Dynamic and Kinetic Delaunay Triangulation in 2D and 3D: A Survey

Farid Karimipour

Institute for Geoinformation and Cartography, Vienna University of Technology

Gusshausstr. 27-29, A-1040 Vienna, Austria

karimipour@geoinfo.tuwien.ac.at

Abstract. Delaunay Triangulation (DT) is one of the most fundamental data structures in computational geometry. It is well known in geosciences for many years to model the real world objects. To extend its applications for simulating the real world processes, however, this data structure must support dynamic point sets (where the points are added or deleted) and kinetic point sets (where the position of the points vary over time). This paper reviews the algorithms introduced in the literature to construct the Delaunay triangulation of 2D and 3D dynamic and kinetic points.

Keywords. Delaunay triangulation, Voronoi diagram, Spatial data structures, Dynamic points, Kinetic and Moving points.

1. Introduction

Delaunay Triangulation (DT) is one of the most fundamental data structures in computational geometry. This structure is commonly used in a large set of applications, from computer graphics, visualization, computer vision, robotics, and image synthesis to mathematical and natural sciences (Cignoni et al., 1998). Delaunay triangulation of a set of points is the partitioning of the space into triangles that satisfies the empty circum-circle property: the circum-circle of each triangle does not contain any other point of the point set. This structure is defined for points of any dimension. The computation of the Delaunay triangulation is one of the classical problems of computational geometry. Many algorithms were proposed to construct the Delaunay triangulation of a set of points of different dimensions (Bowyer, 1981; Brown, 1979; Cignoni et al., 1998; Dwyer, 1991; Edelsbrunner and Seidel, 1986; Edelsbrunner and Shah, 1992; Field, 1986; Fortune, 1987; Joe, 1991; Lawson, 1977; Maus, 1984; Mucke, 1988; Tanemura et al., 1983; Watson, 1981).

Delaunay Triangulation is well known in geosciences for many years (Karimipour et al., 2010). It is the basic data structure for many geoscientific applications such as terrain modeling, spatial interpolation and geological mapping problem. It is also widely used in 3D geoscientific modeling. ‘‘3D Delaunay triangulation is used in many geoscientific applications that collect data about spatial objects and domains such as features of the solid earth (aquifers), oceans (currents) or atmosphere (weather fronts), which fill 3D space’’(Lattuada and Raper, 1995). Furthermore, there are several applications in geosciences for which constructing the 3D Delaunay triangulation is the basis, e.g., surface modeling, iso-surface extraction (Ledoux and Gold, 2007) and reconstruction of 3D complex geological objects (Yong et al., 2004).

The problem of constructing the Delaunay triangulation for a set of points that vary over time has been the subject of several studies. Although this variation is mostly considered as a research interest in computational geometry, it is a practical requirement to use Delaunay triangulation in applications that simulate the real world processes (Ledoux, 2008; Mostafavi, 2002; Mostafavi and Gold, 2004). For example, there is an approach for modeling of fluid flow that uses the Free-Lagrange method (FLM) (Fritts et al., 1985): the flow is approximated by a set of discrete points, called particles. Each particle has a mass and a velocity and is allowed to move freely and interact. The Voronoi diagram, which is the dual structure of Delaunay triangulation and tessellates the space based on a set of points, has been used for tessellation of the space in this modeling. This structure must be modified as the particles move. Earlier implementations of the FLM rebuilt the whole structure at each step of the process, thus they were very slow (Ledoux, 2008). However, by using a method that updates the Delaunay triangulation locally, this simulation is done faster and more efficient (Mostafavi and Gold, 2004).

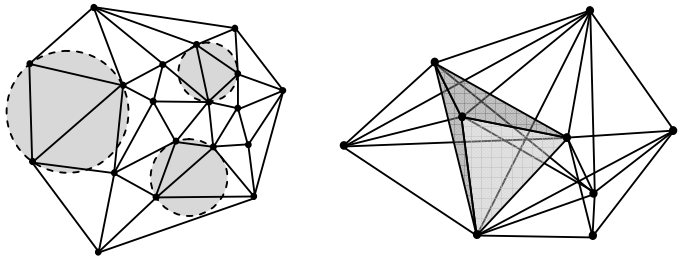
This paper reviews the methods and algorithms introduced in the literature to construct the Delaunay triangulation of 2D and 3D points that vary over time. We use the term 'dynamic' where new points are inserted or some of the points are deleted from the point set; and the term 'kinetic' or 'moving' is used where the position of the points change over time, i.e., the points are moving.

The paper is structured as follows: Section 2 introduces the concept of Delaunay triangulation in more details and reviews the definitions and notions that will be used through the rest of the paper. Sections 3 and 4, respectively, review the methods and algorithms proposed in the literature to construct the Delaunay triangulation of dynamic and kinetic points in two and three dimensional spaces. Finally, Section 5 concludes the paper.

As mentioned earlier, Delaunay triangulation is defined for any dimension. However, we limit the discussion to 2D and 3D, which are of interest in geosciences and their geometrical illustrations are possible. Moreover, most of the concepts and algorithms discussed are firstly introduced by describing their 2D counterparts, because readers are often more familiar with these; and most figures are shown for the 2D case as it is much simpler to understand.

2. Delaunay Triangulation: Definition and Related Concepts and Notions

Given a point set $\boldsymbol{P}$ in the plane, the Delaunay triangulation is a particular triangulation of the points in $\boldsymbol{P}$, which satisfy the empty circum-circle property: the circum-circle of each triangle does not contain any other point $p \in \boldsymbol{P}$. This structure for a set of 3D points is the tetrahedralization of the points in which the circum-sphere of each tetrahedron does not contain any other point of the point set. Figure 1 shows Delaunay triangulation of some 2D and 3D points.



(a) (b)

Figure 1. 2D and 3D Delaunay triangulations (a) Some of the circum-circles are drawn (b) One of the tetrahedra is highlighted.

There are several algorithms to construct the Delaunay triangulation of a set of points in different dimensions. Based on the paradigm used, they can be classified into incremental (Bowyer, 1981; Edelsbrunner and Shah, 1992; Field, 1986; Joe, 1991; Lawson, 1977; Mucke, 1988; Watson, 1981), divide and conquer (Cignoni et al., 1998), and sweepline (Fortune, 1987) algorithms. There are also some other algorithms such as wrapping (Dwyer, 1991; Maus, 1984; Tanemura et al., 1983) and convex hull based (Brown, 1979; Edelsbrunner and Seidel, 1986) algorithms. For more information and comparison, see (Su and Drysdale, 1997).

