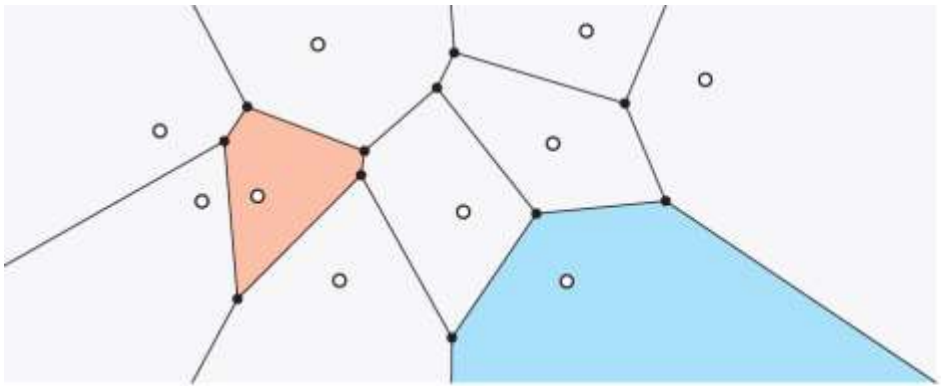


Figure 2.3 The Voronoi diagram for 10 sites (Devadoss et al., 2011, pp. 99).

2.3 Delaunay Triangulation

In mathematics and computational geometry, a Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles (https://en.wikipedia.org/). In 1934

Delaunay proved that when the dual graph is drawn with straight lines, it produces a planar triangulation of the Voronoi sites *P* (if no four sites are co-circular), now called the Delaunay triangulation V(*P*) (O'Rourke, 1997). Delaunay triangulation and Voronoi diagram are dual structures. Generally, it is not obvious that using straight lines in the dual would avoid crossings in the dual; the dual segment between two sites does not necessarily cross the Voronoi edge shared between their Voronoi regions.

The circumcenters of Delaunay triangles are the vertices of the Voronoi diagram. In the 2D case, the Voronoi vertices are connected via edges, that can be derived from adjacency-relationships of the Delaunay triangles: If two triangles share an edge in the Delaunay triangulation, their circumcenters are to be connected with an edge in the Voronoi tesselation (https://en.wikipedia.org/).

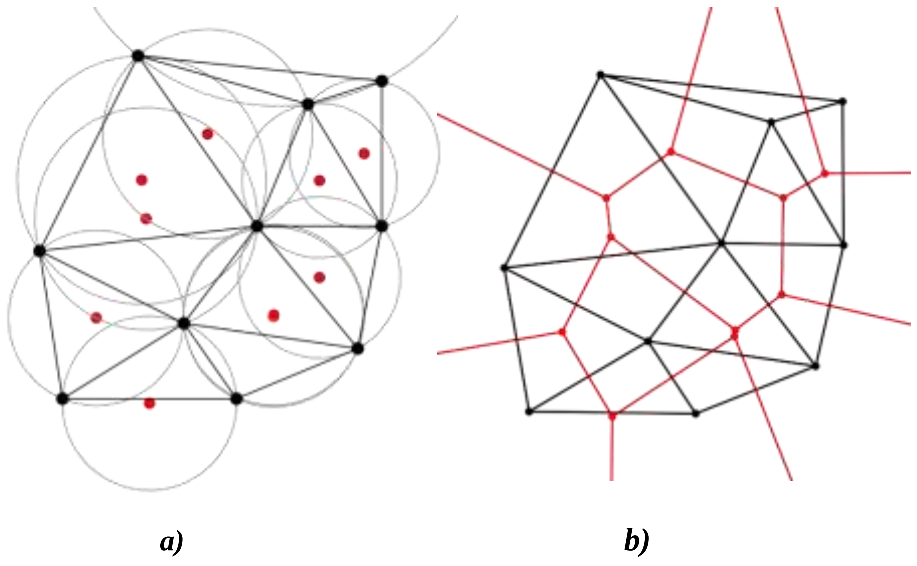


Figure 2.4 *a*) The Delaunay triangulation with all the circumcircles and their centers (in red).

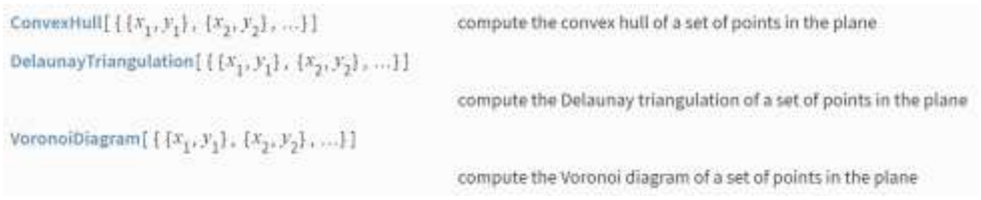
b) Connecting the centers of the circumcircles produces the Voronoi diagram (in red).

3. IMPLEMENTATION OF DELAUNAY TRIANGULATION AND VORONOI DIAGRAM IN MATHEMATICA

In this section, we give three different examples of implementations of the Delaunay triangulation and Voronoi diagram. The Computational Geometry package in Mathematica provides functions for solving these and related problems in the case of planar points and the Euclidean distance metric.

- *The optimization problem of Euclidean minimum spanning trees, Delaunay triangulations, and Voronoi diagrams.*

Computational geometry functions.



ConvexHull[{ $\{x_1, y_1\}$, $\{x_2, y_2\}$, ...}]	compute the convex hull of a set of points in the plane
DelaunayTriangulation[{ $\{x_1, y_1\}$, $\{x_2, y_2\}$, ...}]	compute the Delaunay triangulation of a set of points in the plane
VoronoiDiagram[{ $\{x_1, y_1\}$, $\{x_2, y_2\}$, ...}]	compute the Voronoi diagram of a set of points in the plane

The convex hull of a set *S* is the boundary of the smallest set containing *S*. The Voronoi diagram of *S* is the collection of nearest neighborhoods for each of the points in *S*. For points in the plane, these neighborhoods are polygons. The Delaunay triangulation of *S* is a triangulation of the points in *S* such that no triangle contains a point of *S* in its circumcircle. This is equivalent to connecting the points in *S* according to whether their neighborhood polygons share a common side (https://reference.wolfram.com).

```
<<ComputationalGeometry`

data2D={{5.4,13},{6.7,15.25},{6.9,12.8},{2.1,11.1},{9.5,14.9},
{13.2,11.9}, {10.3,12.3},{6.8,9.5},{3.3,7.7},{0.6,5.1},{5.3,2.4},
{8.45,4.7},{11.5,9.6}, {13.8,7.3},{12.9,3.1},{11,1.1}};
```

`convexhull=ConvexHull[data2D]` (This gives the indices of the points lying on the convex hull in counterclockwise order.)

`ConvexHull[{{0,0},{1,0},{0,0},{2,0},{1,1}}]` (Duplicate points are ignored.)

`(Shallow[#,{5,6}]&)[delval=DelaunayTriangulation[data2D]]`
`{vorvert,vorval}=VoronoiDiagram[data2D];` (This gives the counterclockwise vertex adjacency list for each point in the Delaunay triangulation. For example, the entry {1,{4,3,2}} indicates that the first point in data2D is connected in counterclockwise order to the fourth, third, and second points.)

While Delaunay triangulation need only specify the connections between points, Voronoi diagram must specify both a set of diagram vertices and the connections between those vertices. Another difference between the two functions is that while a triangulation consists of segments, a diagram consists of both segments and rays (https://reference.wolfram.com). For example, in the case of a Voronoi diagram, points in the interior of the convex hull will have nearest neighborhoods that are closed polygons, but the nearest neighborhoods of points on the convex hull will be open.

These considerations make the output of Voronoi diagram more complex than that of Delaunay triangulation. The diagram is given as a list of diagram vertices followed by a diagram vertex adjacency list (https://reference.wolfram.com). The finite vertices of the diagram are listed first in the vertex list. The vertices lying at infinity have head Ray and are listed last.