2.1. Voronoi Diagram (VD)

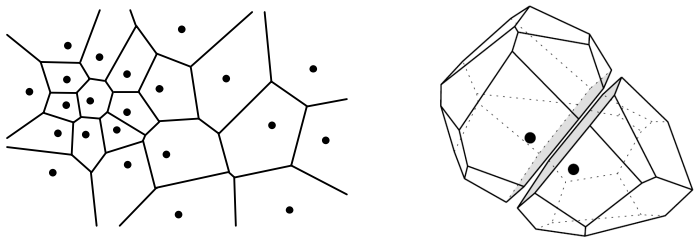
Related to Delaunay triangulation, the Voronoi Diagram (VD) of a set of points is defined as follows: Let **P** be a set of points in an *n*-dimensional Euclidean space $\boldsymbol{R}^n$. The Voronoi cell of a point $p \in \boldsymbol{P}$, called $\boldsymbol{V}_p$(P), is the set of points $x \in \boldsymbol{R}^n$ that are closer to p than to any other point in $\boldsymbol{P}$:

$$\boldsymbol{V}_p(\boldsymbol{P}) = \{x \in \boldsymbol{R}^n \mid \|x-p\| \leq \|x-q\|,\ q \in \boldsymbol{P},\ q \neq p\} \tag{1}$$

The union of the Voronoi cells of all points $p \in \boldsymbol{P}$ form the Voronoi diagram of $\boldsymbol{P}$, noted as $\boldsymbol{VD}(\boldsymbol{P})$:

$$\boldsymbol{VD}(\boldsymbol{P}) = \bigcup \boldsymbol{V}_p(\boldsymbol{P}),\ p \in \boldsymbol{P} \tag{2}$$

Figure 2 shows 2D and 3D examples. The VD is one of the most important spatial structures in sciences, because it is very simple and used in many real-world applications. For an exhaustive surveys on Voronoi diagrams and their applications, see (Aurenhammer, 1991; Okabe et al., 2000).



(a) (b)

Figure 2. 2D and 3D Voronoi diagrams: (a) VD of a set of points in the plane. (b) Two Voronoi cells adjacent to each other in $\mathbf{R}^3$ (they share the grey face).

Delaunay triangulation and Voronoi diagrams are dual structures: the center of circum-circles (-spheres) of Delaunay triangulation are the Voronoi vertices; and joining the adjacent generator points in a VD yield their DT. Therefore, having constructed one structure, the other one can be extracted automatically (Figure 3). This duality is very useful, because construction, manipulation and storage of the VD is more difficult than DT, so all the operations can be performed on DT, and the VD extracted on demand.

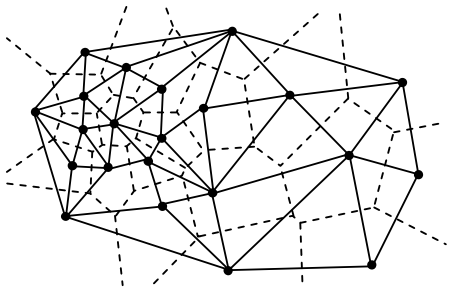


Figure 3. Duality of Delaunay triangulation (solid lines) and Voronoi diagrams (dashed lines)

2.2. Points in general position

Generally, a set of points is said to be in general position when the distribution of these points does not create any ambiguity in the structures derived from the points. For DT and VD, a set of n-dimensional points are in general position if no n+1 points lie on the same hyperplane and no n+2 points lie on an n-sphere. In this paper, we suppose that the points are in general position, otherwise it is explicitly mentioned.

2.3. Circum-circle property

Given a triangle $T = < a, b, c >$ and a point p, the circum-circle property is satisfied, if the point p does not lie in the circum-circle of the triangle T. Its extension to 3D, called circum-sphere property, estates that the point p does not lie in the circum-sphere of the tetrahedron $T = < a, b, c, d >$. The following determinants are used to test the circum-circle and circum-sphere properties for 2D and 3D cases, respectively (Guibas and Stolfi, 1985):

$$\begin{vmatrix} x_a & y_a & x_a^2+y_a^2 & 1 \\ x_b & y_b & x_b^2+y_b^2 & 1 \\ x_c & y_c & x_c^2+y_c^2 & 1 \\ x_p & y_p & x_p^2+y_p^2 & 1 \end{vmatrix} \qquad \begin{vmatrix} x_a & y_a & z_a & x_a^2+y_a^2+z_a^2 & 1 \\ x_b & y_b & z_b & x_b^2+y_b^2+z_b^2 & 1 \\ x_c & y_c & z_c & x_c^2+y_c^2+z_c^2 & 1 \\ x_d & y_d & z_d & x_d^2+y_d^2+z_d^2 & 1 \\ x_p & y_p & z_p & x_p^2+y_p^2+z_p^2 & 1 \end{vmatrix} \qquad (3)$$

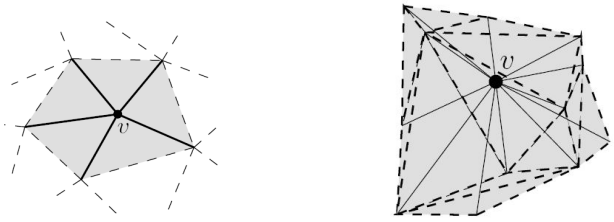
2D **3D**

Positive values for these determinants indicate that the point *p* is inside, while it is outside if the determinants are negative.

2.4. Star, link and ears

For a triangulation, including DT, three concepts *Star*, *Link* and *Ears* are defined. They are introduced in this section, and will be used later.

Star. Let *v* be a vertex in an *n*-dimensional triangulation. As Figure 4 shows, the star of *v*, denoted *star*(*v*), consists of all the simplices that contain *v*; it forms a star-shaped polytope. For example, in two dimensions, all the triangles and edges incident to *v* form *star*(*v*), but notice that the edges and vertices disjoint from *v*, but still part of the triangles incident to *v*, are not contained in *star*(*v*). Also, observe that the vertex *v* itself is part of *star*(*v*), and that a simplex can be part of a *star*(*v*), but not some of its facets.



(a) (b)

Figure 4. The star of a vertex *v* in DT: (a) 2D (b) 3D

Link. The set of simplices incident to the simplices forming *star*(*v*), but `left out' by *star*(*v*), form the link of *v*, denoted *link*(*v*), which is a (*n*-1) triangulation (Figure 5). For example, if *v* is a vertex in a tetrahedralization, then *link*(*v*) is a 2D triangulation formed by the vertices, edges and triangular faces that are contained by the tetrahedra of *star*(*v*), but are disjoint from *v*.

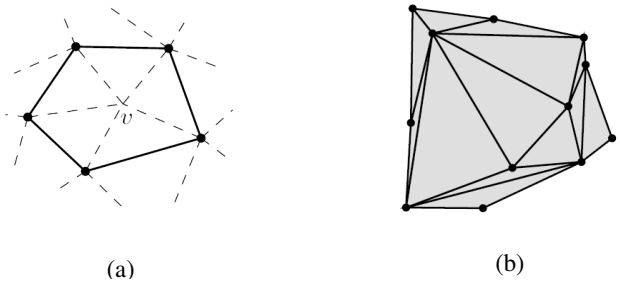


Figure 5. The link of a vertex *v* in DT: (a) 2D (b) 3D

Ear. Let **P** be a simplicial polyhedron, i.e. made up of triangular faces. An ear of **P** is conceptually a potential, or imaginary, tetrahedron that could be used to tetrahedralize **P**. As shown in Figure 6, such a tetrahedron, that does not exist yet, and can be constructed by the four vertices spanning either two adjacent faces, or three faces all sharing a vertex (the vertex has a degree of 3). The former ear is denoted a 2-ear, and the latter a 3-ear. A 3-ear is actually formed by three 2-ears overlapping each other. In practice, a 2-ear can be identified by an edge on **P**, because only two faces are incident to it.