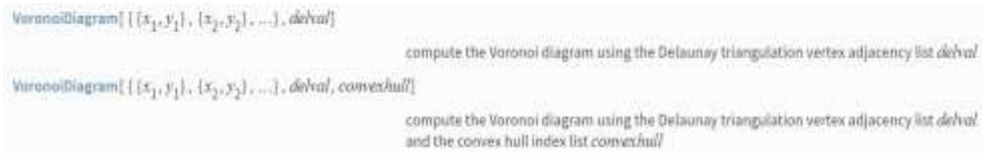
`{First[vorvert],Last[vorvert]}` (This assigns the list of Voronoi diagram vertices to vorvert and the Voronoi diagram vertex adjacency list to vorval.)

`vorval//Short` (Each entry in vorval gives the index of a point in data2D followed by a counterclockwise list of the Voronoi diagram vertices that comprise the point's nearest neighborhood polygon.)

vorval[[1,2]] (Here is the Voronoi polygon vertex adjacency list for the first point in data2D.)

vorvert[[%]] (This selects the coordinates of the polygon vertices from vorvert. The first two vertices have head List, while the last two have head Ray. Thus, the Voronoi polygon associated with the first point in data2D is open and is defined by a segment and two rays.)

Computing the Voronoi diagram using the Delaunay triangulation and the convex hull.

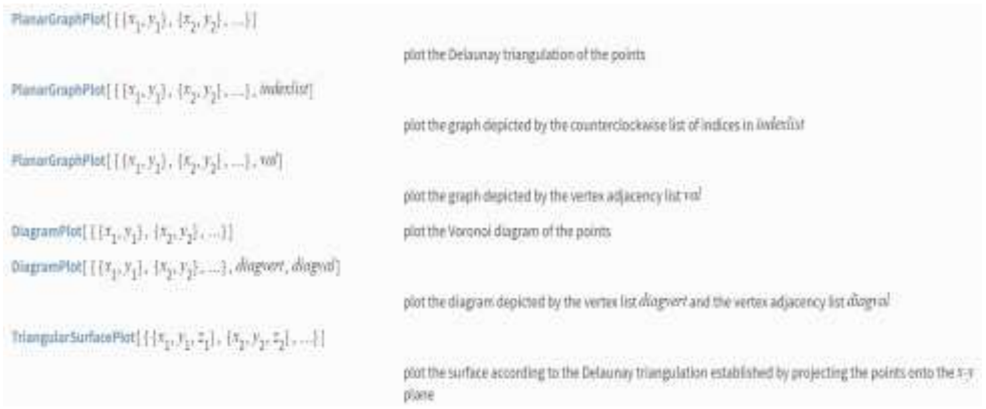


VoronoiDiagram[{{x_1,y_1},{x_2,y_2},…},*delval*]	compute the Voronoi diagram using the Delaunay triangulation vertex adjacency list *delval*
VoronoiDiagram[{{x_1,y_1},{x_2,y_2},…},*delval*, *convexhull*]	compute the Voronoi diagram using the Delaunay triangulation vertex adjacency list *delval* and the convex hull index list *convexhull*

VoronoiDiagram[data2D,delval]; (This computes the Voronoi diagram of data2D more efficiently by making use of the Delaunay triangulation vertex adjacency list)

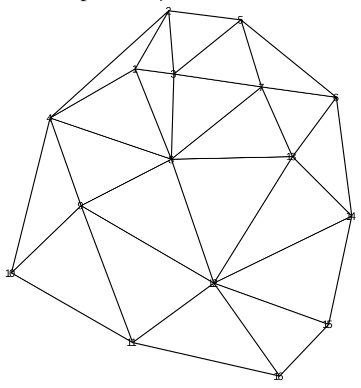
VoronoiDiagram[data2D,delval,convexhull]; (Here the Voronoi diagram is computed using both the Delaunay triangulation and the convex hull.)

Computational geometry plotting functions.