A polyhedron **P** will have many ears, but observe that not every ear is a potential tetrahedron to tetrahedralize **P**, as some adjacent faces form a tetrahedron lying outside **P**. Referring again to Figure 6, a 2-ear *abcd* is said to be valid if and only if the line segment *ad* is inside **P**; and a 3-ear *abcd* is valid if and only if the triangular face *abc* is inside **P**.

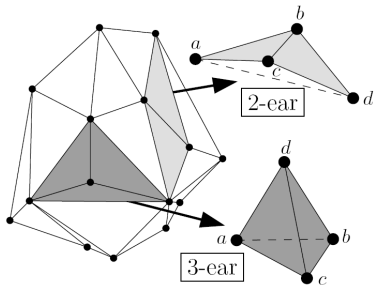


Figure 6. Perspective view of the outside of a polyhedron. Two adjacent triangular faces (e.g., in light grey) form a 2-ear, and three triangular faces incident to the same vertex (e.g., in dark grey) form a 3-ear.

3. Dynamic Delaunay Triangulation

In a *dynamic* set of points, the position of points is fixed, but the number of points may change over time: points are allowed to be inserted to or deleted from the point set. This section reviews the existing algorithms to insert in or delete a vertex from a Delaunay triangulation.

3.1. Insertion

To insert a vertex in a Delaunay triangulation, the incremental methods proposed to construct the DT can be properly used. These methods insert a vertex to an existing Delaunay triangulation and update the data structure locally. *Flipping* and *Bowyer-Watson* algorithms are the two existing methods.

3.1.1. Flipping algorithm

This algorithm was firstly introduced by Lawson (Lawson, 1977). It is based on the fact that there are two possible triangulations for four points in 2D, only one of which satisfies the circum-circle property (Figure 7). Replacing one configuration with the other one is called *flipping*. In 2D case, it is called *flip22*, because there are two triangles before and after the flip operation.

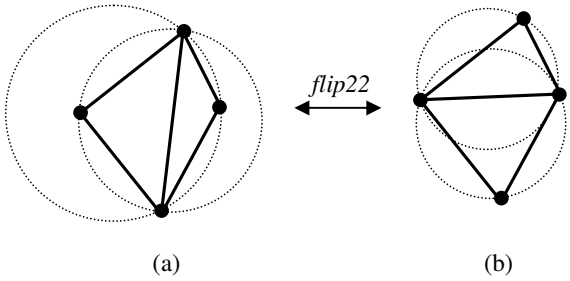


Figure 7. Two possible triangulations for four 2D points. Triangulation in (b) satisfies the circum-circle property.

To insert a vertex in a 2-dimensional DT using the flipping algorithm, the triangle of the DT that contains the new vertex is detected (Figure 8.b) and it is replaced with three new triangles that passes through the new vertex (Figure 8.c). The new triangles are pushed in a stack and each time, an element of the stack is checked against the circum-circle property. If this property is not satisfied, then the triangle and its adjacent are flipped and the new triangles are pushed in the stack. This process repeats till there is no element left in the stack (Figure 8.d to 8.h).

Joe (1991) extended the flipping algorithm to 3D. While this extension is based on generalizing the concept of flipping, this concepts is completely different in 3D (Joe, 1989; Joe, 1991; Lawson, 1986). To tetrahedralize five 3D points, there are two possible solutions: one has two tetrahedra and the other has three (Figure 9). Flipping between the two configurations are called *flip23* and *flip32*, regarding the number of tetrahedra exist before and after the flip operation. Moreover, according to the geometry of a tetrahedron in the DT with its adjacent, it is not always possible to perform a flip (in such cases, the flip is performed by a later element in the stack). For more information, see (Edelsbrunner and Shah, 1992; Ledoux, 2007; Shewchuk, 2003).

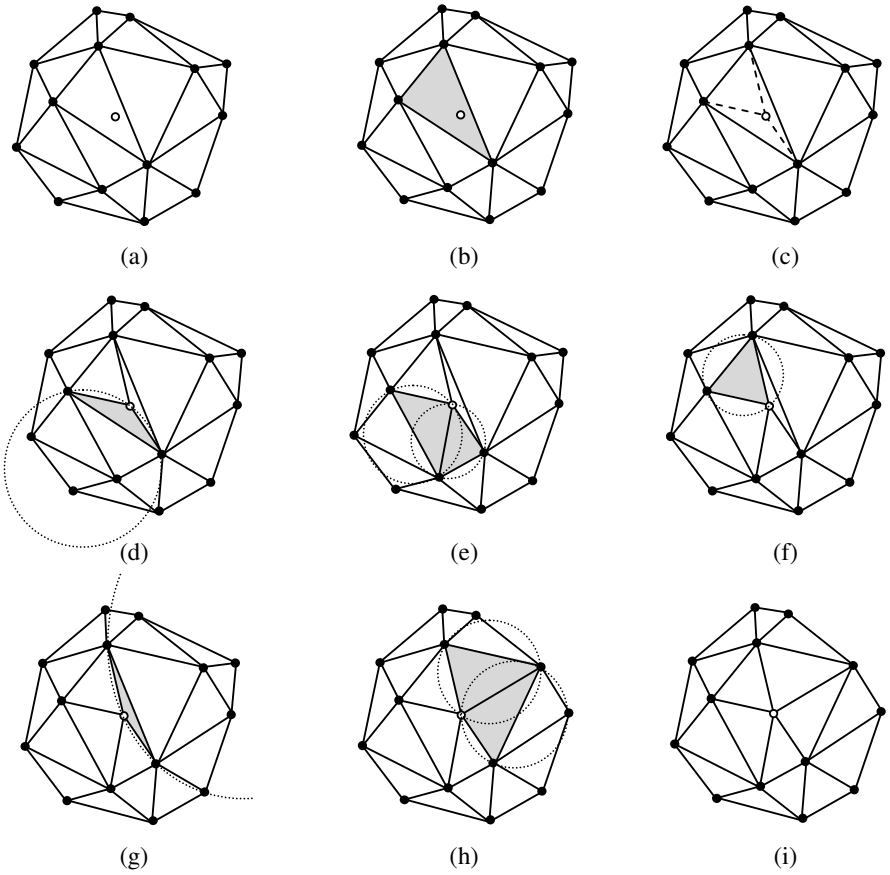


Figure 8. Flipping algorithm to insert a vertex in a DT: (a) Initial DT and the new vertex. (b) Detecting the triangle that contains the new vertex and (c) inserting the new vertex into it. (d) to (h) Checking the circum-circle property and applying flipping if required. (i) new DT that contains the inserted vertex (Ledoux, 2007).

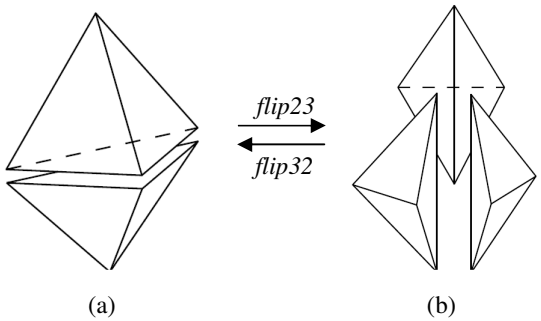


Figure 9. Two possible tetrahedralizations for five 3D points.

3.1.2. Bowyer-Watson algorithm

To insert a vertex in a 2-dimensional DT using the Bowyer-Watson algorithm, all the triangles that violate the circum-circle property, i.e., whose circum-circle contains the new vertex (Figures 10.a), are deleted from the construction (Figures 10.b). This creates a hole, which is filled by new triangles that are created by joining the new vertex to each edge of the boundary of the hole (Figures 10.c) (Bowyer, 1981; Kanaganathan and Goldstein, 1991; Watson, 1981).

For 3-dimensional DT, the procedure is completely the same: after each insertion, all the tetrahedra whose circum-sphere contains the new vertex are deleted, and the hole is filled by new tetrahedra that are created by joining the new vertex to each triangle of the boundary of the hole (Field, 1986; Watson, 1981). Although it is much more easier than flipping algorithm to implement, it is very sensitive to round-off errors, and thus it is not robust (Ledoux, 2006).

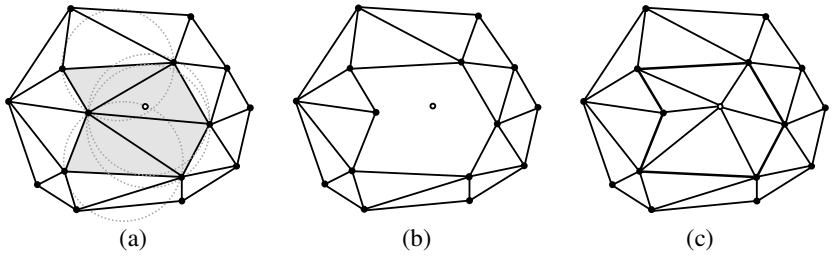


Figure 10. Inserted vertex, indicated as white, is added to DT: (a) and (b) all triangles whose circum-circle contains the new vertex are detected and deleted (c) Hole is filled by new triangles, which are created by joining the new vertex to each edge of the boundary of the hole (Ledoux 2006)

3.2. Deletion

Deleting a vertex *v* from DT can be considered as the inverse problem of inserting a vertex in a DT. General understanding is that the vertex *v* and all the triangles incident to *v* are removed and the created *hole* is re-triangulated (Figure 11).

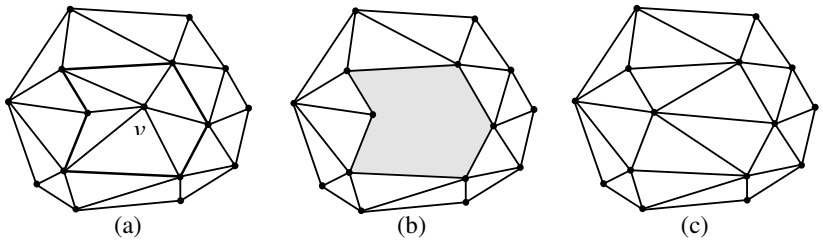


Figure 11. Deleting a vertex *v* from a DT (a) DT Before deletion (b) The hole created by deleting the incident triangles (c) Re-triangulating the hole (Ledoux et al., 2005)

Heller (1990) proposed an algorithm to delete a vertex from a 2-dimensional DT. In his algorithm, the ears of the vertex *v* (triple neighboring vertices that form a potential or imaginary triangle) are examined in anti-clockwise order (Figure 12.b) and the one with the smallest circum-circle (Figure 12.c) is flipped with its adjacent triangle with which it share a link edge (Figure 12.d). This reduces the number of neighbors of *v* by one. The process is repeated until only three triangles left (Figure 12.e). Then, *v* is removed and the three triangles merged into one (Figure 12.f) (Heller, 1990).

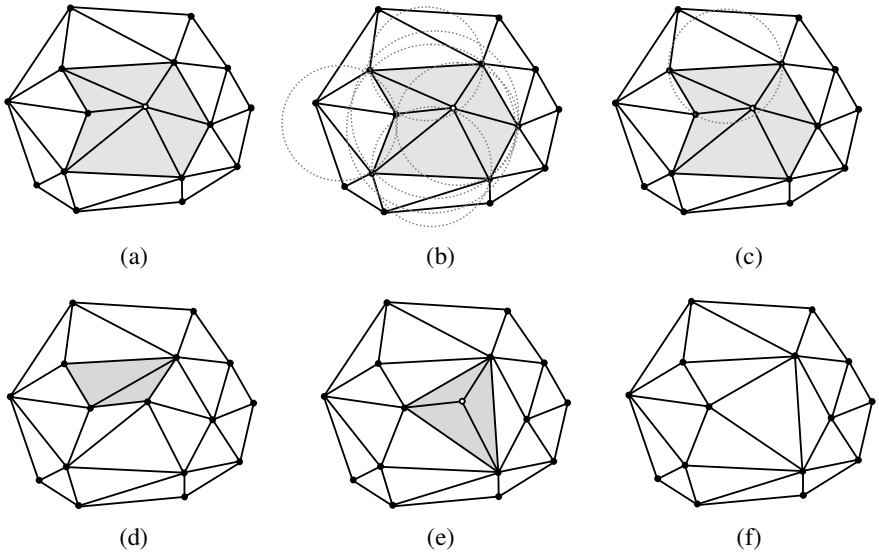


Figure 12. Heller algorithm to delete a vertex, indicated as white, from a DT (a) The initial DT. (b) Circum-circles of the triple neighboring vertices that form an imaginary triangle. (c) and (d) Flipping the imaginary triangle with the smallest circum-circle with its adjacent triangle. (e) Repeating the process until only three triangles left. (f) Removing the vertex and merging the three triangles into one.

Heller assumption was that among all the potential ears, the one with the smallest circum-circle has no other vertex inside and so could become a real triangle. However, Dellivers (2002) showed, through a counter-example, that this assumption is wrong. Instead, he suggested ordering the ears (imaginary triangles) based on the power of the vertex to be removed with respect to that ear. This parameter is computed as follow (Devillers, 2002):

$$power(< a,b,c >, v) = \frac{H(< a,b,c >, v)}{D(a,b,c)} \qquad (4)$$

where

$$D(a,b,c)=\begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}$$

$$H(<a,b,c>,v)=\begin{vmatrix} x_a & y_a & x_a^2+y_a^2 & 1 \\ x_b & y_b & x_b^2+y_b^2 & 1 \\ x_c & y_c & x_c^2+y_c^2 & 1 \\ x_v & y_v & x_v^2+y_v^2 & 1 \end{vmatrix} \quad (5)$$

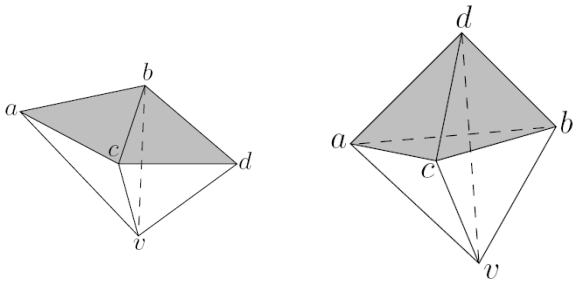
It is proved that Dellivers algorithm works for any dimensions ((Devillers and Teillaud, 2003). However, as we stated earlier in Section 3.1.1, not every polyhedron can be tetrahedralized. Devillers and Teillaud (2003) used perturbations to solve this problem (for more details, see (Devillers and Teillaud, 2003)).