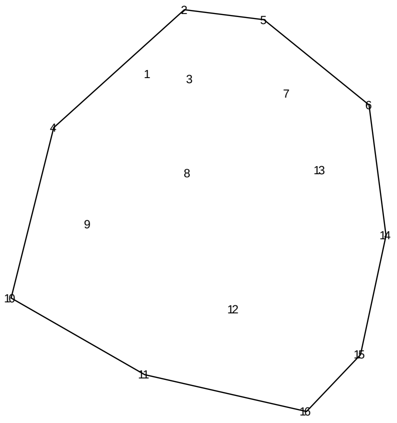


PlanarGraphPlot[{{x_1,y_1},{x_2,y_2},…}]	plot the Delaunay triangulation of the points
PlanarGraphPlot[{{x_1,y_1},{x_2,y_2},…},*indexlist*]	plot the graph depicted by the counterclockwise list of indices in *indexlist*
PlanarGraphPlot[{{x_1,y_1},{x_2,y_2},…},*val*]	plot the graph depicted by the vertex adjacency list *val*
DiagramPlot[{{x_1,y_1},{x_2,y_2},…}]	plot the Voronoi diagram of the points
DiagramPlot[{{x_1,y_1},{x_2,y_2},…},*diagvert*, *diagval*]	plot the diagram depicted by the vertex list *diagvert* and the vertex adjacency list *diagval*
TriangularSurfacePlot[{{x_1,y_1,z_1},{x_2,y_2,z_2},…}]	plot the surface according to the Delaunay triangulation established by projecting the points onto the *x*-*y* plane

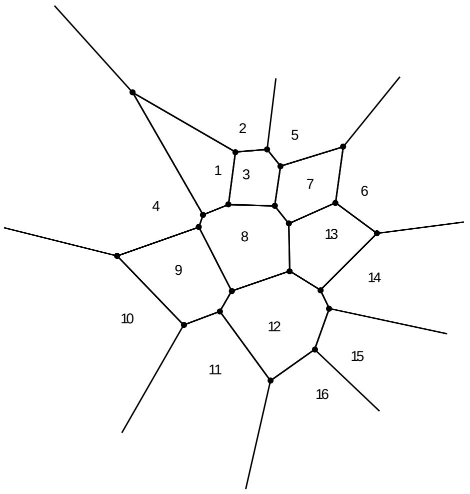
PlanarGraphPlot[data2D] (The default of PlanarGraphPlot is a plot of the Delaunay triangulation of the points.)



PlanarGraphPlot[data2D,convexhull] (This plots the convex hull of the points.)

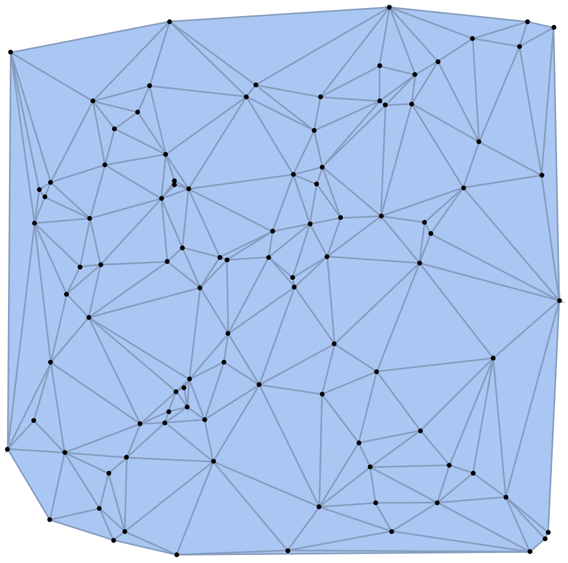


DiagramPlot[data2D] (The default of DiagramPlot is a plot of the Voronoi diagram of the points.)