Mostafavi et al. (2003) proposed an algorithm that does not apply any order on the imaginary triangles. Instead, they test the imaginary triangles one by one, and if it is a valid imaginary triangle, it is flipped with its adjacent. An imaginary triangles T=(v_1, v_2, v_3) is invalid if:

- D(v_1, v_2, v_3) is negative. It means that T forms a re-entrant, not an ear.
- D(v_1, v_3, v) is negative, where v is the vertex to be deleted. It means that T encloses v.
- H(<v_1, v_2, v_3>, x) is positive for at least one of the neighboring vertices x. It means that there is, at least, one neighboring vertex that lies inside the circum-circle of T.

Although this algorithm is simpler, it becomes less efficient as the number of neighbors increases. However, this algorithm is equally efficient up to eight neighbors, which is mostly the case. Moreover, point deletion often has the potential to break down in degenerate cases due to the limitations of floating-point precision, but this algorithm was checked against some nasty cases and it worked without any problem (Mostafavi et al., 2003).

To extend this algorithm to 3D, recall that there are two types of ears in 3D: 2-ears and 3-ears. Let ***P*** be a polyhedron that is made up of triangular faces. A 2-ear is formed by two adjacent triangular faces *abc* and *bcd* sharing edge *bc* (Figure 13.a); and a 3-ear is formed by three adjacent triangular faces *abd*, *acd* and *bcd* sharing vertex *d* (Figure 13.b). A 2-ear is valid if and only if the line segment *ad* is inside ***P***; and a 3-ear is valid if and only if the triangular face *abc* is inside ***P***. In the case of the deletion of a vertex *v* in a DT, ***P*** is a star-shaped polyhedron *star*(*v*). An ear of *star*(*v*) is valid if it is convex outwards from *v*.

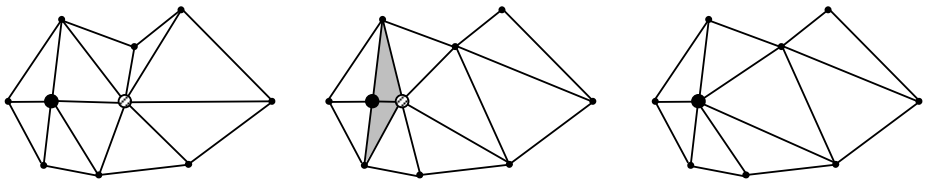


(a) (b)

Figure 13. (a) 2-ear (b) 3-ear

Now, to delete a vertex *v* from a 3-dimensional DT, all the ears of *star*(*v*) are built and stored in a simple list. An ear *e* from the list (any ear) is popped. The ear *e* is flipped if respects these three conditions: *e* is *valid*, *flippable* and *locally Delaunay* (Ledoux et al., 2005). For more details on a flippable tetrahedron, see (Ledoux, 2007). An ear *e* is locally Delaunay if its circum-sphere does not contain any other points on the boundary of *star*(*v*).

Another approach suggested by Schaller and Meyer-Hermann (2004) moves the vertex to be deleted towards its nearest neighbor gradually and update the data structure until the simplices between the two vertices are very flat and can be clipped out of the triangulation without harming its validity. Figure 14 illustrates the idea of the algorithm. The updates are performed using the existing algorithms for kinetic DT, which will be presented in the next section. The main questions to be answered all reduce to the problem of the step size.



(a) (b) (c)

Figure 14. Delete a vertex from a DT: (a) The vertex to be deleted (large hatched point) is moved gradually toward it nearest neighbor (large solid point) and the DT is updated by flipping when required. (b) The movement continues until the inner simplices (shaded region) can be safely deleted. (c) The two vertices are simply merged and the remaining opposing simplices are connected as neighbors.

4. Kinetic Delaunay Triangulation

A *kinetic* or *moving point* is a point whose position changes over time, i.e., it is a function of time:

$$\boldsymbol{P} = (p_1, p_2, \ldots, p_n) = (f_1(t), f_2(t), \ldots, f_n(t)) \qquad (6)$$

4.1. Intuition: Delete and re-insert

Once insertion and deletion of a point have been implemented for a data structure, the intuitionistic approach to handle a moving point is that the point is gradually deleted from the current position and re-inserted at the new position; after each deletion and insertion, the data structure is updated. Although it is a very simple approach, it is computationally expensive, because several unnecessary deletions and insertions are performed. In other words, a point is deleted and re-inserted, no matter if this movement affects the topology of the data structure. This approach nevertheless is an appropriate solution for many applications where the intermediate states are not important (just the start and end states are of interest): the point is deleted from the start and re-inserted at the end.

4.2. Flip-based approach

De Fabritiis and Coveney (2003) modify this approach to move the points in a DT: they gradually move the points toward their destinations. After each movement, the triangle that violate the circum-circle property are detected and flipped. In 2D, each triangle T is checked with all of its neighbors. If the opposite vertex of a neighbor T' lies in the circum-circle of T, then T and T' are flipped and put in a stack to be checked with their new neighbors. This process continues until there is no element left in the stack.

Extension of this method to 3D needs two types of check, because two types of flips (*flip23* and *flip32*) are possible (Schaller and Meyer-Hermann, 2004):

- Each tetrahedron T is checked with its neighbor T' and a *flip23* is performed if the following two conditions are met:
  - The opposite vertex of the neighbor T' lies within the circum-sphere of T.
  - The five union vertices of T and T' form a convex polyhedron.
- Each tetrahedron T is checked with two of its neighbors T_1 and T_2 and a *flip32* is performed if the following two conditions are met:
  - All of the pairs TT_1, TT_2 and T_1T_2 violate the circum-sphere property.
  - T_1 and T_2 are also respective neighbors.

Another extension of this approach to 3D is that instead of performing a sequence of flips on the tetrahedra in order to locally restore the circum-sphere property, the validity of this property is checked for all the tetrahedra. The points for which this property fails are moved back to the preceding position and then "delete and re-insert" is applied (De Fabritiis and Coveney, 2003).

The advantage of the flip-based approach to the "delete and re-insert" is that a simple check is applied on all elements (triangles or tetrahedra here), but further operations (i.e., *flip* and *delete and re-insert*) are applied only when it is required. However, the main drawback is still there: This method uses a fixed time step to move all of the point, no matter if this movement affects the topology of the data structure. Moreover, flips cannot be used to recover from an

invalid triangulation. This becomes an issue when computing a maximum step size for all of the moving vertexes.

4.3. Events-based approaches

Event-based approaches are based on the concept of *topological events*, which is defined as follows:

> "For a data structure *D* consists of moving elements *S*, a topological event *t* is the moment when the movement of elements *S* change the topological structure in *D*".

Based on this concept, to handle moving a point in a data structure, the topology of the data structure is updated at topological events; elsewhere, only the geometry of the data structure is modified, which does not need any computation.

Let p be a vertex in a Delaunay triangulation *DT* and $\boldsymbol{P}$ be the set of its neighboring points, in clock-wise order. Let $\boldsymbol{T}_r$ be the set of opposite triangles (tetrahedra in 3D) of p, i.e., neighbors of incident triangles (tetrahedra in 3D) to p , and $\boldsymbol{T}_i$ be the set of imaginary triangles (tetrahedra in 3D) that could be drawn from three (four in 3D) successive elements of $\boldsymbol{P}$ (Figure 15). Then, the topological events of *DT* caused by the point p are defined as follows (Albers et al., 1998; Ledoux, 2008; Mostafavi, 2002; Roos, 1991):

- If p moves in the circum-circle (-sphere in 3D) of an element of $\boldsymbol{T}_r$ (Figure 16), a flip is performed and the new triangles (tetrahedral in 3D) are updated (i.e., they are checked with their neighbors against the circume-circle (-sphere in 3D) property).
- If p moves out of the circum-circle (-sphere in 3D) of an element of $\boldsymbol{T}_i$ (Figure 17), a flip is performed and the new triangles (tetrahedral in 3D) are updated.

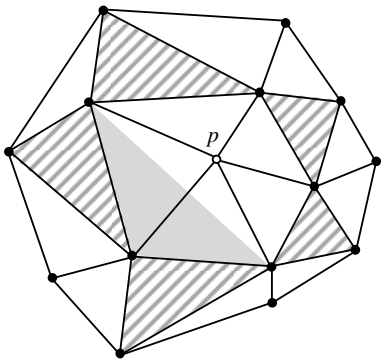
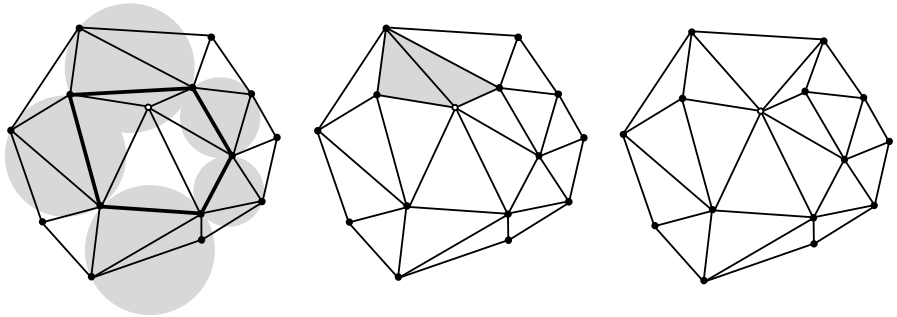
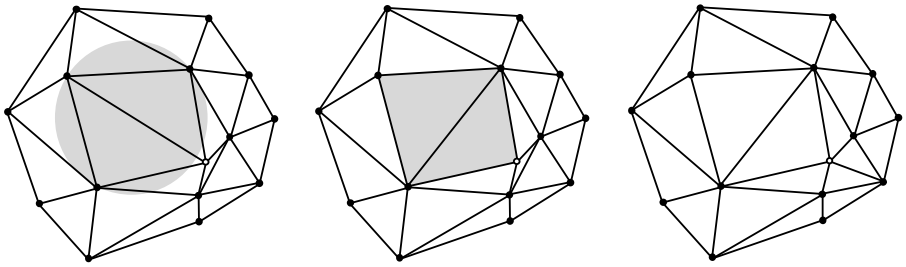


Figure 15. Hashed triangles are the opposite triangles of p. Shaded triangle is one of the imaginary triangles that could be drawn from three successive neighbors of p.



(a) (b) (c)

Figure 16. (a) The white point moves in the circum-circle of an opposite triangle. (b) The two triangles are flipped. (c) Final DT



(a) (b) (c)

Figure 15. (a) The white point moves out of the circum-circle of an imaginary triangle (b) The two triangles are flipped. (c) Final DT

Roos (1991) proposed an algorithm to update a 2-dimensional DT based on the concept of the topological events. All the topological events for all the quadrilaterals (pair of adjacent triangles in DT) are computed and put in a priority queue, sorted according to the time they will arise. The time is computed by finding the zeros of the circum-circle equation developed into a polynomial. Then, the first topological event is popped from the queue, the DT is modified with a *flip22*, and the queue is updated, because the flip has changed locally some triangles. The algorithm continues until there are no topological events left in the queue (Guibas et al., 1992; Roos, 1991; Roos, 1993; Roos and Noltemeier, 1991). Similar algorithms have been proposed in (Bajaj and Bouma, 1990; Imai et al., 1989).

This algorithm has been extended to 3-dimsnioanl DT in (Albers, 1991; Albers et al., 1998; Albers and Roos, 1992). However, it is not very efficient in 3D, because calculating the zeros of the circum-sphere equation cannot be done analytically, as is the case for the circum-circle equation (Gavrilova and Rokne, 2003; Vomacka, 2008). Indeed, the polynomial for the 3-dimensional case has a high degree (8th degree) and iterative numerical solutions must be sought. That results in a much slower implementation, and it could also complicate the update

of the DT when the set of points contains degeneracies. Guibas et al. (2004) proposed a generic framework for handling moving objects that uses fixed precision and exact arithmetic to find the topological events. They claim that using their method to find the zeros of polynomials (circum-sphere) is relatively fast in most cases (Guibas et al., 2004).

Mostafavi (2001) propose a different algorithm and give more implementation details. He focuses on the operations necessary to move a single point *p*, and then explain how to move many points. In this algorithm, the topological events caused by a single point *p* are detected by intersecting the trajectory of the point *p* with the opposite and imaginary circum-circles, which were explained above (Gold, 1990; Gold et al., 1995; Mostafavi, 2002; Mostafavi and Gold, 2004). This algorithm has been equally extended to 3-dimensional DT in (Ledoux, 2008).

5. Conclusions

This paper reviews the methods and algorithms introduced in the literature to construct the Delaunay triangulation of 2D and 3D dynamic point sets (where the points are added or deleted) and kinetic point sets (where the position of the points vary over time). Static DT has been used in geosciences for many years to model the real world objects. However, dynamic kinetic DT are essential and for simulating the real world processes.

Although some of the methods and algorithms introduced were very efficient and smart-designed, there are applications for which the elementary methods works better, in terms of complexity. In other words, the “best” algorithm differs from an application to another.