- *A* 2D *Delaunay mesh from a list of points:*

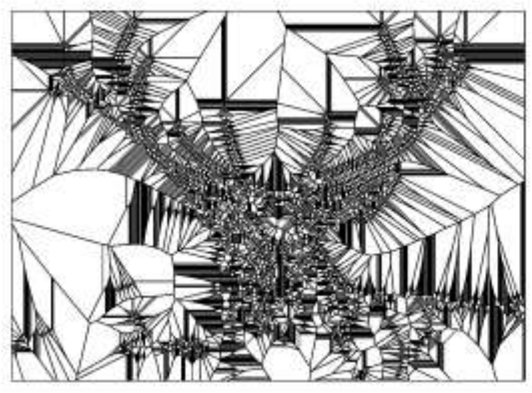
```
pts=RandomReal[{-1,1},{100,2}];
ℛ=DelaunayMesh[pts];
HighlightMesh[ℛ,Style[0,Black]]
```

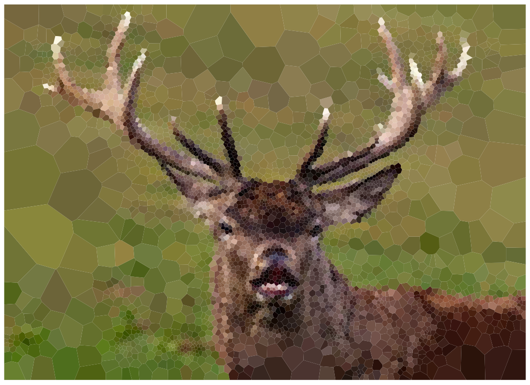


- *Delaunay mesh and Voronoi diagram of given image:*

img=

```
edges=EdgeDetect[img,5]
imgBounds=Transpose[{{0,0},ImageDimensions[img]}];
vm=DelaunayMesh[ImageValuePositions[edges,White],i
mgBounds]
vml=NestList[DelaunayMesh[Mean@@@MeshPrimitive
s[#,2],imgBounds]&,vm,3]
Graphics[Table[{RGBColor[ImageValue[img,Mean@@p]
],p},{p,MeshPrimitives[Last[vml],2]}]]
```





4. CONCLUSION

An exhaustive list of Voronoi diagram applications today we see in programming, game development, and cartography. The numerous of the navigation systems in the developed game engines are based on the Voronoi diagram. The various geolocation software uses Voronoi diagrams. Geolocation reference systems use the Voronoi diagram to determine, for example, the nearest grocery store, for various search and analysis of the location.

Any geographical diagrams showing the distribution of something can be clearly illustrated with the help of colored Voronoi diagrams. This diagram made the transition of the needed indicator (for example, temperature) to be visible. Also, it can make various filter-photo handlers using the Voronoi diagram, getting some kind of mosaic.

There are many ideas of using the Voronoi diagram in architecture and design since it itself is a beautiful drawing, a kind of “geometric web”, so there are many cases of using it as one of the main elements of a composition or even a frame all creation.

In archeology, Voronoi polygons are used to map the range of use of tools in ancient cultures and to study the influence of rival centers of trade. In ecology, the body's ability to survive depends on the number of neighbors with whom it must fight for food and light.

Various kinds of grids (and skeletons) of objects in space are constructed using Voronoi diagrams and Delaunay triangulation.

The Voronoi diagrams are used in the combined influence of electric and short-range forces and help to determine the structure of molecules.

5. REFERENCES

1. Devadoss S. L. and Rourke J. Discrete and computational geometry, Princeton University Press, 2011
2. O'Rourke J., Computational Geometry in C, Second Edition, Cambridge University Press, 1997
3. Selimi A., Saračević M., Computational Geometry Applications, Southeast Europe Journal of Soft Computing., Vol.7, No.2, 2018.
4. Selimi A., Saračević M. Catalan Numbers and Applications, *Vision International Refereed Scientific Journal*, Volume 4, Issue 1, pp. 99 – 114, 2019.
5. Janičić P., Računarska Geometrija, Matematički fakultet Univerziteta u Beogradu, 2016
6. Wolfram S. The Mathematica Book, 5th edition. Wolfram Media, (2003).
7. https://www.macworld.com/article/1051472/workbench.html
8. https://reference.wolfram.com/language/tutorial/UsingATextBasedInterface.html
9. http://robotics.caltech.edu/~radford/jmath/
10. https://en.wikipedia.org/wiki/Delaunay_triangulation
11. https://reference.wolfram.com/language/ComputationalGeometry/tutorial/ComputationalGeometry.html