The current approach in extending spatial analyses to higher dimensions (e.g., 2D dynamic, 2D moving, 3D dynamic and 3D moving) is developing a technical solution to extend spatial analyses to a new multi-dimensional space, with least increase in complexity and speed. These are the parameters to evaluate the efficiency of algorithms in computational geometry. Although there are successful results for this aim (as we saw in the paper), the issue of this approach is that the extension techniques are highly dependent on the specific case studied. It has resulted in developments which cannot be generalized. In other words, a technique used to extend a 2D spatial analysis to a higher dimensional space may not be usable for another spatial analysis, nor to extend the same spatial analysis to another higher dimensional space. The result of following such approaches in the software development stage is recoding each spatial analysis for each dimension. This is not a promising approach to extend all of the required spatial analyses in a practical field of study like geosciences, where several spatial analyses need to be extended. We are working on a principled method to extend 2D spatial analyses to higher dimensions, independent of the analyses at hand, with a minimum amount of recoding (Karimipour and Delavar, 2008; Karimipour et al., 2008; Karimipour et al., 2010; Karimipour et al., 2008; Karimipour et al., 2006).

6. Acknowledgment

This research is supported by the fund from Austrian Marshall Plan Foundation. The author would like to deeply thank Dr. Cyrus Shahabi for his kind hospitality at University of Southern California (USC) and his invaluable contribution and comments on the paper.

References

1. Albers, G. (1991). *Three-Dimensional Dynamic Voronoi Diagrams*, Ph.D. Thesis, University of Wurzburg, Wurzburg, Germany, (In German).

2. Albers, G., Mitchell, J.S.B., Guibas, L.J. and Roos, T. (1998). *Voronoi Diagrams of Moving Points*, *International Journal of Computational Geometry and Applications*, **8**: 365-380.

3. Albers, G. and Roos, T. (1992). *Voronoi Diagrams of Moving Points in Higher Dimensional Spaces*, In Proceedings of the 3rd Scandinavian Workshop On Algorithm Theory (SWAT 92), Helsinki, Finland, Lecture Notes in Computer Science (LNCS), Vol. 621, Springer-Verlag, pp. 399-409.

4. Aurenhammer, F. (1991). *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*, **23**(3): 345-405.

5. Bajaj, C. and Bouma, W. (1990). *Dynamic Voronoi Diagrams and Delaunay Triangulations*, In Proceedings of the 2nd Annual Canadian Conference on Computational Geometry, Ottawa, Canada, pp. 273-277.

6. Bowyer, A. (1981). *Computing Dirichlet Tessellation*, *The Computer Journal*, **24**(2): 162-166.

7. Brown, K.Q. (1979). *Voronoi Diagrams from Convex Hulls*, *Information Processing Letters*, **9**(5): 223–228.

8. Cignoni, P., Montani, C. and Scopigno, R. (1998). *A Fast Divide and Conquer Delaunay Triangulation Algorithm*, *Computer-Aided Design*, **30**(5): 333-341.

9. De Fabritiis, G. and Coveney, P.V. (2003). *Dynamical Geometry for Multiscale Dissipative Particle Dynamics*, *Computer Physics Communications*, **153**: 209-226.

10. Devillers, O. (2002). *On Deletion in Delaunay Triangulations*, *International Journal of Computational Geometry and Applications*, **12**(3): 193-205.

11. Devillers, O. and Teillaud, M. (2003). *Perturbations and Vertex Removal in a 3D Delaunay Triangulation*, In Proceedings of the ACM-SIAM symposium on Discrete Algorithms, Baltimore, USA January 12-14, 2003, pp. 313–319.

12. Dwyer, R.A. (1991). *Higher-dimensional Voronoi Diagrams in Linear Expected Time*, *Discrete and Computational Geometry*, **6**: 343–367.

13. Edelsbrunner, H. and Seidel, R. (1986). *Voronoi Diagrams and Arrangements*, *Discrete & Computational Geometry* **1**: 25–44.

14. Edelsbrunner, H. and Shah, N.R. (1992). *Incremental Topological Flipping Works for Regular Triangulations*, In Proceedings of the 8th Annual Computational Geometry, Berlin, Germany, pp. 43-52.

15. Field, D.A. (1986). *Implementing Watson's Algorithm in Three Dimensions*, In Proceedings of the 2nd Annual Symposium on Computational Geometry, Yorktown Heights, New York, USA, pp. 246-259.

16. Fortune, S. (1987). *A Sweepline Algorithm for Voronoi Diagrams*, *Algorithmica*, **2**: 153-174.

17. Fritts, M., Crowley, W. and Trease, H. (1985). *The Free-Langrange Method*, Lecture Notes in Physics, Vol. 238, Springer-Verlag.

18. Gavrilova, M.L. and Rokne, J. (2003). *Updating the Topology of the Dynamic Voronoi Diagram for Spheres in Euclidean d-Dimensional Space*, *Computer Aided Geometric Design*, **20**: 231-242.

19. Gold, C. (1990). *Spatial Data Structures - the Extension from One to Two Dimensions*, *Mapping and Spatial Modeling for Navigation*, **65**: 11-39.

20. Gold, C.M., Remmele, P.R. and Roos, T. (1995). *Voronoi Diagrams of Line Segments Made Easy*, In In Proceedings 7th Canadian Conference on Computational Geometry, Quebec City, Canada, pp. 223-228.

21. Guibas, L., Karaveles, M. and Russel, D. (2004). *A Computational Framework for Handling Motion*, In Proceedings of the 6th Workshop on Algorithm Engineering and Experiments, New Orleans, USA, January 10, 2004, pp. 129-141.

22. Guibas, L. and Stolfi, J. (1985). *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, *ACM Transactions on Graphics*, **4**(2): 74-123.

23. Guibas, L.J., Mitchell, J.S. and Roos, T. (1992). *Voronoi Diagrams of Moving Points in the Plane*, In Proceedings of the 17th International Workshop, June 17-19, 1991, pp. 113-125.

24. Heller, M. (1990). *Triangulation Algorithms for Adaptive Terrain Modelling*, In Proceedings of the 4th International Symposium on Spatial Data Handling, Zurich, Switzerland, July 23-27, 1990, pp. 163–174.

25. Imai, K., Sumino, S. and H, H.I. (1989). *Geometric Fitting of Two Corresponding Sets of Points*, In Proceedings 5th Annual Symposium on Computational Geometry, Saarbrucken, Germany, ACM Press, pp. 266-275.

26. Joe, B. (1989). *Three-dimensional Triangulations from Local Transformations*, *SIAM Journal on Scientific and Statistical Computing*, **10**(4): 718–741.

27. Joe, B. (1991). *Construction of Three-dimensional Delaunay Triangulations using Local Transformations*, *Computer Aided Geometric Design* **8**: 123–142.

28. Kanaganathan, S. and Goldstein, N.B. (1991). *Comparison of Four-point Adding Algorithms for Delaunay-type 3-Dimensional Mesh Generators*, *IEEE Transaction on Magnetics*, **27**(3): 3444-3451.

29. Karimipour, F. and Delavar, M.R. (2008). *Extension of Spatial Operations for Multi-dimensional GIS*, In: G. Navratil (Ed.) Proceedings of the Colloquium for Andrew U. Frank's 60th Birthday, Vienna, Austria, June 30 - July 1, 2008, Geoinfo Series, Vol. 39, pp. 117-123.

30. Karimipour, F., Delavar, M.R. and Frank, A.U. (2008). *A Mathematical Tool to Extend 2D Spatial Operations to Higher Dimensions*, In: O. Gervasi et al. (Eds.) Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2008), Perugia, Italy, June 30 - July 3, 2008, Lecture Notes in Computer Science (LNCS), Vol. 5072, Springer-Verlag, pp. 153-167.

31. Karimipour, F., Delavar, M.R. and Frank, A.U. (2010). *A Simplex-Based Approach to Implement Dimension Independent Spatial Analyses*, *Journal of Computer and Geosciences (DOI: 10.1016/j.cageo.2010.03.002)*.

32. Karimipour, F., Frank, A.U. and Delavar, M.R. (2008). *An Operation-Independent Approach to Extend 2D Spatial Operations to 3D and Moving Objects*, In: H. Sammet, C. Shahabi and W.G. Aref (Eds.) Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008), Irvine, CA, USA, November 5-7, 2008.

33. Karimipour, F., Rezayan, H. and Delavar, M.R. (2006). *Formalization of Moving Objects Spatial Analyses Using Algebraic Structures*, In: W. Kuhn and M. Rabaul (Eds.) Proceedings of Extended Abstracts of GIScience 2006, Münster, Germany, September 20-23, 2006, IfGI Prints, Vol. 28, pp. 105-111.

34. Lattuada, R. and Raper, J. (1995). *Applications of 3D Delaunay Triangulation Algorithms in Geoscientific Modeling.*, In Proceedings of the 3rd National Conference on GIS Research UK, Newcastle, UK, pp. 150–153.

35. Lawson, C. (1986). *Properties of n-Dimensional Triangulations*, *Computer Aided Geometric Design*, **3**: 231-246.

36. Lawson, C.L. (1977). *Software for C1 Surface Interpolation*, In: J.R. Rice (Ed.), Mathematical Software III, Academic Press, pp. 161–194.

37. Ledoux, H. (2006). *Modelling Three-dimensional Fields in Geo-Science with the Voronoi Diagram and its Dual*, Ph.D. Thesis, Advisor: C. Gold, School of Computing, University of Glamorgan, Pontypridd, Wales, UK.

38. Ledoux, H. (2007). *Computing the 3D Voronoi Diagram Robustly: An Easy Explanation*, In Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering, Pontypridd, Wales, UK, July 9-12, 2007, pp. 117-129.

39. Ledoux, H. (2008). *The Kinetic 3D Voronoi Diagram: A Tool for Simulating Environmental Processes*, In: P.V. Oosterom, S. Zlatanova, F. Penninga and E. Fendel (Eds.) Advances in 3D Geo Information Systems, Proceedings of the 2nd International Workshop on 3D Geoinformation, Delft, the Netherlands, December 12-14, 2007, Lecture Notes in Geoinformation and Cartography (LNCG), Springer-Verlag, pp. 361-380.

40. Ledoux, H. and Gold, C. (2007). *The 3D Voronoi Diagram: A Tool for the Modelling of Geoscientific Datasets*, In Proceedings of the GeoCongres 2007, Quebec, Canada (/http://www.gdmc.nl/publications/2007/3D_Voronoi_Diagram_Tool.pdf), octobre 2-5, 2007.

41. Ledoux, H., Gold, C. and Baciu, G. (2005). *Flipping to Robustly Delete a Vertex in a Delaunay Tetrahedralization*, In Proceedings International Conference on Computational Science and its Applications (ICCSA 2005), Singapore, Lecture Notes in Computer Science (LNCS), Vol. 3480, Springer-Verlag, pp. 737-747.

42. Maus, A. (1984). *Delaunay Triangulation and the Convex Hull of n Points in Expected Linear Time*, *Journal of BIT*, **24**: 151-163.

43. Mostafavi, M.A. (2002). *Development of a Global Dynamic Data Structure*, Ph.D. Thesis, Advisor: C. Gold, University of Laval, Laval, Canada.

44. Mostafavi, M.A., C.Gold and Dakowiczb, M. (2003). *Delete and Insert Operations in Voronoi/Delaunay: Methods and Applications*, *Journal of Computers and Geosciences*, **29**: 523-530.

45. Mostafavi, M.A. and Gold, C. (2004). *A Global Kinetic Spatial Data Structure for a Marine Simulation*, *International Journal of Geographical Information Science (IJGIS)*, **18**(3): 211-228.

46. Mucke, E. (1988). *A Robust Implementation for Three-Dimensional Delaunay Triangulations*, *International Journal of Computational Geometry and Applications*, **8**(2): 255–276.

47. Okabe, A., Boots, B., Sugihara, K. and Chiu, S.N. (2000). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2nd Edition)*, John Wiley.

48. Roos, T. (1991). *Dynamic Voronoi Diagrams*, Ph.D. Thesis, University of Wurzburg, Wurzburg, Germany.

49. Roos, T. (1993). *Voronoi Diagrams Over Dynamic Scenes*, *Discrete Applied Mathematics*, **43**(3): 243-259.

50. Roos, T. and Noltemeier, H. (1991). *Dynamic Voronoi Diagrams in Motion Planning*, In Computational Geometry: Methods, Algorithms and Applications, Proceedings of International Workshop on Computational Geometry (CG 91), Bern, Switzerland, March 21-22, 1991, Lecture Notes in Computer Science (LNCS), Vol. 553, Springer-Verlag, pp. 227-236.

51. Schaller, G. and Meyer-Hermann, M. (2004). *Kinetic and Dynamic Delaunay Tetrahedralizations in Three Dimensions*, *Journal of Computer Physics Communications*, **162**(1): 9–23.

52. Shewchuk, J.R. (2003). *Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips*, In Proceedings of the 19th Annual Symposium on Computational Geometry, San Diego, USA, ACM Press, pp. 181–190.

53. Su, P. and Drysdale, R.L. (1997). *A Comparison of Sequential Delaunay Triangulation Algorithms*, *Journal of Computational Geometry: Theory and Applications*, **7**(5): 361-385.

54. Tanemura, M., Ogawa, T. and Ogita, N. (1983). *A New Algorithm for Three Dimensional Voronoi Tessellation*, *Journal of Computational Physics*, **51**: 191–207.

55. Vomacka, T. (2008). *Delaunay Triangulation of Moving Points*, In Proceedings of the 12th Central European Seminar on Computer Graphics, Budmerice Castle, Slovakia, April 24-26, 2008, pp. 67–74.

56. Watson, D.F. (1981). *Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytops*, *The Computer Journal*, **24**(2): 167-172.

57. Yong, X., Sun, M. and Ma, A. (2004). *On the Reconstruction of Three-Dimensional Complex Geological Objects using Delaunay Triangulation*, *Journal of Future Generation Computer Systems*, **20**: 1227–1234.